Deloitte.

Don’t fear change, embrace it
Advancing the case for agile methods in systems integration

April 2010

Contents

Executive summary

Driven by a competitive and ever changing business landscape, our clients are looking for ways to meet their changing business needs through rapid technology implementations. They want to implement systems and system changes quickly and efficiently in order to realize business benefits early. The need for speed and agility has led them to consider agile software development methods as alternatives to traditional approaches. Agile software development methodologies consist of several methods that focus on reducing overhead and time to delivery. The various methods have different flavors of implementation approaches, but share common characteristics, including:

- Delivering tested software in short, time-boxed iterations
- Accepting and sometimes encouraging requirement changes between iterations in order to meet changing business needs
- Mandating extensive client involvement during each iteration to understand ambiguous requirements and increase development team productivity

However, many organizations struggle with perceived agile downfalls, such as limited planning, weaker controls, and less documentation. This is a common misconception because agile software development can be integrated within existing organizational boundaries of standardized processes. In fact, most agile methods do not prescribe a methodology for project management or quality control. Instead, they focus on simplifying the software development approach, embracing changing business needs, and producing working software as quickly as possible.

This paper will introduce the drivers and background for agile development methods, summarize agile impact on SDLC phases, outline project characteristics and challenges for agile adoption, and address common myths about agile software development.

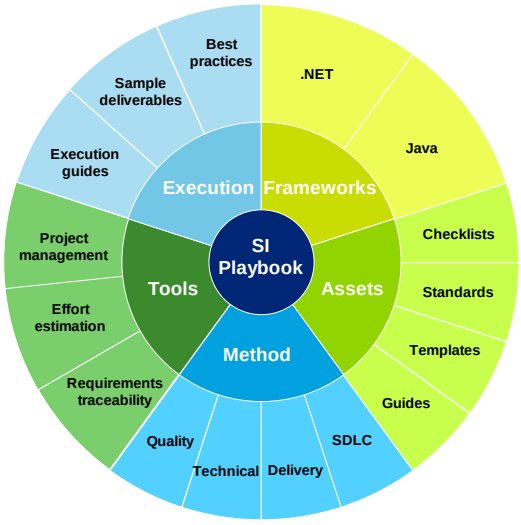
In addition, some agile methods have been successfully used in environments that are assessed at CMMi Level 3 or higher, and that use management principles based on the Project Management Body of Knowledge (PMBOK).

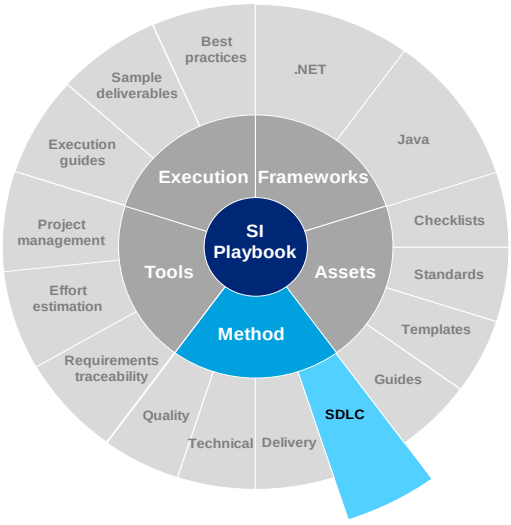
While agile development has the ability to simplify and streamline the software development process, it may not be suitable for every project. Some projects, such as mission critical and large enterprise projects, will continue to require rigorous planning and formal processes. Others will require flexibility and agility in order to meet changing requirements. Project characteristics and organizational limitations are critical considerations when evaluating Software Development Life Cycle (SDLC) methodologies. Agile methods, while scalable, are better suited for small-to medium-sized teams, limited scope to one application or one functional area, user-centric solutions, and short timeframes. There are trade-offs between agile methods and traditional ones that must be considered for every project. The project sponsor must balance the need for agility, speed to market, flexibility, and openness to rework with the need for additional rigor and structure, approved scope, detailed documentation of all aspects of the SDLC, and long-term plans.

A key point to remember is that regardless of the development methodology used, there are many agile methods and tools that are beneficial to any software development project. Examples include Test Driven Development (TDD), Continuous Integration, automated unit testing, code refactoring, and many others. Such methods are proven to increase the quality of the implemented solution and organizations should look to adopt them on as many projects as possible.

Introduction

Deloitte's systems integration service line is focused on the delivery of custom/integrated solutions for our clients across several industry sectors, ranging from public sector, to health care, financial services, consumer business, technology, media, and telecommunications. We provide exceptional software development and project management delivery capabilities using our world class Systems Development Playbook





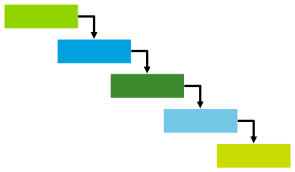
The Method component of the Playbook, which incorporates concepts and tenets from well-established models/approaches, such as Capability Maturity Model Integration (CMMI®) and the Project Management Body of Knowledge (PMBOK®), has evolved throughout the years from the traditional waterfall development process to include more iterative/incremental methods.

In recent years, however, industry has seen new and more innovative approaches to software development that have been shown not only to reduce financial investment and time, but also achieve significantly higher levels of end-user satisfaction.

Although our Playbook can be adapted to these modern and 'agile' methods of software development, we need to formally incorporate these different perspectives into our methodology, and educate our practitioners on their application and benefits in order to stay a current with shifting industry trends and continue to excel in our custom development capability.

As with most standardized processes, software development methods have matured from relatively unstructured practices to more formal and well-defined processes. Among the first structured development methodologies was the waterfall' approach, which garnered support and notoriety in the software development industry.

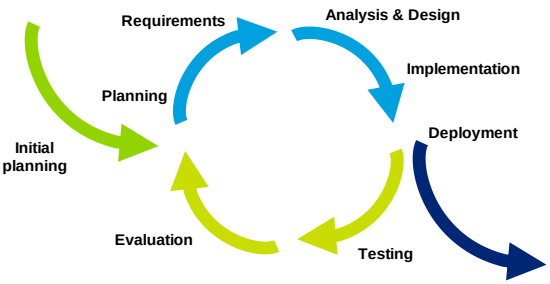
Coined in the 1970s for its sequential flow of progress from top life cycle phases to bottom in waterfall-like fashion, the pure waterfall development methodology is affixed on the central idea that significant time should be spent on each life cycle phase to ensure 100% completion and correctness before the next phase should begin.



While several tenets from this model (e.g., defined phases, appropriate documentation) have survived and have been adopted by other delivery methods, its inflexible structure and rigidly controlled processes have often resulted in increased costs and time investments due to the near impossibility of perfecting any phase of the life cycle without incorporating iterative changes. See 2.1 for details of waterfall model limitations.

As a result, the industry took a turn towards embracing change. Development teams saw the need to incorporate iterations and facilitate their stakeholders' changes in requirements and design through explicit feedback.

Iterative development methods are cyclic in nature and provide stakeholders and customers exposure to incremental and deliverable versions of the system. In turn, they are able to provide better validation of and/or updates to product requirements. Ultimately, this resulted in effective product redesign that better achieved project goals and customer expectations.



Iterative software development paved the way for more modern approaches by introducing the acceptance (and even encouragement) of change, thereby reducing the risks of the traditional waterfall method, which effectively delayed end-user feedback until the very end of the process. But as size, scope, and complexity of software development projects evolved, iterative concepts alone could not offset increasing costs of resources and time required for software development projects.

Traditional ideals of the past including excessive documentation, micromanagement, rigid processes and tools; and affixing the project course to predefined plans (created well before any real idea of the design of a solution) still existed. These practices limited human collaboration, and timely response to changes. Ultimately, these limitations affected project timelines, increased development costs, and decreased the success rates of projects.

In attempts to minimize these risks, industry developers made changes to lighten the load of the heavily regulated processes of the traditional development methods. Categorized as 'lightweight' methodologies, these concepts were varied, nonstandard, and were not unified until the publication of the Agile Manifesto in 2001. Here a group of the leading developers and lightweight methodology proponents met, discussed, and agreed on common concepts that composed the manifesto that is still widely referenced and drawn upon today.

Based on manifesto's concepts and detailed principles, numerous agile methods were developed and garnered support in the software development industry. See Section 3 for descriptions of agile methods and advantages.

Motivation for change

The problem statement

As mentioned previously, the waterfall's heavily regimented, and often micromanaged, processes along with its sequential structure inhibit the channels for change that are needed to achieve desirable end results.

The most significant issue with the waterfall method is its sequential and noniterative structure. Customers are often unaware of and unable to define sets of requirements until later stages of development in which visibility to deliverable versions of the product are available. As a result, the team has what seems like one shot to get each phase correct. If any changes occur, the whole process will likely need to be redone. For example, if the client identifies changes to initial requirements in a later phase, the design must be modified to accommodate the new requirements. This creates a domino effect that results in recoding based on updated design specifications, retesting based on new code, etc. This issue is magnified in complex development projects, where requirements, design, and development efforts require significant resources and time.

Another typical issue exists in the collaboration between cross-functional teams and their customers. Waterfall methods do not support the frequent exchanging of information, review, and feedback processes. In addition, the teams are typically in disparate locations and do not interact until certain hand-off points from one phase to the next. As the size of projects and solutions grow, so do the teams and without

careful consideration and mitigation, the issues above are exacerbated.

Business customers or 'end-users' are negotiated with through the medium of written and formalized contracts, and the attendant 'sign-off' process is usually heavily enforced. These characteristics along with the sequential structure of the waterfall method make it difficult to consistently produce a high-quality solution with accompanying deliverables that adapts to feedback, changes, or other unforeseen issues. For example, a designer may design system functionality a certain way, but fail to realize the difficulty of the programming involved to provide that functionality. The problem appears after the design has been finalized and agreed to by the customer, as the developers are unable to overcome a programming roadblock uncovered during the build phase. Similarly, the end-user may not have understood the exact application of technology and the specific requirements that were needed until the solution was presented. These issues could have been avoided if better collaboration, iterative reviews, and feedback on the solution were made part of the process.

That being said, there are certain situations where the waterfall model is appropriate and proven to be successful. These projects typically share the characteristics of being mission critical and are executed in highly regulated environments; thus requiring a comprehensive set of requirements upfront and a stable project environment. However, as history and our experiences have proven, software development projects do not often fall into these situations. Instead, we continue to see more cases where unanticipated issues occur and the project team is forced to take a change in direction to resolve the problem.

These projects, like most, are sensitive to overruns in budget and time, and despite the frequent changes, have demanding clients that request frequent and continuous visibility on progress. In these situations, a more 'agile' approach to software development is needed.

How should things change?

We have recognized that the waterfall method is not suitable for projects that are susceptible to changes and requires strong customer collaboration and readily available communication channels between functional teams. In order to adapt to these situations and the evolving needs of clients and customers, software development projects should consider the following changes:

- Welcome changing requirements from customers
- Incorporate iterative elements into the life cycle in order to proactively anticipate changes to requirements and design of initial prototype
- Implement a disciplined project management process that encourages frequent inspection and adaptation while reducing rigorous documentation and process controls
- Provide frequent and tested and working versions of the product to facilitate feedback from customers
- Promote teamwork and establish a collaborative working environment to facilitate continuous communication

These logical recommendations are aligned with the overarching guidelines described in the Agile Manifesto and are apparent in all agile methods.

Why agile and why now?

Since the publication of the Agile Manifesto in 2001, agile software development has gained significant adoption and is no longer considered an alternative approach to software development. Indeed, agile software development has been adopted by a large number of organizations, including some of our clients. For some, it is now the default way of building and integrating software solutions.

Since the millennium, we have seen more and more custom development projects being delivered in an agile manner and we are increasingly being asked by clients to respond to RFPs or assist in troubled projects by using more agile approaches such as spiral and scrum.

One recent example is the U.S. Transportation Security Administration (TSA). Traditional timeframes for application development efforts are regarded as too lengthy for TSA's IT services to effectively develop departmental solutions, often leaving departments with limited options for addressing emerging needs. The TSA engaged Deloitte in response to these needs. To resolve the client's issue, Deloitte utilized an agile method in Rapid Application Development (RAD) to develop solutions for departmental customers. The value was apparent:

- Decreased overall time to deployment and life cycle costs for application development efforts
- Worked with TSA IT leadership to modify application deployment process to provide additional flexibility around application development timeframes
- Supported development activities for 30+ applications
- Through highly visible wins, the team has provided the IT organization with a reputation for responsiveness to its customers

In addition, clients are also asking us to help modify their own generally accepted, tried-and-tested, waterfall-based SDLC methodologies to incorporate more agile approaches. For example, Hitachi Data Systems (HDS) is a storage solutions provider that has traditionally used a standard waterfall SDLC for most of their IT projects — whether they are small fixes, custom-developed solutions, or larger ERP projects. HDS is requesting our assistance in developing new delivery methods and a playbook that will guide IT personnel in their decision making as to when to employ the most appropriate approach. The effort also includes developing a transition plan to successfully implement the recommended SDLCs into the client's IT organization.

Agile methods

The agile evolution

In February of 2001, 17 prominent software development figures (Person A, Person B, Person C, Person D, Person E, etc. — just to name a few) met in The Lodge at Snowbird ski resort in the Wasatch mountains of Utah to share their ideas on software development methods known as 'lightweight' methods at the time. The result of the meeting was the Agile Manifesto

Table 1 — The Agile manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value:

Individuals and interactions	Over	Processes and tools
Working software	Over	Comprehensive documentation
Customer collaboration	Over	Contract negotiations
Responding to change	Over	Following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Source: web address cleansed.

Following the publication of the Manifesto, the Agile Alliance was created to promote and evolve agile software development methods. The manifesto was derived from the ideas of many iterative development methods including Extreme Programming (XP), Scrum, DSDM, Crystal, Adaptive Software Development, and several others.

It is rather difficult to disagree with the Manifesto once it is fully understood. In fact, most people in the software development space nod their heads when they hear the above statements. These principals resonate in high-quality organizations that have great talent and pride themselves on collaboration, client service, and adaptability to change. The fact remains that without talent, no software development process will ever lead to success.

Incorporating agile methodologies into our playbook puts us in good company along industry leaders (Google, IBM, Lockheed Martin, and many others) and positions us to provide our clients with the right type of SDLC transformation based on their culture and business needs.

Agile characteristics and techniques

There are more than 11 well-known agile software development methods, however, they all share common characteristics, including:

Table 2 — Agile characteristics

Agile characteristic	Description	Benefits
Time-boxed iterations	• Scope (in the form of user stories) is broken up in small chunks based on priority and/or complexity • Iterations are time boxed in short timeframes (generally one to six weeks) • Each iteration includes the entire SDLC life cycle • The result of each iteration is a working solution with minimal bugs • Multiple iterations may be required before releasing a system to end users • Scope is generally fixed during iterations, but can change in between iteration based on new business needs	• Improved quality due code integration and test within each iteration • Early return on investment as working code is produced early on in the process
Client collaboration	• Client lead (product owner) is an active team member who is engaged in iteration planning and review. • Product owner is always aware of progress being made and the direction of system development • Product owner is available to answer the team's question	• Quick resolution of issues • Adaptive evolution of requirements from high-level user stories to detailed requirements through continuous client discussion • Trust-based relationship with the client
Self-organizing teams	• Agile teams are typical assembly of cross-functional team members capable of carrying out all aspects of the SDLC • Team members are empowered to make decisions, estimate effort, decide roles and responsibilities, and are protected from outside influences • All team members are given the opportunity to interact with the client • Colocated teams with emphasis on face–to-face communication. If team members are not colocated, then the use of video conferencing and conference calls to keep the team connected	• Improved productivity • Increased motivation • Clear responsibility and accountability • Staff development and growth • Improved team chemistry
Adaptable to change	• Agile methods embrace change as inevitable. Business needs are changing rapidly and agile methods do not assume that requirements can be frozen • Thus, agile methods work closely with the client to adapt the solution to their changing requirements • Scope and requirements are reviewed between iterations, modified, and prioritized for the next iteration • In addition to adaptability to requirements, the design of the system is also adaptable to changing needs. Team members are encouraged to refactor code and improve their quality regularly. The team is encouraged to adapt any aspect of the system as new information becomes available.	• Increased client satisfaction as the system is adapted to their needs • Improved system design that is simple and easy to understand (due to frequent refactoring of code)
Product focused/results oriented	• Working software is the main objective of the agile team • Every activity must be beneficial to the main objective • Client needs related to software documentation are treated as requirements that are prioritized along with the rest	• Working code (early progress demonstrated) • Less distractions

In addition, agile methods emphasize several development techniques to improve code quality and simplify system design.

These techniques, if implemented, provide the necessary protection against a “code and fix” culture

Table 3 — Agile techniques

Agile technique	Description	Benefits
Continuous integration	• Integration of code frequently (whenever a developer commits new code to the existing code base) using automated build scripts • Typically, includes the execution of unit test suites and reporting the results • Managed and monitored by automated tools such as CruiseControl, which can integrate, compile and deploy code, execute unit tests, apply standard compliance checks, and provide code quality statistics	• Published build results • Frequent detection and correction of bugs • Early warning of code integration issues • Automated unit tests with metrics around code coverage of test scenarios
Refactoring	• Modification of source code without impacting external functionality • The objective is to improve code readability and reduce complexity	• A code base that is easier to maintain • Extensible solution
Pair programming	• Two programmers work side by side on one machine • One developer codes (the driver), while the other reviews the code and provides feedback (the navigator) • Roles are switched frequently	• Improved code quality • Less bugs • Increased focus and productivity (less distractions)
Test-driven development	• The test is written first — failed test • Code is written to pass the test • Code is refactored to improve its quality • Tests serve as a safety net that allows developers to make changes without fail	• A comprehensive test suite • Code that is easier to maintain • Introduction of new bugs is detected early on in the process • Instills a testing culture within the team, which in turn improves code quality and customer satisfaction
Automated unit testing (xUnit)	• A framework that allows developer to write test suites that can automatically invoke code objects and test their functionality • Frameworks are capable of tracking code coverage (indicating how much of the code is invoked during testing)	• Automation of unit test execution • Enables the team to test code objects early without integration with a front/back end

A range of agile methods

There are many flavors of agile methods that have evolved overtime to meet different needs in various industries. The below table features some of the better known methodologies and contrasts the methods based on certain project characteristics. This list is not comprehensive and these methodologies are not mutually exclusive as they can be combined at times based on organizational needs.

Table 4 — Agile methodologies

Agile method	Description	Unique concepts/ Differentiators	Choose methodology when				
			Project scope	Team size	Resource quality	Risk profile	Culture and business involvement
Extreme programming (XP)	A method that is focused primarily on development and testing. It was formalized by Person B in the late 90s. The methodology is simple and focused on empowering the development team to increase productivity and produce software in short releases For more information, reference: http://www.extremeprogramming.org	• One-to two-week iterations • TDD, peer programming, code refactoring • Daily involvement of business people with the development team • Embraces changes • Introduces an architectural spike to outline the design of the system and explore unknown areas upfront • User-centric approach through user stories	**Small** User centric. Limited complexity	**Small** XP teams (five to seven), but can be scaled to multiple teams	**High**	**Low**	**Very collaborative** Daily business involvement required
Scrum	Formalized by Person E and Jeff Sutherland in 2001. Scrum adds a simple Skelton of “management practices” to software development techniques shared with XP. Its main focus is on collaboration, cross-functional and self-organizing teams, and frequent product reviews with customers to solicit feedback For more information, reference: *Agile Software Development with Scrum by Person E and Mike Beedle* *Agile Project Management with Scrum by Person E*	• Four-week sprints with daily Scrum meetings • Simple and effective release and sprint planning • Role of the Scrum Master as the facilitator of issue resolution and the protector of the team from outside distractions • Sprint review and Sprint retrospective meetings at the end of each sprint • Provides suggestions for producing a product backlog, sprint backlog, and a Burn down chart • Introduces a Scrum Master certification	**Small–Medium**	**Small** Scrum teams (<10) **Scalable** up to 100 total members	**High** Requires cross-functional resources	**Low–medium**	**Collaborative culture** Emphasizes **colocation** and verbal communication. Product owner involved regularly

Agile method	Description	Unique concepts/ Differentiators	Choose methodology when				
			Project scope	Team size	Resource quality	Risk profile	Culture and business involvement
Spiral	Introduced by Jim Boehm in 1988 as a methodology that is focused on iterative prototyping and development of high-risk solutions. Spiral consists of "spirals" of Customer communication, Planning, Risk analysis, Engineering, Construction and release, Customer evaluation For more information, reference http://www.softdevteam.com/Spiral-life cycle.asp	• Include a thorough risk analysis phase to assess both technical and management risks • Produces software early for customer evaluation • The risk analysis phase is critical to the success of the project • More formal planning phase than other iterative methods with an upfront planning phase to identify resources and set timelines	**Complex** and mission critical	**Large**	**Very high** Significant risk analysis experience	**High**	**Low-risk tolerance**
Feature-driven development	Introduced by Jeff DeLuca in 1999 as a model-driven iterative approach to development that is focused on producing working features in short cycle (two weeks). Its activities consist of developing overall model, building feature list, planning by feature, designing by feature, and building by feature For more information, reference http://www.featuredrivendevelopment.com	• Features that take longer than two weeks to develop are decomposed • Iterations are not longer than two weeks • Small teams focused on interrelated features • Provides more structured roles than XP and Scrum including testers, tech writers, release managers, etc.	**Medium** complexity	**Medium**-sized teams focused on features No defined limit on scalability	**Medium** (defined roles reduce the need for superstars)	**Low–medium**	**Transparent** culture that is open to visibility and allows team to make some mistakes and improve. Medium level involvement of business stakeholders is required (not daily)
Dynamic systems development method (DSDM)	Based on RAD, DSDM is focused on projects with tight schedules and budgets and includes three phases: preproject (initiation), project life cycle, and postproject. The project life cycle consists of five stages applied iteratively: Study, functional model iteration, design and build iteration, and implementation (including UAT and training) For more information, reference: http://en.wikipedia.org/wiki/Dynamic_Systems_Development_Method DSDM Consortium — www.dsdm.org	• Nine principles that must be followed • Emphasizes active user involvement • Focuses on frequent delivery of both software and documentation • Encourages baselining of requirements at a high level • Fosters a collaborative and cooperative approach between business and technology • Methodology is formally tracked and versioned (current version is 4.2) • Timeboxes less than six weeks	**Low–Medium**	**Smaller teams** (Five core members) A project is typically two teams. There are reports of scalability of up to 150 team members	**High**	**Low**	**Executive buy-in** is key and significant business involvement is expected

Other well-known agile software development methods include Agile Modeling, Adaptive Software Development, Lean Software Development, Agile Unified Process (AUP), Essential Unified Process (EssUP), and Open Unitified Process (OpenUP)

Advantages of Agile

While Agile methodologies may not be ideal for every project, There are many advantages to implementing agile practices on some projects (we will explore the characteristics of projects suitable for Agile adoption in Section 5):

Table 5 — Advantages of Agile

Category	Advantages
Client satisfaction	Increased client involvement and frequent reviews of actual progress (as opposed to status reports) increases the client's level of comfort with the end product. In addition, the flexibility the client had in reprioritizing scope in between iterations leads to increased satisfaction
Quality improvements	When the development team applies TDD, Continuous Integration, Refactoring, and Pair Programming, the quality of the system design and the maintainability of the code base improve. In addition, automated processes allow the team to gather performance metrics providing valuable information (e.g., unit test code coverage, coding standard violation, etc.)
Reduced risk of failure	Iterations produce tangible and working software that is tested by the team and reviewed by project stakeholders. Frequent testing and verification by the business owner reduce the risk of catastrophic failure.
Faster ROI	Given that working software is the main goal of software development projects, agile development produces working code much earlier than traditional methods, thus providing the client with the ability to release features earlier than they would able to if they used traditional methods. In addition, if the project is canceled for any reason (e.g., lost funding), agile projects leave behind more assets in the form of working code, which can be picked up again and reused if the project is resumed. Documents on the other hand might not be as helpful to restarting the project.
Team morale	Agile teams require well-rounded developers who can understand business needs, interact with the customer, design components, and develop code. The well-rounded nature of team members, coupled with frequent access to the client to clarify questions and resolve issues, allows the agile team to be very productive during normal working hours. In addition, practices such as pair programming can be a great development opportunity for less-experienced developer to grow and take on more responsibility.

Impact on the traditional SDLC

A traditional SDLC approach will continue to be used in environments that require regimented and structured approaches to software development. Agile on the other hand lends itself to a more adaptive and creative approach. In a vast majority of situations, A hybrid approach combining agile principles with necessary structure may yield the best results. The level of agile adoption varies from project to project based on the circumstances.

This section provides one perspective on how agile software development methods impact SDLC phases for large projects

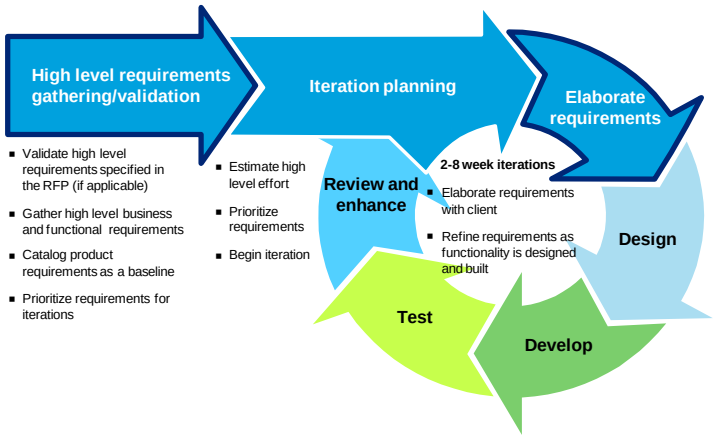
Requirements gathering

The approach to requirements' gathering depends on the type of the project and any existing contractual restrictions. Generally, agile methods call for requirements evolution during iterations, however, there will be situations where requirement gathering is necessary early on. Examples include:

- RFPs may include high level requirements, thus the requirements need to be used as a basis for estimation and iteration planning
- An overall budget estimate maybe required, thus requiring the team to gather high-level requirements upfront. The details of the requirements depends on the degree of confidence associated with the estimate

Agile methods allow for high-level requirements gathering, but suggest that we should stay away from details (prescriptive) requirements because those tend to evolve throughout the project.

Agile methods emphasize focus on "user stories," instead of detailed system requirements. The high-level requirements are helpful in defining the scope of the project and create tentative iteration plans.



The project team works with the client/product owner to elaborate prioritized requirements during iterations. The requirements evolve as the team proceeds to design and develop the functionality, and the client gets to validate whether the completeness of accuracy of the requirements during regular product demonstrations.

Given the flexibility in agile methods toward change in requirements, one of the most important process adjustments that project teams must make involves change management. Requirement changes will happen often and can be categorized into two types:

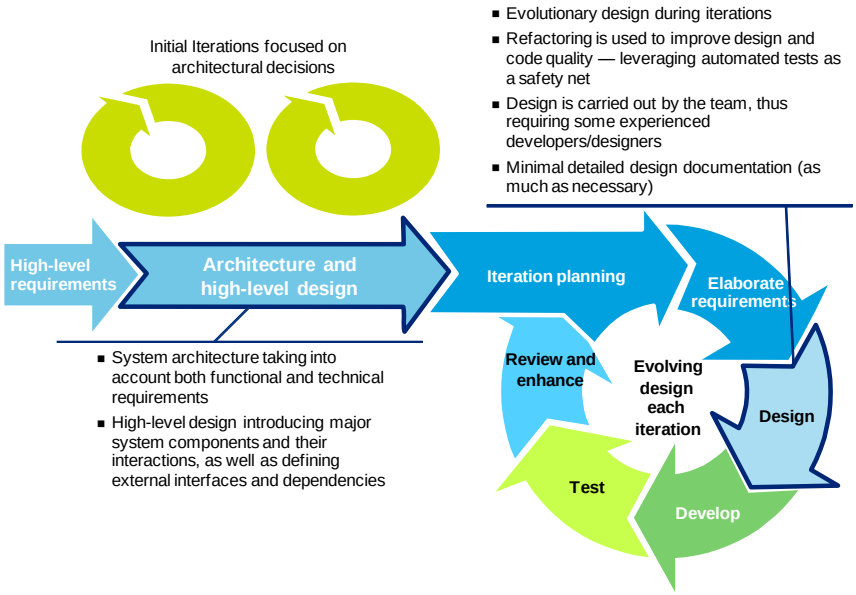
- Elaboration of high-level user requirements
  - Occur during iterations
  - Refined as functionality is reviewed by the client
  - Do not constitute a change in scope as the high-level requirements were scoped for the iteration
  - If new user requirements are discovered as a result of elaborating user requirements, they are added to the requirement catalogues and prioritized between iterations. They are not added to the scope of the ongoing iteration
- Addition of new user requirements
  - Can be added between iterations
  - Are prioritized with the client and stakeholders and assigned to a particular iterations
  - May result in delays in delivering other features and increased overall cost of the product.
  - The formality of how these changes are managed depends on the contractual agreement with the client. Fixed price contracts will naturally require a more formal approval structure to ensure that additional funding is approved, while Time and Material contracts may have more flexible approach that includes communicating the impact to stakeholders, reprioritizing requirements for iterations, and identifying impacts on timelines, and gaining approval from the project owner in short timeframes

Design

Traditionalist sometimes label agile methods as code and fix or cowboy coding methods, but that could not be further from the truth. Agile methods evolved design practices because it is evident that detailed design of a system evolves during development as developers uncover challenges that could not be predicted using traditional up-front detailed design. Agile methods focus on evolving the design as the code is developed and employ practices that help the team refactor their code with confidence to improve the quality of their code.

This is not to say, as we explore later in this document, that no design should be done or that architectural decision should be left for every team to make. In fact, many agile advocates agree that it is important for the architecture of the foundations of system design to be laid out upfront, particularly on large implementations. It is a common practice to focus on system architecture during the first few iterations before widespread development of features begin.

On the other hand, detailed design of the system features is carried out during iterations. Team members may opt to use a modeling tool or simply draw out designs on a whiteboard or a flip chart before writing code. The design is evolved over time by using practices such as code refactoring to increase the readability and maintainability of code.



Development

Development is one area where several agile tools and methods have widely infiltrated waterfall and RUP-based projects. Development managers have successfully adopted Continuous Integration, automated unit testing, and in some cases TDD. The development phase can benefit from some key agile methods even if the rest of the SDLC does not follow agile methods. Although, investing a vast amount of time into detailed technical design specifications before using iterative development is not an efficient use of resources.

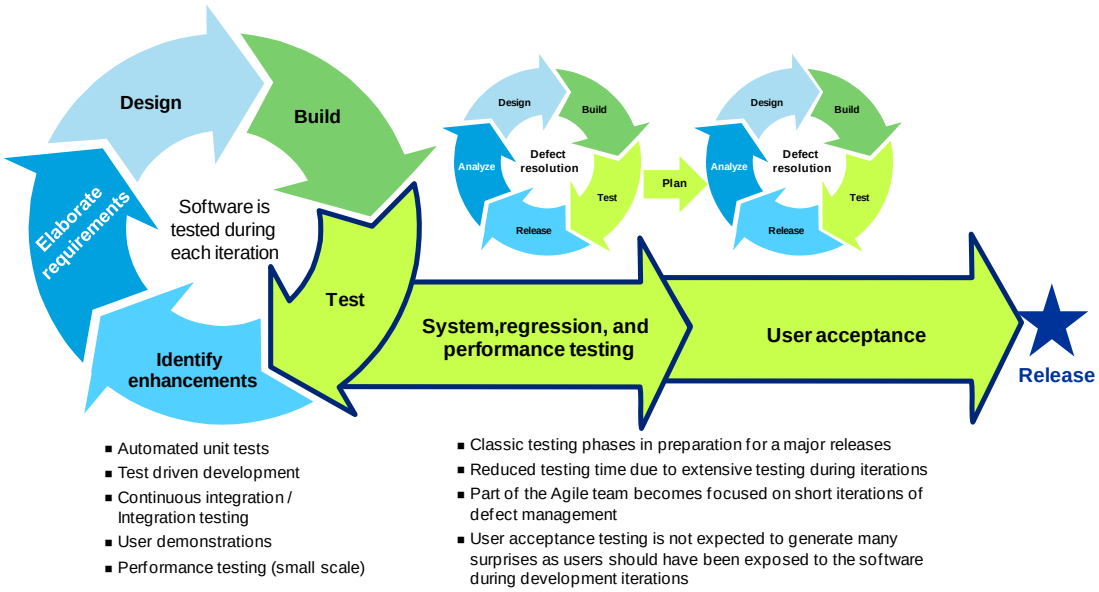
Some of the key practices required to run an agile development shop that is effective at producing working code in small iterations include:

- **Continuous integration** — Team members must integrate their code into the code repository frequently (at least daily). The code is integrated using an automated build process to identify integration issues early on. When all integration issues are fixed, the build is deployed into a test environment for functionality testing. In general, continuous integration reduces the risks and long timeframes associated with code integration when carried out at the end of a long development cycle.
- **Code refactoring** — Team members will continue to improve the code base, refactor common code, and enhance system functionality in a continuous drive toward delivering a quality product. Code refactoring is key to improving the design of the system and thus its maintainability over time
- **Evolutionary design** — As mentioned earlier, teams need to adapt the design of the system as they elaborate on requirements and develop additional functionality. The evolution of design becomes part of core development operations
- **Unit and integration testing** become part of development because it is the only way to verify that tangible and working software is delivered in every iteration

- **Roles and team composition** — Agile team members must possess cross-functional and well-rounded skills as they are expected to participate in requirements and design discussions, interact with the customer on regular basis, understand and test business functionality, and contribute to iteration planning efforts by providing realistic time estimates

Testing

Agile methods call for integrating testing into each iteration, but generally do not specify how prerelease performance and user acceptance testing is conducted. On small projects, mini UATs can take place during iterations. However, large projects will require a more formal testing approach prior to releasing software to a large population of end users.



Testing activities can be grouped into iteration testing and formal release testing.

Iteration testing

- **Automated unit testing (xUnit)** — Automated unit testing suites are integrated as part of the build process and used as a safety net to provide developers with the necessary confidence to change code without fear of introducing bugs. Combining automated unit testing and TDD and continuous integration yields the best results. After each build (at least daily), test results are published and developers must fix any defects detected. Source code is not migrated out of the Dev environment unless all unit tests pass.
- **Iteration integration testing** — While the code base is integrated daily, integration testing is typically conducted at the end of each iteration (prior to conducting client demonstrations). Similar to typical test approaches, integration testing is carried out in a separate integration environment. Most integration scenarios are often automated using regression test tools, while some are carried out manually. Any bugs discovered during integration testing are fixed during the same iteration. Alternatively, enhancements are prioritized for future iterations.
- **Iteration performance and load testing** — some form of load and performance testing should be conducted during iterations to help ensure that the team did not implement features that greatly hinder the performance of the system. However, this small scale performance test should not be considered a substitute to formal load/performance testing

If the client determines that the product of a particular iteration is ready for release, more formal tests are required to certify the functionality and release it to end users. The formal testing process is similar to traditional SDLC approaches.

Formal release testing

1. **System and regression testing —** For large systems, it is critical to run a full suite of system and regression testing when a product is ready for testing/release. The duration for this type of activity should be shorter than it is in traditional SDLC methods due to the improved quality of the product and rigorous iteration testing
2. **Load and performance testing** — All systems need to undergo load and performance testing in an environment identical to Production if possible. This process remains similar to traditional methods
3. **User acceptance testing** — A form of user testing is conducted at the end of iterations to demonstrate product features and obtain feedback from business owners. However, a formal UAT process will probably be required by most organizations before code can move into production. Agile will help make the team avoid surprises in UAT and make the outcome more predictable. It is also possible to treat UAT as a prerelease iteration where end users test the same and the team works on resolving identified defects in the development environment.

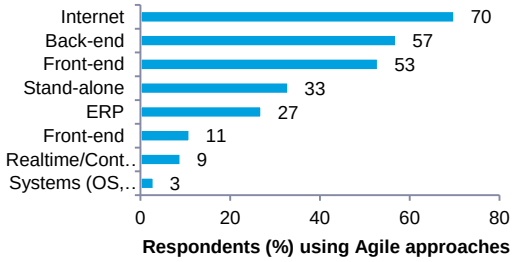
Identified defects are prioritized for short defect resolution development iterations. The agile team focuses on resolving all defects and releasing them for continued testing. It is important that enhancements are not squeezed in at the last second. All new enhancements need to be prioritized and added to iteration plans based on the change management process.

Agile adoption

Candidate projects

The degree of agile adoptions on projects varies. Some projects benefit from full agile adoption, others may need a hybrid mix. Generally, early adoption trends show that agile is more frequently used for applications with high levels of user interactions, such as Web applications, business applications to manage operations, mobile applications, etc. The below diagram shows how early agile adopters have leveraged agile for a wide variety of project types

Figure 1 — Project types that used Agile development — early adopters



Source: Journal of Information Technology Management — *Agile Software Development: A Survey of Early Adopters*. Leo R. Vijayasarathy, Dan Turk, Colorado State University

When assessing whether agile is the right answer for a particular project, we need to take into consideration several characteristics including risk profile, requirements volatility, timelines, scope and team size, and level of end-user interaction. Generally, agile processes are best suited for projects with the following:

Table 6 — Project characteristics best suited for agile projects

Characteristic	Description
Risk profile	**Low–medium risk profile —** agile is best suited for dealing with non-mission-critical projects, simply because such projects require rigorous documentation and specifications upfront, therefore reducing the benefits of an agile approach. On the other hand, business applications, eCommerce Web sites, self-service portals, and similar Web-based applications are prime opportunities for agile adoption.
Requirements volatility	**High requirements volatility —** high requirements volatility is a key indicator of projects that can benefit from agile methods. The volatility can be due to client uncertainty or simply the nature of the business. The adaptability of agile methods can help mitigate the risk of volatile requirement and even help the client figure out what they need through regular interactions with the team and the system they are building.
Aggressive timelines	**Aggressive timelines (speedy delivery)** — when rapid delivery is at a premium, working software is much more important than elaborate documentation. Agile adoption can demonstrate value quickly by producing code and helping the client focus on what is critical versus what is nice to have. The ultra collaborative nature of agile teams also increases team productivity and subsequently, the likelihood of meeting the timelines via an initial release containing key features.

Characteristic	Description
Team size	**Smaller teams —** agile methods are typically better suited to smaller teams, generally less than a 100 people. Larger projects require rigorous processes to ensure communication across large teams. In fact, many agile methods call for teams of a 100 or less to be broken down into small agile teams of 5-10 people. On projects of 50-100 people, a hybrid agile approach maybe best as high-level requirements, interface definitions, overall system architecture, and high-level planning will be much more essential to the overall success of the project.
Level of end-user interaction	**Significant end-user interaction — An** agile is more desireable when the target system has a significant UI component. UI requirements evolve as users get to see and test the product. Frequent product demonstrations during development provide early opportunities to incorporate feedback and meet end user needs.

Key considerations

As is the case with adopting any SDLC methodologies, there are several key considerations that must be taken into account when adopting agile. Paying close attention to these considerations and determining ways of handling them can make the difference between success and failure.

Table 7 — Key considerations

Scope	Description
Organizational culture	A hierarchal and rigid culture with limited transparency may pose serious challenge to agile adoption. It is critical that the necessary steps are taken to identify the right “pockets” for initial agile adoption. Training may also be required to ease people into a more transparent and flat structure on agile projects.
Management buy-in and project manager adoption	Senior management may not be aware of agile software development methods and project managers are likely to resist what appears to be “loss of power.” However, senior management opinion can be swayed by demonstrating progress and project manager need to realize that their most important role is to deliver projects. Agile, on some projects, may increase their chances of success. Project managers need to become part of the team and focus on resolving issues to help the team progress toward a common goal of delivering working software.
Increased business involvement	Agile methods will certainly require increased business involvement and in some cases dedicated business resource. Frequent business feedback is essential to the success of agile projects. In some cases, it is helpful to set up guidelines of how frequently the business will meet with the team, review progress, answer outstanding questions, and provide feedback. This interactions need to happen at the beginning and end of each iteration, as well as frequent touch points during iterations (preferably daily).
Regulatory requirements and constraints (SOX, Audit, 21 CFR, etc.)	Regulatory requirements typically impact agile development approaches because they included a greater level of rigor in documenting system functionality, testing, and requirements traceability. There is no reason why these requirements cannot be met with agile development methods. The regulatory needs need to become part of the work required during each iteration requiring certain types of documentation that can be audited. The regulatory requirements need to be taken into account and factored into iteration planning. They are activities like all others and should be considered critical to the success of the project and treated as “system features” by the development team.
Team composition and resource quality	The agile team requires well-rounded cross-functional resources. The same resources are expected to interact with the customer, understand the business, design features, and develop and test code. Often times, agile teams are much more successful when they are composed of experienced developers. Junior developers should be given the opportunity to grow by pair programming with more senior team members. Depending on the complexity of the project, some team members can be dedicated to certain activities, such as system architecture and documentation/testing.

Major challenges

Adopting any new methodology or approach includes facing new and unfamiliar challenges. Agile methods, despite significant benefits, still present several challenges to our current way of doing business. Agile methods impact fundamental areas, such as contracting and cost estimation.

Table 8 — Major challenges

Challenge	Description
Contract types	Given the dynamic nature of agile software development projects, most contracts need to be Time and Material. The business owner gets to decide how much functionality will be development in each iteration, knows the team size, and can control the budget accordingly. Major deliverables are replaced with iteration progress reports and actual software demonstrations.
Cost estimation	Producing effort, duration, and cost estimates proves problematic with agile methods because scope is not typically frozen and thus estimates can change at the end of each iteration. However, if such estimates are required, it is best to gather high-level requirements upfront in order to produce cost estimates. As the requirements change for each iteration, estimates can be revised (up or down).

In addition to the above challenges, agile methods present organization transformation challenges to our clients, including willingness to adopt new methods, the ability to function is a "flat" world without hierarchies, and the commitment of staff to the success of projects. The nature of agile methods may require organizations to take a bottom-up approach to introducing agile instead of a forced top-down approach typically associated with large transformations. Buy-in from project managers, developers, and business ownership is the most important success factor.

Where does this leave Design?

Is Design dead?

With all the talk around different methodologies, one might reasonably ask the question: where does this leave design; is design dead?

Far from it! Sure, design needs to evolve — as does every other discipline of software engineering — but dead? Absolutely not!

Design is and will remain an integral part of every delivery method, beginning with the overall architectural design of a solution, and evolving into a more detailed set of designs that cover the various domains of the solution.

In fact the overall architecture design of a solution is arguably more important in an agile project than any other discipline, since more often than not, it will be carried out in the first true phase of the project and fundamentally articulates the boundaries of the solution, starting the wheels rolling as the project enters the next phase.

Carried out effectively, design begins with an initial outline of the solution that is filled in by further design activity carried out in successive phases or iterations. In an agile project, design receives — and improves itself through — regular feedback via the process of continuous integration within each phase or iteration. In a sense, this should be regarded as original design followed by successive improvements through redesign — almost Darwinian in nature.

However, there is a balance to be maintained: what design cannot be is an undisciplined, try-as-you-go process akin to the slap-dash, code-and-fix 'method.' Previously in this document, we introduced Test-Driven approach as a very smart way to go about architecting and building a solution according to a well-considered, unambiguous, and comprehensive set of desired capabilities. Indeed Test-Driven-Design provides a frequent and essential feedback on each piece of the solution, which leads to refactoring in order to address the failures encountered.

The only real drawback to this approach is that TDD tends towards producing a very detailed set of tests, since it is effectively a bottom-up way of thinking about the required behaviors of a system. Therefore, this may not be the most effective way of designing the overall functionality of the solution.

To complement TDD, an effective hybrid approach is to combine TDD with a series of end-to-end scenarios pertinent during iterations, or phase of work that guides the way the solution is designed and built. Not only does this approach validates the series of individual tests, it validates the overall solution design; and provides the basis to create a series of automated smoke tests that checks the end-to-end functionality of a release.

Long live Design

So far, we have talked about design in pretty vague terms — certainly with no specific reference to the nature of the solution that we might be building, which is suitable for a generic, methodology-based discussion.

That said, there is certainly room for discussion on more specific topics relevant to design; an acknowledgement that design operates at many levels: from high-level architecture and industry-specific designs through recognized frameworks and design patterns; all the way down to the detailed design of specific components of a solution.

Therefore we recognize that design is a fast-moving software engineering discipline. Indeed, far from stagnating or being left behind by other movements, design is as vibrant as ever.

The following examples represent a sub-set of the various flavors of design that demand specialization, benefit from experience, and are evolving in an exciting and progressive fashion — especially as the delivery methods evolve around us.

Table 9 — Agile takeaways from Design

Focus area	Agile takeaways
High-level architecture	As we have learned, the high-level architecture of a solution is the first true design artifact produced in an agile project. Sets the direction for the rest of the project and as such, is arguably the most important part of design.
UI/presentation/experimental design	One of the most visible and therefore critical areas of a solution to the user community. Clearly, the presentation layer must not allow yet-to-be-implemented features to obstruct the operation of the solution, yet in an agile project any change in the UI of a solution needs to improve behavior through each subsequent release. However, changes cannot be so dramatic as to confuse, or otherwise make difficult the usage of the solution to the most important stakeholder group: the users.
Patterns and frameworks	Includes the recognized design patterns and frameworks, including key design guidelines that offer advice on when to apply them. In addition, designers in this space task themselves with factoring out common components, harvesting a library of reusable components that eliminate redundancy and accelerate the development of the solution.
Information design	Designing, modeling, layering, separation of concerns, automated refactoring, and testing. Yes, these all apply to data, data models/structures, and schemas too. Agile development will not happen effectively, unless the underlying data and information management components evolve and develop in the same way.
Services and events	These days one could not dream of writing an agile paper, or any technical document for that matter without mentioning services. The process behind definition and subsequent evolution of a services catalog and the design of a well-layered and modular service architecture; designing the orchestration layer, including the business and system events that invoke said services, is something that can hugely complement an agile project.

Appendix — References

1. Alan S. Koch. Agile Software Development: Evaluating the Methods for Your Organization
2. Ken Schwaber. Agile Project Management with Scrum
3. Schuh, Peter. Integrating Agile Development in the Real World.
4. Robinson, Michael. Controlling Chaos — Deloitte's Agile Approach to software development (2007)
5. Agile Adoption Rate Survey Results: March 2006 — Ambysoft.com — http://www.ambysoft.com/surveys/agileMarch2006.html
6. Vijayasarathy, Leo R., Turk, Dan. Colorado State University. Agile software Development: A Survey of Early Adopters. Journal of Information Technology Management (Nov. 2008)
7. http://agilemanifesto.org
8. http://www.extremeprogramming.org
9. http://www.softdevteam.com/Spiral-lifecycle.asp
10. http://www.featuredrivendevelopment.com
11. http://en.wikipedia.org/wiki/Waterfall_model
12. http://en.wikipedia.org/wiki/Iterative_and_incremental_development
13. http://en.wikipedia.org/wiki/Agile_software_development
14. http://www.agilealliance.org/home
15. http://en.wikiversity.org/wiki/Agile_software_development

Contacts

Alejandro Danylyszyn
Partner, Systems Integration
National Leader of Systems Integration — Design
Deloitte Consulting LLP
adanylyszyn@deloitte.com

George Collins
Senior Manager, Technology Strategy & Architecture
Deloitte Consulting LLP
georgecollins@deloitte.com

Roland Waz
Manager, Systems Integration
Deloitte Consulting LLP
rwaz@deloitte.com