Adopting agile methods for safety-critical systems development

Bruce Powel Douglass, Chief Evangelist, IBM Rational
Leslie Ekas, Development Manager, IBM Software Group
Because of their discipline and efficiency, agile development practices can be applied to the development of safety-critical systems. This paper addresses concerns that some traditional practitioners in the safety-critical space have about agile methods, and it shows how agile methods are well-suited for delivering safety-critical systems.

Agile methods have a reputation for being fast and adaptive but undisciplined and lacking in robustness. In fact, agile methods require a great deal of discipline, and these practices enhance both quality and team productivity. This paper examines how agile methods can be used in the development of safety-critical systems.

Properties of safety-critical systems
The term “safety-critical” refers to systems that either can cause harm or are responsible for preventing harm. Such systems range from medical devices to automotive braking, nuclear power plant control to avionic flight management systems. Most safety-critical systems must be certified by a regulatory agency to ensure that they are fit-for-purpose, that is, proper development practices have been applied to promote “system correctness” as the final outcome. It is important that adherence to the objectives of the relevant standards can be demonstrated. Because it is virtually impossible to demonstrate deterministic correctness for any significant piece of software, most of these standards have concentrated on specifying process objectives and requirements for evidence that the processes have been followed. DO-178C (Software Considerations in Airborne Systems and Equipment Certification), the recently released avionics standard, for example, requires that a system project supply evidence in several categories for up to 71 objectives, depending on the safety-criticality level. Supplements to this standard, such as DO-331 (Model-Based Development and Verification) and DO-332 (Object Oriented Technology and Related Techniques) add further objectives if those technologies are employed. These objectives relate to a number of topics, including planning (including the specification of the safety level of the device), software development process definition, requirements management, software design and coding. Other objectives relate to configuration management, quality assurance, integration, verification, tool qualification, and system certification.

It is important to note that, even though these standards specify the objectives that any process must meet if it is used to develop a safety-critical system, the standards do not specify the processes themselves. As long as a process can be demonstrated to meet the needs of the relevant standard, the development team is free to use whatever processes they desire. This leaves the option available to use agile methods, with their accompanying advantages, provided that the safety objectives can be achieved.
Traditional methods for developing safety-critical systems

Traditional methods, including the waterfall and V lifecycles, involve enormous effort in planning, documenting and reworking, but relatively little time in actual development. Early planning is often done to great detail, far more than is justifiable given available information at the onset of a project. Such detailed plans are inevitably in error and, when they are discovered to be incorrect, they require additional great effort to rework them. Verification of a work product, whether it is a requirements specification or target software, is far removed in time from its creation. This means that defects are embedded early and discovered late, with a huge cost in time and effort to identify and remove the defects and rework the artifacts. Great store is placed in separate analysis and thinking early on, but there is often little verification of that reasoning, at least until the end of the project. The primary means for “verification” is a manual, error-prone and expensive review process applied to each of the hundreds of work products produced during the project lifecycle. The typically high defect rate and frequency of missed deadlines attest to the inadequacy of the approach.

This is not to imply that deep reasoning about correctness and safety is misplaced. Clearly, this analysis must be done, but only after the appropriate information to support it is made available. And, more importantly, the analysis should be verified as the work is being performed. Agile holds that this work should be done in a way that discovers and repairs defects immediately so that the emphasis is on defect avoidance in the product itself, rather than production of documentation.

Why agile?

In general, there are two reasons why teams move to agile methods. The first is a need to improve quality, aspects of which include customer acceptability, usability, (low) defect rates and compliance with relevant standards. The second reason is to save development time and costs.

Frequently, the improved quality that’s achievable through agile methods gets lost in all the hype about agile itself. Many of us have encountered people and teams who claim to be “agile,” but who, in fact, are just making this claim to avoid planning, documentation and design. If agile methods were simply about avoiding those aspects of software and systems delivery, then agile methods would be completely inappropriate for safety-critical systems development. In reality, that is a misrepresentation of the intent and the practice of agile methods.

What we mean by the term “agile” is a highly disciplined approach to software or systems development, where you try to avoid defects by getting the development right the first time, and by doing things in such a way that you can verify them as you are doing them. By contrast, traditional methods insist on a long period of feature creation followed by an even longer period of stabilization (which can amount to long periods of “bug insertion” and “bug removal”).

Many organizations working in the safety-critical area—whether it is medical, automotive, transportation, aerospace, or military systems—use traditional, heavyweight approaches with lots of
Adopting agile methods for safety-critical systems development

paper, lots of manual processes and a variety of disparate and often non-integrated tools, including documentation through word processors and spreadsheets. These traditional processes require considerable due diligence, creating a huge burden on the engineering staff; and they are often hugely expensive. With agile methods you can incrementally create verifiable artifacts that can then be mathematically analyzed, simulated or tested on the target platform. The construction and verification of components during system development is a key benefit that agile brings; this iterative process offers the sort of quality improvement needed in safety-critical systems at a (relatively) low price.

In safety-critical development, the key concern is safety, and in agile methods, the paramount concern is quality. There is no contradiction here because by paying careful attention to quality, safety can be improved and managed. It makes sense, then, to adapt agile methods to the specific needs of safety-critical projects. This enables the benefits of agile to be realized in such demanding development environments. It should be noted that there are also secondary benefits to using agile methods, including improved productivity, improved time to market, improved customer satisfaction and decreased development costs, but in the context of safety, the additional benefits are of lesser importance. What is important is tailoring the various agile practices to make them applicable to the needs of safety-critical systems development.

Traditional methods may appear (to some, at least) as more disciplined than agile because the governance they require produces reams of documentation, hundreds of hours of management overhead and built-in time delays to handle the inevitable scrap and rework that comes as the project nears completion. However, poor planning and execution is not the result of actual discipline.

The overhead of ostensibly disciplined traditional methods has little to do with creating executable, verifiable software because the development and testing aspects of traditional methods (such as are found in waterfall techniques) are “open loop” with verification and validation occurring far later in the process. Typically, in traditional methods, teams do not work closely together. Much of the cost and overhead of traditional methods arises because they produce such poor quality output. Without extensive review and rework, the delivered products simply cannot be deployed because they don’t meet the requirements. Performance is rarely robust and reliable and defect rates are often far too high. All of these problems mean that, in a safety-critical context, traditionally developed systems may have the potential to cause significant harm.

Practices needed for agile development
In contrast to traditional approaches, agile methods are, by their very nature, more disciplined. They’re focused on defect avoidance and the continuous creation of executable, verifiable code. The code is delivered frequently for stakeholder review throughout the project. Instead of the reams of paper to “tell” what has been accomplished, the more disciplined agile process itself enables teams to “show” what’s been accomplished, step by step, as the project progresses. Thus, the rigor in agile comes from
producing working, verified code throughout the project, requiring teams to solve problems as they are encountered. And agile teams are usually cross-functional, ensuring they take advantage of their collective expertise from the beginning of the project.

Some of the most important agile practices that impact safety-critical systems development are:

- Incremental development
- Test-driven development
- Continuous integration
- Dynamic planning
- Risk management

**Incremental development**

One major change in project management style from waterfall to agile is the use of “incremental development,” which helps teams handle complexity better than ‘big design up front’ (BDUF). Many teams create prototypes (early system versions created for a specific purpose such as “proof of concept”), then spend a great deal of time doing detailed design followed by—at some point in time, often far in the future—coding. The challenge of the BDUF practice is that it creates a rigid, brittle architecture; any issue with the architecture discovered well into the development cycle is usually hard to fix and tends to be only superficially remediated because a better engineered solution would be too costly. Agile offers an alternative: it is better for teams to remain in “prototype mode,” using evolutionary design to take thin, vertical slices of the architecture and start to build the product sooner. Engineers find their biggest problems earlier, as opposed to discovering them late in the project, when coding is supposed to be nearing completion.

Note that the meaning of the word “prototype” here changes between the two approaches. In a traditional approach, prototype means a roughed out implementation that is eventually thrown away. In an incremental design approach, prototype means an initial implementation that evolves incrementally into the final, released system. With agile, you start with a thin sliver of functionality that crosses the proposed architecture stack and build the most minimal implementation. The interfaces between the components in the architecture stack are tested immediately, so teams are able to discover and eliminate the sources of many of the big problems.

As the project progresses, more functionality is added, while teams continuously verify the work as it is being done. The system capabilities can be elaborated on a highest-risk first basis or by other criteria, such as urgency, criticality, or availability (for example, availability of some resource or subject matter expert). Continuous verification ensures low defect rates as the system prototypes evolve. Continuous integration guarantees that the work products from different teams properly assimilate. And although reviews can be done, they become a secondary measure of adding quality, not the primary one.

Do you think this general approach is new? There are decades of research supporting the effectiveness of incremental development, and the literature has been advising teams to take this “new” approach for a long time now. These iterative, incremental ways of working are significantly better at producing software with fewer defects in less time than serial waterfall approaches.
Test-driven development

Test-driven development (TDD) is another fundamental agile practice. The idea is to only produce work products that are free of defects. It is carried out through a process of very short (20 - 60 minute) cycles in which some portion of a work product is produced and verified. This most commonly applies to source code. A key aspect of TDD is that you write the test before you write the code to be tested. If defects are found, they are immediately fixed before more functionality is added.

The result of this approach is that you write “just enough code” to pass the test versus writing additional code that may or may not be necessary but will need to be tested. Well-formed tests that target the required functionality will produce more sustainable code because there are fewer code paths to manage.

In the IBM® Rational® Harmony process, this is known as the nanocycle. Figure 1 shows the Rational Harmony nanocycle workflow. The important thing to remember is the speed at which this cycle runs; 20 minutes per cycle is not uncommon.

Although this discussion has focused on source code, any verifiable work product can use this TDD approach. The Rational Harmony process focuses on the creation of executable work products for this reason, including requirements models, architectural models and design models. TDD-style unit testing is highly recommended because of the huge impact it has on quality.

Continuous integration

Continuous integration is, in a sense, a communal TDD practice. Its purpose is to avoid costly integration problems identified late in the development process. Similar to TDD, Continuous integration accomplishes this goal by applying defect identification and removal early and often. Most commonly, this practice results in a daily build in which the work of multiple engineers is
linked, and test cases that cross component boundaries are run. These tests become more complete and elaborate as the complexity of the components grows. When integration problems are discovered, they are immediately fixed before new features can be added. In this way, the software always integrates. This practice greatly reduces the integration time at the end of the project over more traditional methods.

The key benefit of continuous integration is that it enables everyone on the team to be working on the same build; the team can therefore make the most of its collective skills to manage quality problems as they occur. Creating a well running continuous integration system might be costly up front for teams that have not invested in tuning their builds or test automation, but the payoff for this investment is higher productivity.

Dynamic planning
Most traditional projects are managed according to a detailed plan drawn up early in the project, which makes hundreds or thousands of assumptions based on limited information, many of which may be optimistic, and many of which may turn out to be incorrect. In comparison, agile approaches recognize that plans can and must change as more information becomes available during a project and therefore replace much up-front planning with a more evolutionary approach.

In fact, agile planning happens at the beginning of each iteration so it can include changes resulting from knowledge gained in the previous iteration. Some guiding principles for agile planning are:

- Do not plan beyond your level of available information.
- Put into place metrics that continuously measure progress and success based on outcomes.
- Use the outcomes of the work to update plans frequently during the project.

Planning is crucial to understand what resources you need, how many (or much) of them you need, when you need them and what your outcomes will be. However, software development is largely a matter of invention and predicting invention is fraught with inaccuracies. By putting in place a plan with clear expectations, by continuously measuring progress versus desired outcomes and by adjusting the plans to meet reality, better project success can be achieved.

Risk management
The last practice is risk management, which refers to project risk, not user risk. Risk is a function of the unknown. So, project risk management is about identifying dangerous lapses in knowledge and acting (known as “spikes” in Scrum or “risk mitigation
activities” in Rational Harmony) to improve understanding and thereby reducing risk. Risks might be technology related (“I'm not sure if CORBA is fast enough to meet our communication needs”), user-based (“Will pilots accept this kind of interface in their cockpit?”), or managerial (“Can this milestone be achieved before the deadline?”). The answer to such risk concerns is information. Risk is usually managed with a risk list or risk management plan which typically tracks the following key items for each identified risk:

- Quantified impact and likelihood
- Risk strategy (acceptance, avoidance, mitigation, transference)
- Responsible parties for risk strategy
- Status and results

Agile methods emphasize risk management because the lack of it is the most common cause of project failure. Agile manages technical risk by frontloading the work backlog with high risk items. For each challenge, a thin sliver of the code from the risky technology is prototyped and tested to determine if the solution works before the rest of the code is completed. This process is repeated for all risky parts of the code at the beginning of the project. The risk of not building the right product is handled by providing customer demos at every iteration. Planning at the beginning of each iteration enables teams to repeatedly evaluate their highest risk issues and decide how to handle them as they gain the information required.

**Agile practices for safety-critical development: A recap**

The agile practices described here do not avoid useful work, or promote undisciplined coding instead of meeting requirements. Agile practices primarily avoid tasks that add little or no value. They emphasize activities that provide high value, and they reorder tasks to address risk as rapidly as possible.

Safety-critical systems development has special needs beyond those of most projects. In such projects, additional practices to address those needs are included, such as:

- Initial safety analysis
- Continuous safety assessment
- Continuous traceability analysis
- Change management
- Requirements-based verification

Initial safety analysis—using techniques such as fault tree analysis, failure mode and effect analysis, and hazard analysis—looks at the specific safety concerns of the system, codifies the safety risk and identifies additional requirements for safety control measures. As the project progresses, technological solutions to the requirements are realized. These solutions might introduce
further safety concerns, so an ongoing assessment is required to determine the need for additional safety control measures. Traceability analysis is required by most safety standards and it enables the development team to make its safety case, showing how all requirements are realized in the design and code and how they are adequately tested.

Traceability connects the different work products into a cohesive, coherent whole. The focus is on “continuous traceability” because adding traceability after the fact is error-prone and expensive. Traceability is far better achieved by adding traceability links as the work elements are being created rather than in a late “clean up” phase. Change management evaluates the impact of changes to work products (such as requirements or design) using traceability as a guide. This ensures that the effect of changes on system performance and safety are well understood and acceptable. Lastly, requirements-based verification uses detailed traceability to ensure all requirements are satisfied by design and code and adequately covered by test cases, and there is no implementation for which there are no requirements.

Adherence to safety objectives is neither cheap nor easy. Atego HighRely, a DO-178 consulting firm estimates that certification under the DO-178B standard adds at least 25 - 40 percent to total project costs, and it can be as high as 75 - 150 percent.³

It is not advisable to begin the certification process at the end of the project. This typically results in unplanned rework, plus cost overruns. Costs are likely to be lower if certification agencies are consulted early. In DO-178 projects, the key document is the Plan for Software Aspects of Certification (PSAC), which details the proposed approaches, tasks and work products that will be used to gain certification. Early exposure of such a document to the certification agency enables the project team to incorporate its feedback early, thus helping to reduce the amount of rework.

In an agile project, the PSAC and supporting plans (such the quality assurance plan, configuration management plan and software verification plan) or their equivalent help identify the practices that will be employed, the work product content and structure. They also outline how all of the relevant safety objectives will be met. Just as with all other agile work products, this is best done incrementally with frequent verification (with the certification agency) as the project progresses.
Adopting agile for safety-critical projects

Agile methods can help provide significant benefits in terms of quality and team productivity, but their adoption is neither easy nor pain-free, and it requires effort and thought to do it properly.

The steps to successful agile adoption are:

1. Identify where you are. Assess your strengths and weaknesses as a development organization. Your assessment should include not only software development, but also other engineering disciplines (for example, electrical, mechanical, human-machine interface, and systems engineering), project planning, project governance, quality assurance and testing.

2. Identify where you want to be. Do not state what you want the process to be, but state the objective outcomes and goals you want to achieve, such as dependability, quality and stakeholder satisfaction.

3. Identify the gaps. Assess where you are and where you want to be as an engineering organization.

4. Identify measures of success. Objective, goal-based metrics can be used to assess the success of changes made to your processes and practices. These metrics give you evidence, rather than perceived or hoped-for results.

5. Plan incremental remediation. A good remediation plan uses agile practices to adopt agile practices. Identify specific practices and methods that address the high priority concerns (identified in Step 3) with goal-based metrics that allow continuous monitoring of progress.

6. Implement that plan dynamically. By “dynamic” we mean using the objective, goal-based metrics to steer the adoption of the practices to ensure their optimal benefit. Remember: a plan is a theory and theories need to be adjusted to account for reality.

Conclusion

Safety-critical systems are difficult to develop. In addition to normal concerns about quality and time-to-market, safety-critical systems must also meet the demanding objectives of relevant safety standards and are subject to rigorous certification. Agile methods are a set of practices that can help improve both quality and productivity and can be employed in the development of safety-critical systems as well. The common agile practices apply well to safety-critical systems, but they must be tailored and customized to ensure that safety objectives are met.

The key agile practices that can assist in the development of safety-critical systems are incremental development (evolutionary development with frequent requirements-based verification), test-driven development (development and application of unit tests as the code is developed), continuous integration (continuously building software and verifying the various components work together properly), dynamic planning (updating plans based on continuously measured “ground truth”) and risk management (identifying and prioritizing project risks and reducing them through risk strategies). In addition, the traditional safety-critical practices of initial safety analysis, continuous safety assessment, managing traceability links among work products, change management and requirements-based verification can ensure an efficient, high-quality development process.
About the authors

Bruce Powel Douglass, who has a doctorate in neurocybernetics from the USD Medical School, has over 30 years experience designing safety-critical real-time applications in a variety of hard real-time environments. He has designed and taught courses in agile methods, object-orientation, MDA, real-time systems, and safety-critical systems development, and is the author of over 5000 book pages from a number of technical books including Real-Time UML, Real-Time UML Workshop for Embedded Systems, Real-Time Design Patterns, Doing Hard Time, Real-Time Agility, and Design Patterns for Embedded Systems in C. He is the Chief Evangelist at IBM Rational, where he is a thought leader in the systems space, consulting with and mentoring IBM customers all over the world, represents IBM at many different conferences, and authors tools and processes for the embedded real-time industry. He can be followed on Twitter @BruceDouglass. Papers and presentations are available at his Real-Time UML Yahoo technical group (http://tech.groups.yahoo.com/group/RT-UML) and from his IBM page (www-01.ibm.com/software/rational/leadership/thought/brucedouglass.html).

Leslie Ekas joined IBM from the FileNet acquisition, bringing 20 years of software development and management experience. Leslie started practicing Scrum and applying Lean techniques to improve the effectiveness of her team, the quality of the software and the value delivered to our customers. After successfully adopting Agile, she became the first coach in the Agile/Lean Center of Competence. As a part of the SWG Strategy and Technology team, this group accelerated the successful adoption of Agile/Lean practices throughout IBM’s Software Group. She moved back into Development as the world wide manager for Rational Development, and she is now leading a Tivoli governance team for the Smarter Physical Infrastructure products to adopt business agility.
For more information
To learn more about the agile practices for safety-critical products, please contact your IBM marketing representative or IBM Business Partner, or visit the following websites:
ibm.com/software/rational/agile/embeddedagile/ or
ibm.com/software/rational/services/harmony/

Additionally, IBM Global Financing can help you acquire the software capabilities that your business needs in the most cost-effective and strategic way possible. We’ll partner with credit-qualified clients to customize a financing solution to suit your business and development goals, enable effective cash management, and improve your total cost of ownership. Fund your critical IT investment and propel your business forward with IBM Global Financing. For more information, visit:
ibm.com/financing