IBM Software
Thought Leadership White Paper

December 2014

Adopting agile methods for safety-critical systems development

IBM

Executive summary

Agile methods are known for their speed and flexibility, but they also have an undeserved reputation for being undisciplined and lacking robustness. In fact, agile methods require a great deal of discipline, and enhance both quality and team productivity.

This paper explains how the inherent discipline and efficiency of agile development practices makes them ideally suited to the development of safety-critical systems.

Properties of safety-critical systems

The label "safety-critical" is generally applied to systems that either can cause harm or are responsible for preventing harm. Such systems range from medical devices to automotive braking systems, and from 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. This includes verifying that 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.

As it is virtually impossible to demonstrate deterministic correctness for any significant piece of software, most standards focus on specifying process objectives and requirements for evidence that the processes have been followed. For example, the recently released avionics standard DO-178C (Software Considerations in Airborne Systems and Equipment Certification) 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, covering 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 shown to meet the requirements of the relevant standard, the development team is free to use whatever processes they wish. Provided that the safety objectives are achieved, this means that teams are free to benefit from the advantages of using agile methods.

Traditional methods for developing safety-critical systems

Traditional methods, including the waterfall and V lifecycles, involve enormous effort in planning, documentation and rework, but relatively little time in actual development. This is sometimes referred to as 'big design up front'. Early planning is often done in far more detail than is justifiable given the information that is typically available at the outset of a project. Planning at this level of detail means that plans inevitably contain errors. When these errors are discovered, a great deal of additional effort is required to rectify them. Likewise, the verification of any given work product, whether it is a requirements specification or target software, is typically far removed in time from its creation. This means that defects are embedded early and discovered late, implying a huge downstream cost in time and effort to identify and remove the defects and rework the artifacts. Great emphasis is placed on 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 typical high defect rate and high 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, such 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 performed. Agile development principles state that this work should be done in a way that discovers and repairs defects immediately, such that the emphasis is on defect avoidance in the product itself, rather than on the production of documentation.

Why agile?

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

Frequently, however, the improved quality achievable through agile methods gets overshadowed by the surrounding hype and misinterpretations of the term. Some people and teams claim to be agile, but are simply skipping the planning, documentation and design stages. If this behavior truly represented agile methods, then clearly agile would be completely inappropriate for safety-critical systems development. In fact, the correct definition of agile is a highly disciplined approach to software or systems development, in which developers attempt to avoid defects by getting the development right the first time, and by verifying work as it is created.

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 for medical, automotive, transportation, aerospace, or military systems—use traditional, heavyweight approaches involving reams of paper, extensive manual processes and a variety of disparate and often non-integrated tools, including documentation created in 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, teams 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 of agile practices; this iterative process can deliver the quality needed in safety-critical systems at a relatively low cost. 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, so that the benefits of agile can be realized in these 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. However, in the context of safety, these additional benefits are naturally 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 be more disciplined than agile—if only because their governance can require reams of documentation, hundreds of hours of management overhead and built-in time delays to handle the inevitable scrap and rework as the project nears completion. But in reality, all of this additional material and work may produce only the illusion of discipline, and does not necessarily contribute to creating executable, verifiable software.

In contrast to traditional methods (for example, waterfall techniques), verification occurs early in the agile process, alongside development, and teams work closely together to ensure that the delivered products meet the requirements. This enables agile approaches to avoid much of the additional cost and overhead that would otherwise arise from the need to rework sub-standard output. For products created in an agile environment, performance is robust and reliable, and defect rates are low. Compared to traditionally developed systems, where defect rates are typically significantly higher, systems developed in an agile approach are more likely to quickly meet with approval in a safety-critical context.

Practices needed for agile development

Agile methods promote greater discipline than traditional approaches, in that they focus on defect avoidance and the continuous creation of executable, verifiable code. The code is delivered frequently for stakeholder review throughout the project. Rather than relying on reams of paper to "tell" what has been accomplished, the more disciplined agile process itself enables teams to "show" the accomplishments, step by step, as the project progresses. In this way, the rigor inherent in agile stems from producing working, verified code throughout the project, which requires teams to solve problems as they are encountered. Equally, agile teams are usually cross-functional, and can therefore draw on a broader range of collective expertise right from the outset of the project.

This paper will now outline five of the most important agile practices that impact safety-critical systems development:

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

Incremental development

One major difference in project management style between traditional and agile approaches is the use of incremental development, which can help teams handle complexity better. In a traditional approach, many teams create prototypes (early system versions created for a specific purpose, such as a proof of concept), then spend a great deal of time on detailed design. Only much later, will they actually get down to coding. The challenge of this 'big design up front' practice is that it creates a rigid, brittle architecture. Any architectural issues discovered later in the development cycle will usually be hard to fix—and the prohibitively high cost of creating a better-engineered solution makes it likely that remediation will be merely superficial. Agile offers an alternative, enabling teams to remain

in prototype mode and use evolutionary design to take thin, vertical slices of the architecture and start to build the product sooner. Engineers discover the major problems earlier, rather than late in the project, when coding is meant to be nearing completion.

There is an important difference in the meaning of the word "prototype" as used within the two approaches. In a traditional approach, "prototype" means a roughed-out implementation that is ultimately discarded. In an incremental design approach, "prototype" means an initial implementation that evolves incrementally into the final, released system. Agile teams start with a thin slice of functionality that crosses the proposed architectural stack, and then build the smallest possible implementation. The interfaces between the components in the architectural stack are tested immediately, so teams are able to discover—and address at source—many of the major problems. As the project progresses, more functionality is added, and 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, the 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 will assimilate correctly. Although reviews can certainly be done, they become a secondary means of adding quality, not the primary one. The incremental approach is well established, and decades of research and practical experience attest to its effectiveness in producing software with fewer defects and in less time than serial waterfall approaches.[1]

Test-driven development

Test-driven development (TDD) is another fundamental agile practice. The idea is to produce only work products that are free of defects. It is carried out through a process of very short (20- to 60-minute) cycles in which some portion of a work product is produced and verified. This most commonly applies to source code. A key tenet of TDD is that the test should be written before the code that will be tested. If defects are found, they should be immediately fixed, before more functionality is added. The result of this approach is that teams write "just enough code" to pass the test, not wasting time and effort writing additional code that may or may not be necessary but will certainly need to be tested. Well-formed tests that target the required functionality will produce more sustainable code because there will be fewer code paths to manage. In the IBM® Harmony/Agile for Embedded Software Development process[2] (hereafter referred to as the IBM Harmony/Agile process), this is known as the nanocycle. Figure 1 shows the IBM Harmony/Agile nanocycle workflow. The important thing to note 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 IBM Harmony/Agile 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 positive impact it has on quality.

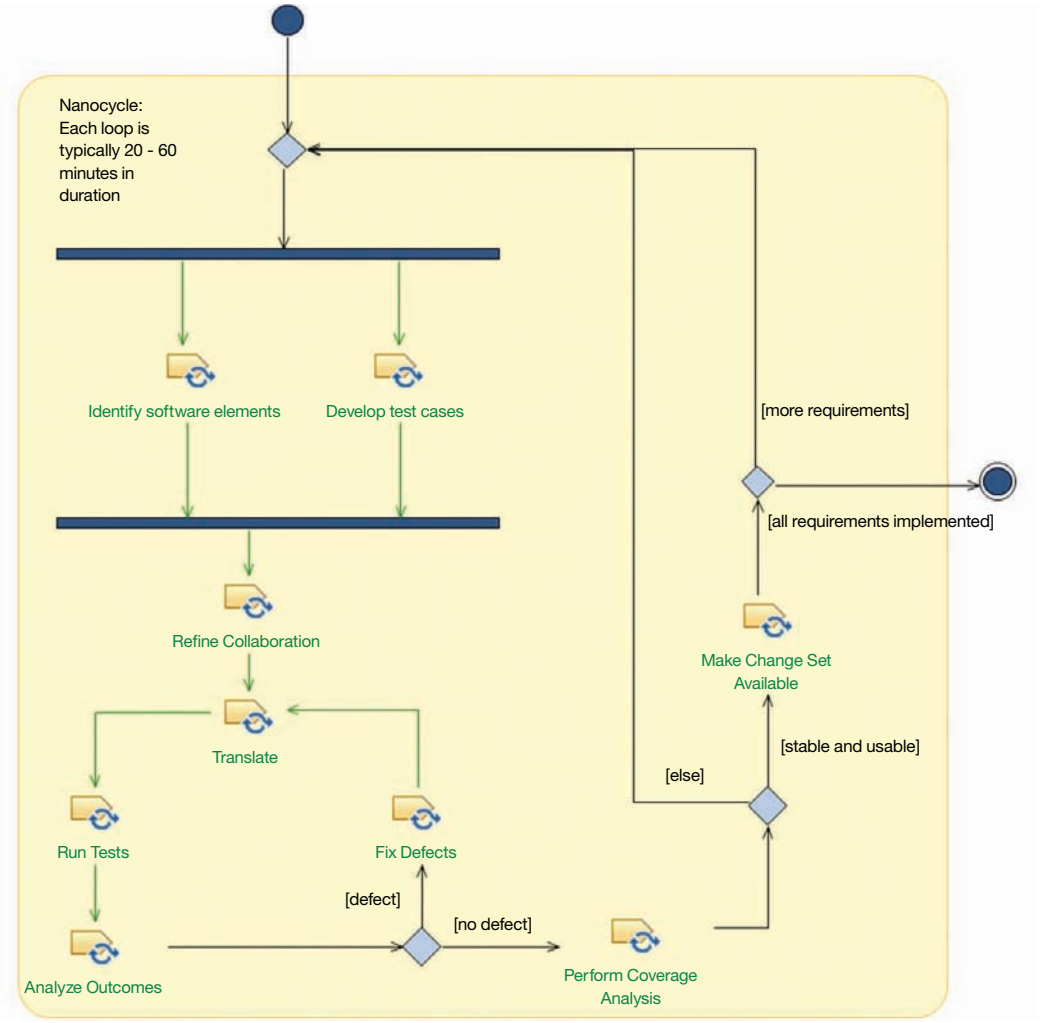

Figure 1. The IBM Harmony/Agile nanocycle

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 in the running of test cases that cross component boundaries. These tests become more elaborate and complete 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 compared with 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 smoothly 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 significantly 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. They therefore replace much up-front planning with a more evolutionary approach.

In point of fact, agile planning happens at the beginning of each iteration so that 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. Many teams feel uncomfortable moving from traditional planning, with its apparent certainties, to agile planning, where timescales and deliverables are more clearly estimates. However, software development is largely a matter of invention, and making accurate predictions about invention is difficult. By putting in place a plan with clear expectations, by continuously measuring progress versus desired outcomes and by adjusting the plans to meet reality, greater confidence can be gained in predictions of deliverables and outcomes, and better project success can be achieved.

Risk management

The last practice is project risk management. Risk is a function of both known and unknown factors, so project risk management is a question of identifying potentially dangerous lapses in knowledge and then taking specific actions. These actions—known as “spikes” in Scrum or “risk mitigation activities” in IBM Harmony/Agile—improve understanding and thereby reduce risk. Risks might be technology related (for example, “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 lies in the discovery, dissemination and collective understanding of information. Risk is usually managed using 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 this discipline is among the key contributors to project success. Agile manages technical risk by frontloading the work backlog with high-risk items. For each challenge, a thin slice of the code from the technology thought to be risky is prototyped and tested to determine whether 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

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, and for safety-critical projects agile includes additional practices to address those needs, 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—required by most safety standards—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"—that is, traceability which is created at the same time as work products are created or modified—because adding traceability after the fact is error-prone and expensive. 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 that their acceptability can therefore be determined. Lastly, requirements-based verification uses detailed traceability to ensure that all requirements are satisfied by design and code, are adequately covered by test cases, and that there is no implementation for which there are no requirements.

Safety-critical practices: regulations and compliance

Safety-critical systems must be certified against their applicable standards. These standards identify the objectives that a system or project must meet before it is allowed to be deployed in its operational environment. For example, avionics software must be certified against DO-178B (or the newer DO-178C) and its associated supplements. Medical software is certified against IEC 62304. Nuclear power plants are certified against IEC 61513. Passenger automobiles are certified against ISO 26262.

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 to 40 percent to total project costs, and the premium can be as high as 75 to 150 percent.[3]

It is not advisable to begin the certification process at the end of a 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 release of such a document to the certification agency enables the project team to incorporate the agency's feedback early, thus helping to reduce the amount of rework. In an agile project, the PSAC and supporting plans (such as the quality assurance plan, configuration management plan and software verification plan) help identify the practices that will be employed and 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 provide significant benefits in terms of quality and team productivity, but their adoption is neither easy nor pain-free. The recommended steps to successful agile adoption are:

1. Identify where you are. Assess your strengths and weaknesses as an engineering 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. This means using objective, goal-based metrics to steer the adoption of the practices to ensure their optimal benefit. A plan is really a theory, and theories need to be adjusted continually to account for reality.

IBM solutions for adopting agile for safety-critical systems development

The IBM solution for systems and software engineering provides a collaborative lifecycle solution for agile development. By integrating support for the different activities across the systems and software delivery lifecycles the IBM solution helps to avoid siloes of activity and enables teams to work collaboratively. The solution supports key engineering activities including requirements management, model-based architecture and design, testing and quality management, change and configuration, and collaborative planning and workflow management. It is integrated with the IBM Harmony/Agile process and its variants to support the objectives of industry-specific standards. Furthermore, the solution is extensible through open standards to support IBM and third-party domain- and industry-specific needs.

Conclusion

Safety-critical systems are difficult to develop. In addition to addressing 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 that can also be employed in the development of safety-critical systems. The standard 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 that the various components work together properly)
- Dynamic planning (updating plans based on continuously measured "ground truth")
- 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 35 years' experience designing safety-critical real-time systems 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 6,000 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. He represents IBM at many different conferences, and authors tools and processes for the embedded real-time industry. He can be followed on Twitter @IronmanBruce. Papers and presentations are available at his Real-Time UML Yahoo technical group (http://yhoo.it/1EXhJda) and from his IBM page (http://ibm.co/11hr5mR).

Jonathon Chard, who has a doctorate in electrical engineering from the University of Manchester, UK, has over 25 years' experience in the development of products and systems. He is currently responsible for the worldwide marketing strategy for IBM's solutions for real-time and embedded software development including agile approaches for product and systems development. Previously with IBM and formerly with Telelogic, he has held pre- and post-sales technical roles covering the product and systems development lifecycle including requirements management, software and systems modeling, code quality and testing, working with a variety of product and systems industries. In his earlier career, Jon was a systems engineering practitioner in the automotive industry.

For more information

To learn more about agile practices for safety-critical products, please contact your IBM marketing representative or IBM Business Partner, or visit the following pages:

- **ibm.com**/software/rational/agile/
- ibm.co/1rWL1Tb

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

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
December 2014

[1] See, for example, Barry Boehm's *Software Engineering Economics* (Prentice Hall, 1981) or *Balancing Agility and Discipline* (Addison-Wesley, 2003).

[2] The IBM Harmony/Agile process integrates other practices such as high-fidelity modeling (along with source code generation) and continuous integration in the development of software designs and source code. See Bruce Powel Douglass, *Real-Time Agility*, (Addison-Wesley, 2009) for more information.

[3] *DO-178B Cost and Benefits: what are the true DO-178B costs and benefits; a detailed analysis.* Vance Hildeman, Atego HighRely. (http://highrely.com/whitepapers.php)

Please Recycle

RAW14313-USEN-01