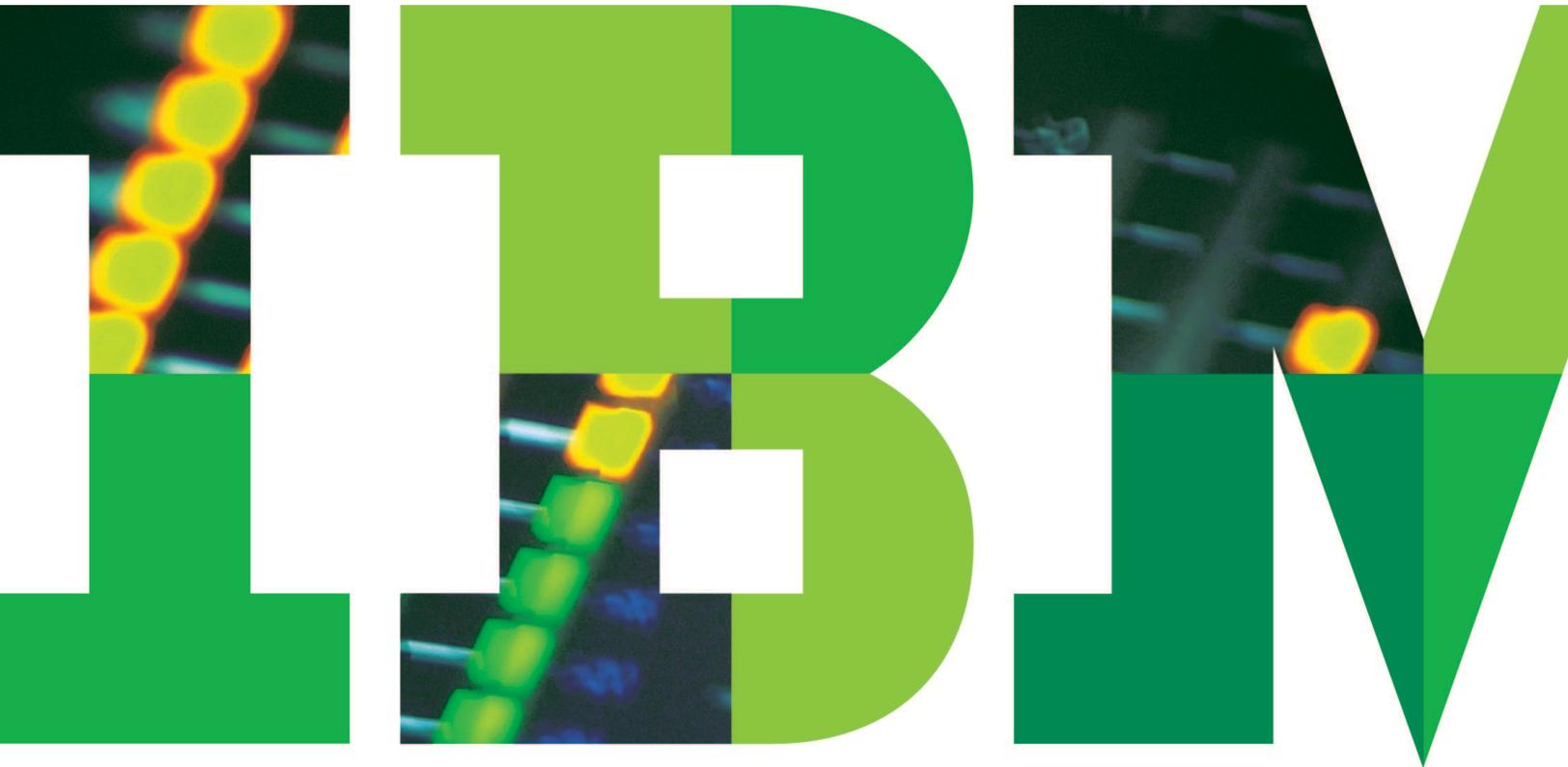


Five benefits of agile practices in software-intensive systems development

Achieve enhanced productivity, predictability, and flexibility



For the past decade, agile methods have proved beneficial for helping Information Technology (IT) software teams deliver, on schedule, high quality software that satisfies stakeholder needs. Software teams pursue agile methods because they need a process that can respond efficiently to change to the product under development. Agile methods afford more flexibility compared to traditional plan-driven approaches, which lock in the project details early and are less able to adjust to stakeholders' evolving needs, market changes, and unplanned technology challenges.

Software-intensive and real-time and embedded systems teams (software-intensive systems teams) face many of the same challenges. But can agile methods work for them?

Different constraints define how traditional IT software teams and software-intensive systems teams develop their products. In this context "products" means both software systems written for IT operations as well as devices and systems that depend on embedded software at the core of their capabilities, such as cell phones, automobiles and jet fighter planes. Teams that co-develop software and hardware must ensure the combined solution works on specific hardware/software configurations and delivers the required real-time performance. IT software teams must typically deliver a software solution on a variety of platform configurations (hardware, operating systems, etc.) and meet

performance requirements for each. Agile practices can be applied to both types of product development, despite their differences.

As markets get more competitive, products get more sophisticated, and ensuring quality becomes a more significant challenge, software-intensive systems teams need to evolve their development methodologies in order to succeed. With the appropriate tooling support, agile methods can deliver a successful solution and add significant benefits, which will be explained in detail in this paper:

- Agile methods can work for teams governed by industry standards because they put into practice a regular cadence of time-boxed development phases called "iterations." This requires teams to establish and achieve a set of "done" criteria for each iteration that can be used to track progress toward final delivery.
- The agile focus on not accumulating technical debt and "delivering just enough" helps drive overall quality improvements.
- Project timelines become more predictable because risk is better managed through visible, prioritized backlogs and parallel risk assessment methods.

- As teams adopt agile methods, they learn to work together as a whole team to deliver working software at every iteration, which results in a high productivity profile throughout the project.
- Tools that work together seamlessly throughout the lifecycle, from the requirements to the delivered product, further enhance team collaboration and productivity.

These benefits have been successfully realized in IT software development and can also be enjoyed by software-intensive systems development organizations.

If your development organization has already invested heavily in tools and development processes, how do you safely and cost-effectively evolve your development processes to adopt agile practices? This paper will answer that question by examining five key business benefits in moving to agile practices.

Five key business benefits of agile

1. Streamlined process overhead

Teams working to create products regulated by industry standards including DO-178B, ISO 26262 and ISO 62304 must be able to demonstrate compliance with those standards. These types of teams often adopt considerable process overhead to ensure that they meet stringent code mandates. Agile methods can help them meet industry standards with less overhead using short, time-boxed iterations. At first glance, an agile approach may appear to be “process light,” because a limited set of

practices is recommended. On closer examination, we see that these practices are lean, and because each must be completed with each iteration, the regular delivery pattern established helps ensure that critical process steps are completed on time and consistently. At first, teams beginning to adopt agile practices may feel the new method is more process intensive, because the same steps are executed repeatedly. Instead of completing the planning activity at the beginning of the process, and constantly readjusting the plan, agile practice requires planning at each iteration while allowing the plans to self-adjust as the project progresses and issues are discovered. Teams that become familiar with the lean but repetitive processes drive better predictability at every iteration.

Up-front plans, especially detailed ones that span the entire lifecycle in traditional projects, are expensive to update as requirements change, and they typically need to change numerous times throughout a project. Agile planning, by contrast, takes into account the likelihood that plans will change as the software takes shape over the course of the project, so detailed plans are avoided at the outset. Change is managed as one of many factors that determine project outcome. The net benefit is a process that:

- Can adapt to the changes that inevitably arise.
- Requires less end-to-end process overhead.
- Leads to less rework as the project’s end-date draws near.

Traditional projects often require a resource just to track the schedule. This is not needed for agile projects

The elements of iterative planning

As noted above, an agile development process divides the project timeline into short, time-boxed iterations. In the context of software development, an “iteration” is a phase during which a functional part of the software under development is created and tested. Iterations are designed for whole teams to complete planned work. Whole teams include everyone who is required to bring work to completion, and they typically include developers to write the code, testers to validate the code, builders to compile the code (i.e., make it functional and testable), writers to document how to use the code, and so forth. In the case of systems software this would also include the hardware team. For each iteration, the whole team targets completion of the planned work. Most product teams include all the members of each component who contribute to the product. The distinction in agile development is that these component teams work together as a “whole team” throughout the product lifecycle, which means that the planned work is an integration of the components.

In an ideal agile scenario, teams should be able to complete and ship their work at each iteration. But that is rarely achievable with large projects, especially when the product includes embedded systems. Nevertheless, the goal of completing

“shippable” functionality with each iteration is important, because stakeholders need to review those functions and provide early and regular feedback during the project lifecycle. The iterative process affords this sort of review, and allows for midstream course correction as needed. Experienced agile teams target work that is achievable and relevant for each time-boxed iteration across the product stack. In other words, they aim to complete a thin slice of work that includes the entire product stack so that some portion of the evolving product starts to work, and the team can identify the critical integration issues early. Working this way from the beginning of the development cycle helps identify cross-product issues early and verify that the planned architecture works. It is worth noting, however, that software-intensive systems teams may find it difficult to be able to complete a thin enough slice of work that crosses the software/hardware boundaries in a single iteration. They may find it necessary to use techniques such as software/hardware simulation to provide the mechanism to work across the boundaries at each iteration.

Challenges and benefits of whole-team collaboration

Building the product across the product stack at the beginning of the development cycle is a challenging way for teams to work. In traditional software and hardware development projects, the

component teams develop in isolation at the beginning of the project. Design, integration points, and interfaces are completed early in the project then each component team typically develops separately to meet the design specifications. By contrast, agile methods prescribe working across the architecture boundary from the beginning of the project. Instead of completely developing individual components that are expected to work together based on their pre-established interfaces, agile practice recommends building thin slices of functionality that cross the architecture each iteration. Whole teams of people from across the architecture work together from the beginning of the project. A critical advantage is that the team tests the architecture through integration early in the project and can make adjustments when the impact is the smallest. This presents a challenge for IT project teams, who are not accustomed to breaking down work down in this fashion. It poses the same if not harder challenges for software-intensive systems teams, because they have to integrate across the software and hardware boundaries.

Nevertheless, integration risks are high in hardware and software co-development efforts, and iteration-based whole-team integration can potentially yield great benefits to these teams since their integration challenges are significant. Once teams get experience working in this manner, they typically discover issues very early and see the value of continuing with these agile

practices. Engaging the whole team to work together across the product stack may delay finalizing design and implementation decisions, but it provides key benefits by helping to ensure that:

- The architecture works early in the process.
- Stakeholders become educated about how the product works.
- Product limitations are revealed during the development process.

Improved compliance with standards and mandates

Projects with processes and artifacts governed by standards and/or compliance mandates are well-suited to the iterative development model, because standards-driven deliverables can be specifically identified for completion at the beginning of each iteration. For example, if compliance with a given standard requires the ability to associate a requirement with a specific code segment and its related testing and documentation, then each of these deliverables can be enumerated as tasks in the iteration plan. If the work is not successfully completed then it most typically moves to the beginning of the next iteration to be finished. This rule is only broken if other work is reprioritized higher and pushes this unfinished work into the “backlog.”

As previously noted, agile teams establish “done” criteria for work that to be completed with each iteration. This may include things like completing all code reviews, automating all unit tests and achieving 100 percent pass rates, all document reviews and updates to be completed, etc. The “done” criteria can also include all elements required for compliance with mandated standards. Using this mechanism allows teams to not only validate that the standards processes are completed as the project progresses but, with the right tools, it can provide an audit trail for compliance activities.

2. Improved quality

Agile development practices deliver just enough functionality to meet stakeholders needs with high quality. There are several aspects to delivering high quality that can benefit software-intensive systems development in the same way that they benefit IT development.

Technical debt

Technical debt refers to any unfinished work that the product tracks. Examples of technical debt include defect backlogs that are carried from release to release, code refactoring that never gets scheduled, unwritten tests, and incomplete documentation. “No technical debt,” a well-known agile mantra, means that teams need to get into the habit of removing technical debt before it collects to the point that it slows product development. Agile methods recommend that technical debt be addressed immediately or eliminated altogether.

Code refactoring means restructuring code to improve maintainability and architecture without changing its behaviour. Code refactoring can be considered technical debt if it is never prioritized and addressed. Agile methods recommend that you add work items such as code refactoring to the backlog so they can be prioritized as work items equivalent to new feature functionality. In many cases, these items improve the product by making it easier to fix or maintain.

Minimal functionality, high quality

Consider delivering “just enough”: That is, deliver the minimal functionality with high quality. Minimal functionality does not imply poor functionality; it implies enough to get the job done. Software projects suffer frequently when teams try to meet all stakeholder needs and anticipate all problems that might be encountered. Known as “gold plating,” this behavior often produces code and hardware that does more than is needed and subsequently requires more work to maintain it, or even to provide the essential functionality of the product.

Stakeholders tend to believe they know what they want as they specify high-level requirements for an IT or software-intensive product. But most of the time, when they see the finished product it often does not quite solve the problem; they simply

did not imagine the problem accurately, or the problem has changed, or the technology was not quite as promised, or the product does not actually work the way the stakeholder intended—even though they thought they described their requirements clearly. Developing in short, time-boxed iterations and demonstrating to stakeholders early and often enables teams and stakeholders to agree when the product has met each of the needs.

3. Improved predictability through better risk management

When development projects miss their release target dates, there are often many justifiable reasons. The team did not understand how hard the new technology would be to use; the requirements were not clear; the customers changed their minds after development was nearly finished. However, businesses demand that products meet shipping deadlines so that related business plans can be met. There are several ways that agile methods can help IT projects achieve release predictability that can also be applied to software-intensive systems development projects.

Prioritizing risk

Agile practice prioritizes high-risk aspects of development enabling early risk reduction. The product backlog is generally prioritized in order of highest demand from stakeholders. However, critical items considered a risk to successful implementation due to technology or architecture challenges are also prioritized at the top of the list. Addressing the risky problems

early in the project can enable the team to change their plans more easily if the risks cannot be managed, or allow the team to have the time to find an alternate solution if the need arises.

High risk items do not have to be completed before the next risk item is tackled. For instance, your team may have to tackle several risk items, such as using a new technology, finding a solution to a performance problem, or meeting new security mandates across the architecture. Each such item must be at the top of the backlog, but it is not necessary to complete the work on each item before you move on to the next. Figure out what user stories (the agile term for a requirement or set of requirements) or work items are required to tackle the risky questions; and do just enough reconnaissance work to make that determination, then do the same for the next high risk item, and so forth. Do not complete the work on each risk item serially. For example, the team wonders whether or not the new technology can be built with the existing build system, integrated with the existing architecture, and finally passing data successfully in the most complex scenario. They tackle this question by producing just enough working code and hardware to demonstrate that the risk has been addressed, and they leave it there and move on to the next risk area. You must devise a working plan for all the risk areas, then move back to the item of highest stakeholder demand and complete that work.

Parallel risk assessment

For risk areas where there might be multiple solutions and the team cannot agree on the right path, consider multi-set development. This requires that multiple teams work in parallel on solving the same problem with different solutions. Most teams will not even consider this approach because they are convinced that the time and cost required to do the parallel evaluation are too great. In fact, teams that try this method often find that they not only discover the best solution for the problem fairly rapidly, they also are able to leverage good aspects from each approach, which ultimately saves project time. Parallel development of alternatives is likely to bring key decisions forward.

4. Improved productivity profile

Agile teams are more productive than teams using traditional methods throughout the development cycle. Traditional development often demonstrates a hockey-stick pattern of work, starting with a design cycle up front, moving into a prototyping phase, then a long development cycle, followed by an unpredictable cycle of integrating the pieces, and on to a final test phase. As the project progresses, teams have to work together more coherently and hope that all the pieces work together as expected. But this is rarely the case, so team interaction increases as integration problems are addressed, and ultimately the test phase gets squeezed, which requires a cross-team scramble to

complete the testing and fix the defects. By that time, they've missed the deadline by weeks or months, and the business may have missed a critical opportunity

By working together as a whole team on vertical slices of the product from the beginning of the product cycle, the traditional hockey stick productivity cycle is avoided. Agile teams tend to be very productive from the first iteration throughout the release and their pace has to be managed so that burnout does not occur. Agile teams that maintain working code with every iteration enable performance and system testing to start in the earliest iterations as well. Critical defects such as integration issues are discovered sooner, the overall product quality is higher, and the team works productively throughout the development cycle.

Consistent, staged delivery is better for the team and the business. An agile planning approach and a focus on delivering working increments of functionality can lead to improved productivity compared to “high-ceremony” processes. Emphasis on incrementally delivering working software products means you can quantify progress—making projects easier to manage and improving stakeholder confidence.

5. Ability to leverage existing tooling investments

Adopting agile techniques successfully requires strong tool support. Most teams have already invested heavily in good software tools, and they need to leverage that investment and reduce the amount of change as they begin to adopt agile methods. The most critical investment is a good planning and work management tool that provides a visible team backlog and will associate work with each item in the backlog. Ideally, the planning tool integrates with other development tooling, which enables teams to maintain traceability from the backlog to other artifacts including:

- The requirements that drive it.
- The architecture under development.
- The software that delivers the solution.
- The testing that validates the solution.

Rational tools work well together and with other products to help successfully deliver agile projects. IBM Rational Team Concert™ is the core workflow and collaboration component of the IBM Rational solution for systems and software engineering. It provides a flexible Open Services for Lifecycle Collaboration

(OSLC) interface that allows OSLC-enabled tools to integrate seamlessly. Rational Team Concert provides the team with proven, visible agile planning; its adoption for just that purpose has been viral within IBM Software Group. Rational Team Concert provides work item management, team planning associated with work, and project visibility, all critical elements in agile projects. Work items in Rational Team Concert are very flexible in their usage and can be associated with releases, teams, sub-work items and so forth.

In particular, most teams have heavily invested in configuration and change management software such as IBM Rational ClearCase, Rational ClearQuest, Rational Synergy, and Rational Change. These products can work well with agile processes and specifically with Rational Team Concert. Agile processes managed in a process tool must link to real assets that deliver the work so that teams can quickly validate progress on the work and later be able to audit what was completed. The work that

delivers product requirements must be easily discovered and tracked. From the planned work, it must also be easy to discover the associated code that delivers the required capability, the builds that deliver the code, and the tests that validate the builds.

Rational Team Concert software also interoperates closely with other components of the Rational solution for systems and software engineering to provide agile tooling support for other aspects of the software-intensive systems software development lifecycle, including requirements management, model-driven architecture and design and quality management.

Agile methods help organizations face an array of challenges

Organizations developing software-intensive systems face a perfect storm of rising competition, product complexity, and customer expectations—coupled with development resource and time constraints that are stretching development organizations to the limit. To meet the challenges of this new reality, new development approaches are required. Teams need to deliver

greater productivity and predictability, reduced cycle-time, and improved quality without adding implementation risk to the projects. Agile methods offer a proven approach that can help deliver such business benefits to software-intensive systems development, while mitigating delivery risk through consistent and staged delivery. Agile techniques can also help overcome process overheads in meeting regulatory and standards compliance.

The Rational solution for systems and software engineering supports organizations moving to agile methods with flexible tooling and processes that cover the full development lifecycle. By adopting agile planning and management solutions alongside existing tooling investments for change and configuration management, teams can transform their delivery processes while helping to minimize investment cost and implementation risk.

For more information

To learn more about IBM Rational agile solutions, please contact your IBM marketing representative or IBM Business Partner, or visit the following website: ibm.com/software/rational/agile

To learn more about IBM solutions for software intensive systems development visit: ibm.com/software/rational/workbench/systems



© IBM Corporation
Software Group
Route 100
Somers, NY 10589 U.S.A.

Produced in the United States of America
December 2011

IBM, the IBM logo, ibm.com, ClearCase, ClearQuest, and Rational are trademarks of International Business Machines Corporation in the United States, other countries or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at ibm.com/legal/copytrade.shtml

Other product, company or service names may be trademarks or service marks of others.



Please Recycle
