What Is Agile?

Agile describes a set of principles and practices for delivering software. What we know as agile today has actually emerged over a long period of time.

It is in this new era of knowledge workers that the old organization paradigm no longer works, and the shift to agile principles and practices begins. Now workers must be treated as "volunteers," because like volunteers, these workers can leave and take with them their "means of production": their knowledge. Also like volunteers, knowledge workers do not want to be ordered around. Instead they want to be engaged, they want to participate, and they want to know where they're going and what impact their work will have on others. They want to be challenged, and feel as though their efforts are appreciated. This means that the old approach of commanding workers must be replaced with the newer approach of information sharing and persuasion. As Drucker states, "one does not begin with the question, What do we want? One begins with the questions, What does the other party want? What are its values? What are its goals? What does it consider results? ... The starting point may have to be managing for performance...the starting point may be a definition of results." And so in agile, we focus on providing value to the customer and constantly improving our productivity as we do so.

In 2001, a group of 17 "lightweight" methodologists met in Snowbird, Utah, to discuss their approaches to delivering software. This group of people consisted of representatives from eXtreme Programming (XP), Scrum, DSDM, Adaptive Software Development, and others "sympathetic to the need for an alternative to documentation driven, heavyweight software development process." Jim Highsmith says that most of the agile principles boil down to "mushy stuff" and that "the meteoric rise of interest in and sometimes tremendous criticism of Agile Methodologies is about the mushy stuff of values and culture." After much discussion about said mushy stuff and the ways in which they were creating software, the Agile Manifesto was written:

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 Negotiation
- 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.

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn

Ward Cunningham Martin Fowler James Grenning Jim Highsmith

Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin

Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas

Although the four value statements are discussed in the following sections, we feel that it is important to emphasize the last statement of the Manifesto. The writers were not claiming that agile teams do not write documentation, for example, although this myth surrounds agile teams. Rather, agile teams question the value of the documentation that they have traditionally produced, and whittle that away to only write documentation that is useful and valuable, such as tests or high-level designs, or end-user documentation deliverables such as online help or manuals. When thinking about the last statement of the Manifesto, it is important to remember to stress the items on the left as value statements and employ the items on the right as needed and when they add value. Now let’s explore the Agile Manifesto in more depth.

Individuals and Interactions over Processes and Tools

This first value of the manifesto underscores the importance of individuals and interactions when building software systems. All agile approaches focus on empowered, self-managing teams; autonomous teams do not need the day-to-day intervention of management. Instead, if management protects a team from outside interference, and focuses on removing obstacles in the way of creating product, teams become highly effective and productive.

Agile teams are composed of a mix of skills—everyone necessary to create the product increment is on the team. This means that an agile team is composed of developers, testers, database experts, writers, business analysts, user interface experts, and other skilled professionals. Through working together daily to meet the goals of the iteration, the individuals on the team start to create a shared direction for the product. Ideas overlap. Leadership emerges. Agile team members are able to step in for each other as necessary; they create the system as a team and not as a series of handoffs that we normally associate with a serial process. They apply what they've learned, and the collective knowledge grows; design, quality and productivity is improved as a result.

Humans have tried to create tools to replace face-to-face communication. From collaboration portals, to online communities, to virtual whiteboards, the tech space abounds with products to help us get better at communicating. The Agile Manifesto, by focusing on individuals and interactions, forces teams to rethink the best approaches to communication, realizing the power of team members collaborating in person to solve a mutual problem. And when the team members aren't face to face (this is indeed a global market), tools can help support—not replace—those conversations. Never underestimate the value of a simple phone call.

Working Software over Comprehensive Documentation

Agile projects value working software, which is a profoundly different emphasis from traditional, phased projects. Traditionally, one would measure a project's progress by the percent complete of the functional milestones (that is, analysis complete, documentation complete, code complete, and so on). In agile projects, however, working software is the ultimate quantification of project status. Instead of status meetings where everyone reports, "I'm 90 percent complete," agile teams provide actual working product as a status report, called a "product review," at the end of each iteration. Inspecting software that works enables us to respond appropriately to the true state of the project. Everything is visible; decisions can be made based on product that exists, not documented representations thereof.

Customer Collaboration over Contract Negotiation

Contract negotiation in the traditional sense means that we identify and define everything the customer wants and then draw up the contract that spells out the payment and date specifications. This has resulted in many fixed-price/fixed-scope situations. What we have learned is that this isn't always the best approach for software development projects. Too many teams have found themselves in death-march situations, working 80 hours a week to meet the deadline set forth in the contract—a contract that was estimated and agreed to by somebody other than the team in the first place. Just as bad, customers have found themselves committed to work that no longer makes sense, all in the name of a contract.

Agile "customer collaboration," on the other hand, implies that customers become a part of the development process. To develop the right system, customer feedback is essential. Agile teams value the contributions made by the customer—or the customer representative—and learn to let the customer make the business decision. In turn, customers rely on important technical information that the team can provide in order to make appropriate decisions. Sometimes customers don't know what they want until they can see it.

Responding to Change over Following a Plan

It's much easier to respond to change when the organization and the customer share a clear understanding of the project's status. Focusing on increments of working software and collaborating with the customer allow development teams to more readily respond to change.

In plan-driven environments, all requirements are specified up front, broken down to the task level and estimated. Costs and dates are calculated bottom-up from these very granular tasks. The resulting schedule becomes a baseline for the project and is utilized to measure the project's performance. Therefore, it is very important to stay on task and control scope creep in a plan-driven project setting to limit or eliminate cost overruns or schedule slippage.

The Agile Manifesto does not say that plans are not important. In fact, agile teams are very disciplined and dedicated to planning and to revisiting those plans; because they are involved in the creation of the plans, they are deeply committed. Agile plans follow more of a rolling wave approach using top-down planning, into which we'll take a deeper look later in this book.

What Are the Agile Principles That Guide Teams?

The Agile Manifesto was written along with a set of 12 principles to guide teams.

1. *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*

   This principle underscores the focus of delivering value to the customer. At the end of the day, it is this philosophy of customer satisfaction that drives agile teams.

2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

   In a plan-based project, staying on scope means that scope changes are often discouraged. By allowing for change, we can be certain that we're building products that **help** bring value to our customers. By delivering early and continuously, while also allowing for change, we can stay flexible and thus "agile" in our ability to meet changing conditions.

3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*

   This principle specifies the **timebox** as a way of delivering continuously to the customer. A shorter timescale means that the team doesn't create product for too long a time without getting feedback. **This is a way to mitigate the risk posed by building in long development cycles in which the customer is surprised at the end.** In agile, by delivering very frequently, there are seldom surprises.

4. *Business people and developers must work together daily throughout the project.*

   Business people, or stakeholders, **represent the business needs** for a product. Just like customers, they have certain ideas and beliefs about how the system should work. Also, just like customers, they don't necessarily know what those needs are until they see the system. **Therefore, when business people collaborate with developers, they ensure that their business interests will be better met by sharing contexts, ideas, and answers.**

5. *Build projects around motivated individuals. Give them the* ***environment and support*** *they need, and trust them to get the job done.*

   In agile projects, teams are said to be **self-managing within the timebox;** that is, while building product during the iteration, they should be able to have all the necessary resources at their fingertips and the trust of management to get the job done.

6. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*

   Although seemingly common sense, this statement refers to replacing some documentation with words. Instead of writing detailed design specifications, for example, the team members will work together to discuss and explore ideas about how the software should be built. This is not to say that agile teams do not document—**they do. Instead, it's to say that documentation is not the primary vehicle of communication.** We are aware that many teams are distributed across thousands of miles, and sometimes it may seem that the only way to communicate is through extensive documents; however, many teams have **successfully implemented instant messenger, video conferencing, wikis, updated engineering environments, and other uses of technology to support effective collaboration.**

7. *Working software is the primary measure of progress.*

   There is no better way to understand the status of our project than by inspecting the current state of our system. What has been delivered? How does it work? How can we be sure that it works the way we need it to? **By having a look at the product as it's being built, business people and customers can ask these questions—and get answers by looking at the emerging product!**

8. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*

   Working at a sustainable pace is important to the quality of life for everyone involved in the project. Not only the quality of life, but the quality of the product can suffer if overworked engineers are struggling to focus and finish.

9. *Continuous attention to technical excellence and good design enhances agility.*

Professionalism in anyone's craft is paramount. Technical professionals who write software based on good, simple design are able to respond **to change quickly and effectively.** This flexibility allows for organizations to be agile.

10. *Simplicity—the art of maximizing the amount of work not done—is essential.*

    Simplicity is the **anti-gold-plating** mechanism of agile. Agile teams only build what the customer wants, and no more. This very simple principle ensures that teams are not building functionality that a **customer does not want or will not use.** Once a feature is built, it must be maintained for the life of the system, or the investment must be made to back it out. Either way, this is waste that should be avoided.

11. *The best architectures, requirements, and designs emerge from self-organizing teams.*

    The **collective wisdom** of a team magnificently outweighs the wisdom of one individual. When team members get their heads together, they collaborate to build the best system possible. This agile principle echoes **Drucker's teachings about the knowledge worker.**

12. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

    Although product can be inspected and adapted at the end of each iteration, this principle discusses the **need for process tuning.** Agile teams retrospect in order to determine which processes are working well and which are not. Team members work together to collectively solve any challenges with their processes and focus on improvement over time.

*"Agility is the ability to both create and respond to change in order to profit in a turbulent business environment. Agility is the ability to balance flexibility and stability" (**Highsmith 2002**).*

To some, agile **represents a people approach to project management**; to others, it describes lean development and removing waste from a process. And yet to others, it describes a way to work in a fast-paced world full of churn and change.

Agile Business Objectives

Building innovative products, processes, and business models requires a new approach to management in general and project management in particular. There are **five key business objectives** for a good exploration process such as Agile Project Management (APM):

1. ***Continuous innovation—*** to deliver on current customer requirements
2. ***Product adaptability—*** to deliver on future customer requirements
3. ***Improved time-to-market—*** to meet market windows and improve return on investment (ROI)
4. ***People and process adaptability—*** to respond rapidly to product and business change
5. ***Reliable results—*** to support business growth and profitability

   CPIPR

Continuous Innovation

Developing new products and new services in today’s complex business and technology world requires a mindset that fosters innovation. **Striving to deliver customer value, to create a product that meets today’s customer requirements, drives this continuous innovation process.** Innovative ideas aren’t generated in structured, authoritarian environments but in an adaptive culture based on the principles of self-organization and self-discipline.

Product Adaptability

No matter how prescient a person, a team, or a company, the future will always surprise us. For some products**, changes in the market, technology, or specific requirements happen weekly.** For other products, the timeframe for incorporating changes varies from months to years. With the pace of change increasing and response time shrinking, the only way to **survive is to strive for product adaptability**—a critical design criterion for a development process. In fact, in an agile project, **technical excellence is measured by both capacity to deliver customer value today *and* create an adaptable product for tomorrow.**

Improved Time-to-Market

As the statistics for rapidly shrinking product development times indicate, **reducing delivery schedules** to meet market windows continues to be a high-priority business goal for managers and executives. The iterative, feature-based nature of APM contributes to improving time-to-market in three key ways: **focus, streamlining, and skill development**.

First, the constant attention to product features and their prioritization in short, iterative timeboxes forces teams (product management and developers) to carefully consider **both the number of features to include in the product and the depth of those features.** Constant attention reduces the overall workload by eliminating both marginally beneficial features and marginal requirements of those features. Second, APM—like its lean development counterparts—**streamlines the development process**, concentrating on value-adding activities and eliminating overhead and compliance activities. Third, APM focuses on **selecting the right skills for project team members and molding them into productive teams.**

People and Process Adaptability

Just as products need to adapt to marketplace reality over time, so do people and processes. In fact, **if we want adaptable products, we must first build adaptable teams**—teams whose members are comfortable with change, who view change not as an obstacle to resist but as part and parcel of thriving in a dynamic business environment. The APM principles and framework encourage **learning and adapting as an integral part of delivering value to customers.**

Reliable Results

Production processes are designed to be repeatable, to deliver the same result time after time after time. **Good production processes deliver the anticipated result, for a standard cost, within a given time—they are predictable.** Exploration processes are different. Because of the uncertainty surrounding requirements and new technology, exploration projects can't **deliver a known, completely pre-specified result**, but they can deliver a valuable result—a releasable product that meets **customer goals and business requirements as they become known.** Good exploration processes can deliver innovation reliably—time after time. But while performance measures for production processes can be based on actual **scope, cost, and schedule versus their predicted values**, exploration processes need to be measured differently.

A repeatable process is one in which doing the same thing in the same way produces the same results. ***One that is reliable delivers regardless of the impediments thrown in the way****—reliability means* ***constantly adapting*** *to meet a goal.*

Confusion about reliable and repeatable has caused many organizations to pursue repeatable processes—very structured and precise—when exactly the opposite approach—mildly structured and flexible—works astonishingly better for new product and service development. If your goal is to deliver a product that meets a known and unchanging specification, then try a repeatable process. However, if your goal is to deliver a valuable product to a customer within some targeted boundaries, when change and deadlines are significant factors, then reliable agile processes work better.

Agile Leadership Values

*"In high-performance teams, "the leaders managed the principles, and the principles managed the team." Carl Larson and Frank LaFasto (**1989**).*

If we want to build great products, we need great people. If we want to attract and keep great people, we need great principles.

We live in an age in which the volume of available information stupefies us. On any relatively interesting subject we can find thousands of Webpages, tens—if not hundreds—of books, and article after article. How do we filter all this information? How do we process all this information? **Core values and principles provide one mechanism for processing and filtering information.** They steer us in the direction of what is more, or less, important. They help us make product decisions and evaluate development practices.

Principles, or "rules" in complexity theory terminology, affect how tools and practices are implemented. **Practices are how principles are acted out.** Grand principles that generate no action are mere vapor. Conversely, specific practices in the absence of guiding principles are often inappropriately used. Although the use of agile practices may vary from team to team, the principles are constant. Principles are the simple rules of complex human adaptive systems.

The *Declaration of Interdependence* is authored by the founding members of the Agile Project Leadership Network (**www.apln.org**) and the *Manifesto for Agile Software Development* shown is authored by many of the founding members of the Agile Alliance (**www.agilealliance.org**).

The *Declaration of Interdependence* was developed with project leaders in mind, whereas the *Agile Manifesto* (the oft-used short name) was developed with software development in mind.

Figure 1-1. The Declaration of Interdependence

The Declaration of Interdependence

We increase return on investment by making continuous flow of value our focus.

We deliver reliable results by engaging customers in frequent interactions and shared ownership.

We expect uncertainty and manage for it through iterations, anticipation, and adaptation.

We unleash creativity and innovation by recognizing that individuals are the ultimate source of value, and creating an environment where they can make a difference.

We boost performance through group accountability for results and shared responsibility for team effectiveness.

We improve effectiveness and reliability through situationally specific strategies, processes, and practices.

©2005 David Anderson, Sanjiv Augustine, Christopher Avery, Alistair Cockburn, Mike Cohn, Doug DeCarlo, Donna Fitzgerald, Jim Highsmith, Ole Jepsen, Lowell Lindstrom, Todd Little, Kent McDonald, Pollyanna Pixton, Preston Smith and Robert Wysocki.

Team Manager Vs Task Manager

"Team managers" enable teams to self-manage their own tasks to facilitate the completion of features. "Task managers" focus on task completion as a measure of how well the team is conforming to the project plan. "Team managers" assist their teams (and the broader project community) remain coordinated and effective so that they can succeed. "Task managers" *oversee* teams to ensure that they remain "productive," on task, and don't lag behind the plan.

Three critical values that both summarize those in the *Agile Manifesto* and *Declaration of Interdependence* and highlight key values for agile leaders:

- Delivering value over meeting constraints (Value over Constraints)
- Leading the team over managing tasks (Team over Tasks)
- Adapting to change over conforming to plans (Adapting over Conforming)

A traditional project manager focuses on following the plan with minimal changes, whereas an agile leader focuses on adapting successfully to inevitable changes.

"Commanders know the objective; leaders grasp the direction.

Commanders dictate; leaders influence.

Controllers demand; collaborators facilitate.

Controllers micro-manage; collaborators encourage.

Managers who embrace the leadership-collaboration model understand their primary role is to set direction, ***to provide guidance, and to facilitate connecting people and teams"*** *(**Highsmith 2000**).*

Building Self-Organizing (Self-Disciplined) Teams

Self-organizing teams form the core of Agile. They blend freedom and responsibility, flexibility and structure. In the face of inconsistency and ambiguity, the teams strive to consistently deliver on the product vision within the project constraints. Accomplishing this requires teams with a self-organizing structure and self-disciplined individual team members. Building this kind of team is the core of an agile project leader’s job.

In a self-organized team, individuals take accountability for managing their own workload, shift work among themselves based on need and best fit, and take responsibility for team effectiveness. Team members have considerable leeway in how they deliver results, they are self-disciplined in their accountability for those results, and they work within a flexible framework.

Self-organizing teams are not, as some perceive, leaderless teams. Any group left to its own devices will self-organize in some fashion, but to be effective in delivering results, it needs to be steered in the right direction.

Self-organizing teams aren’t characterized by a lack of leadership, but by a style of leadership.

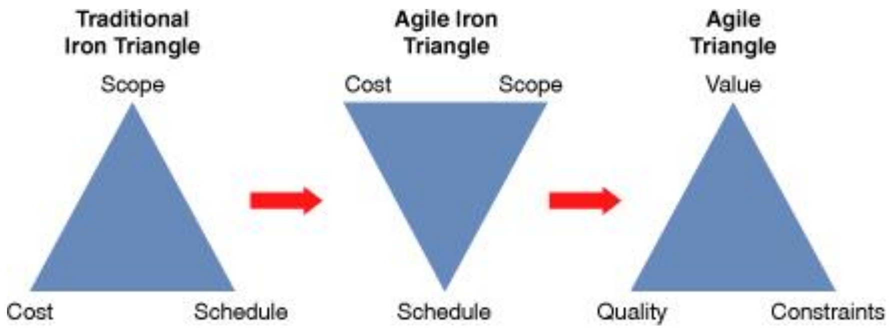
Creating a self-organizing team entails

- Getting the right people
- Articulating the product vision, boundaries, and team roles
- Encouraging collaboration
- Insisting on accountability
- Fostering self-discipline
- Steering rather than control

Agile Performance Measurement

So to attain significant levels of improvement in an organization, we have to address how performance is measured.

The traditional iron triangle of project management, the left-most triangle, consists of scope, schedule, and cost. In many cases scope was the primary driver and cost and schedule varied—although many managers attempt to lock down all three dimensions.



The Evolution to an Agile Triangle

The second triangle represents an early view of measuring agile development where schedule was fixed (timeboxed) and scope was allowed to vary—that is, time was used as a fixed constraint. Unfortunately, this second triangle still conforms to the existing iron triangle measures. Success, in many organizations is still viewed as conformance to cost, schedule, and scope. If conformance to plan defines success, then how will agile projects that adapt continuously ever be deemed successful?

This brings up the third triangle the Agile Triangle. The measures here are value (to the customer), quality (required to deliver continuous value to the customer), and constraints (scope, schedule, and cost). **Constraints are still important project parameters, but they are not the project's goal. Value is the goal and constraints may need to be adjusted as**

the project moves forward to increase customer value. Schedule might still be a fixed constraint, but then scope could be adjusted to deliver the highest value within the schedule constraint**. If we want adaptability we must reward it.** Adjusting constraints to meet value or quality goals helps organizations meet this need. To summarize:

- *Value goal:* Build a releasable product
- *Quality goal:* Build a reliable, adaptable product
- *Constraint goal:* Achieve value and quality goals within acceptable constraints

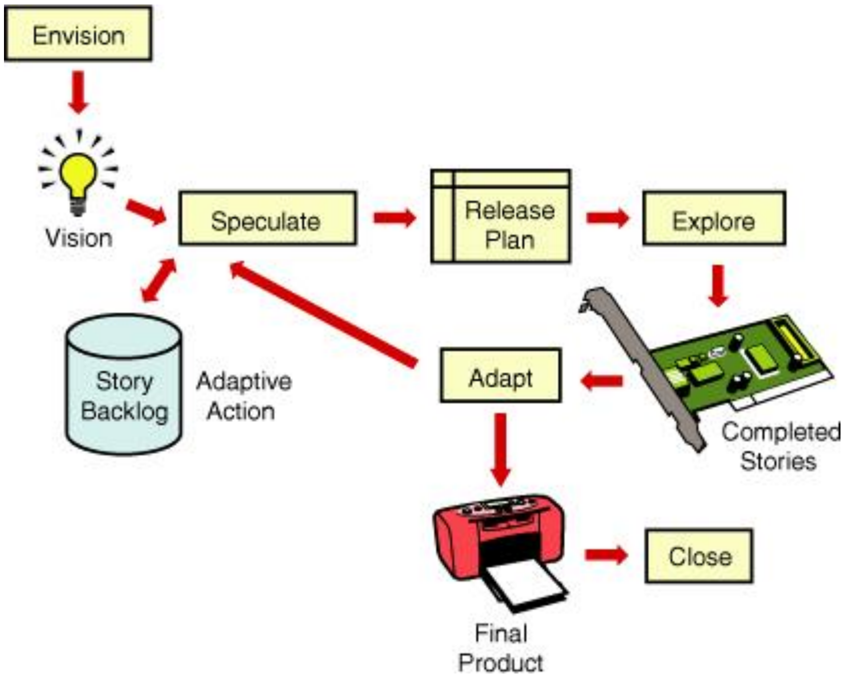
An Agile Delivery Framework

Agile delivery framework:

- Support an envision, explore, adapt culture
- Support a self-organizing, self-disciplined team
- Promote reliability and consistency to the extent possible given the level of project uncertainty
- Be flexible and easy to adapt
- Support visibility into the process
- Incorporate learning
- Incorporate practices that support each phase
- Provide management checkpoints for review

The APM model's structure—Envision-Speculate-Explore-Adapt-Close—focuses on delivery (execution) and adaptation. It is based on a model first described in *Adaptive Software Development* (**Highsmith 2000**).

Figure 5-2. The APM Delivery Framework



The departure from traditional phase names—such as Initiate, Plan, Define, Design, Build, Test—is significant. First, Envision replaces the more traditional Initiate phase to indicate the criticality of vision. Second, a Speculate phase replaces a Plan phase. The word "plan" has become associated with prediction and relative certainty. "Speculate" indicates that the future is uncertain. Many traditional project managers faced with uncertainty try to "plan" that uncertainty away. **We have to learn to *speculate* and *adapt* rather than *plan* and *build*.**

Third, the APM model replaces the common Design, Build, Test phases with Explore. Explore, with its iterative delivery style, is *expelicitly* a nonlinear, concurrent, non-waterfall model. Questions developed in the Speculate phase are "explored." Speculating implies the need for flexibility based on the fact that you cannot fully predict the results. The APM model emphasizes execution and it is exploratory rather than deterministic. A team practicing APM keeps its eyes on the vision, monitors information, and adapts to current conditions—therefore the Adapt phase. Finally, the APM model ends with a Close phase, in which the primary objectives are knowledge transfer and, of course, a celebration.

To sum up, the five phases of agile project management are

1. *Envision:* Determine the product vision and project objectives and constraints, the project community, and how the team will work together
2. *Speculate:* Develop a capability and/or feature-based release plan to deliver on the vision
3. *Explore:* Plan and deliver running tested stories in a short iteration, constantly seeking to reduce the risk and uncertainty of the project
4. *Adapt:* Review the delivered results, the current situation, and the team's performance, and adapt as necessary
5. *Close:* Conclude the project, pass along key learnings, and celebrate.

We can say there are two major collaborative cycles—an Envision cycle and an Explore cycle. The Envision cycle includes the Envision and Speculate phases (product vision, project objectives and constraints, and release plan), whereas the Explore cycle includes the Explore and Adapt phases (iteration plan, develop, and review/adapt). A revised release plan might (and should) be constructed every iteration or two. In a longer project, the entire Envision cycle results might be revised periodically.

Phase: Envision

The Envision phase creates a vision for the product teams that covers what, who, and how. Absent a vision, the remaining activities in getting a project off the ground are wasted effort. In business-speak, vision is the "critical success factor" early in a project. First, team members need to envision *what* to deliver—a vision of the product and the scope of the project. Second, they need to envision *who* will be involved—the community of customers, product managers, team members, and stakeholders. And, third, the team members must envision *how* they intend to work together.

Phase: Speculate

The Speculate phase consists of:

- Gathering the initial broad requirements for the product
- Defining the workload as a backlog of product features
- Creating a iterative, feature-based release plan
- Incorporating risk mitigation strategies into the plan
- Estimating project costs and generating other required administrative and financial information

Phase: Explore

The Explore phase delivers product stories. From a project management perspective there are three critical activity areas during this phase. The first is delivering planned stories by managing the workload and using appropriate technical practices and risk mitigation strategies. The second is creating a collaborative, self-organizing project community, which is everyone's responsibility but is facilitated by the project and iteration leaders. The third activity is managing the interactions among customers, product management, and other stakeholders.

Phase: Adapt

Control and correction are common terms applied to this lifecycle phase. Plans are made, results are monitored, and corrections are made—implying that the plans were right and the actual results, if different from the plan, are wrong. "Adapt" implies modification or change rather than success or failure. In projects guided by the Agile Manifesto value that "responding to change is more important than following a plan," attributing failure to **variation from the plan** isn't productive. A purely ad hoc process fails to learn from its mistakes, whereas the incorporation and retention of lessons learned are key pieces of APM.

In the Adapt phase the results are reviewed from customer, technical, people and process performance, and project status perspectives. The analysis looks at actual versus planned, but even more importantly, it considers actual versus a revised outlook on the project given up-to-the-minute information. The results of adaptation are fed into a replanning effort to begin the next iteration.

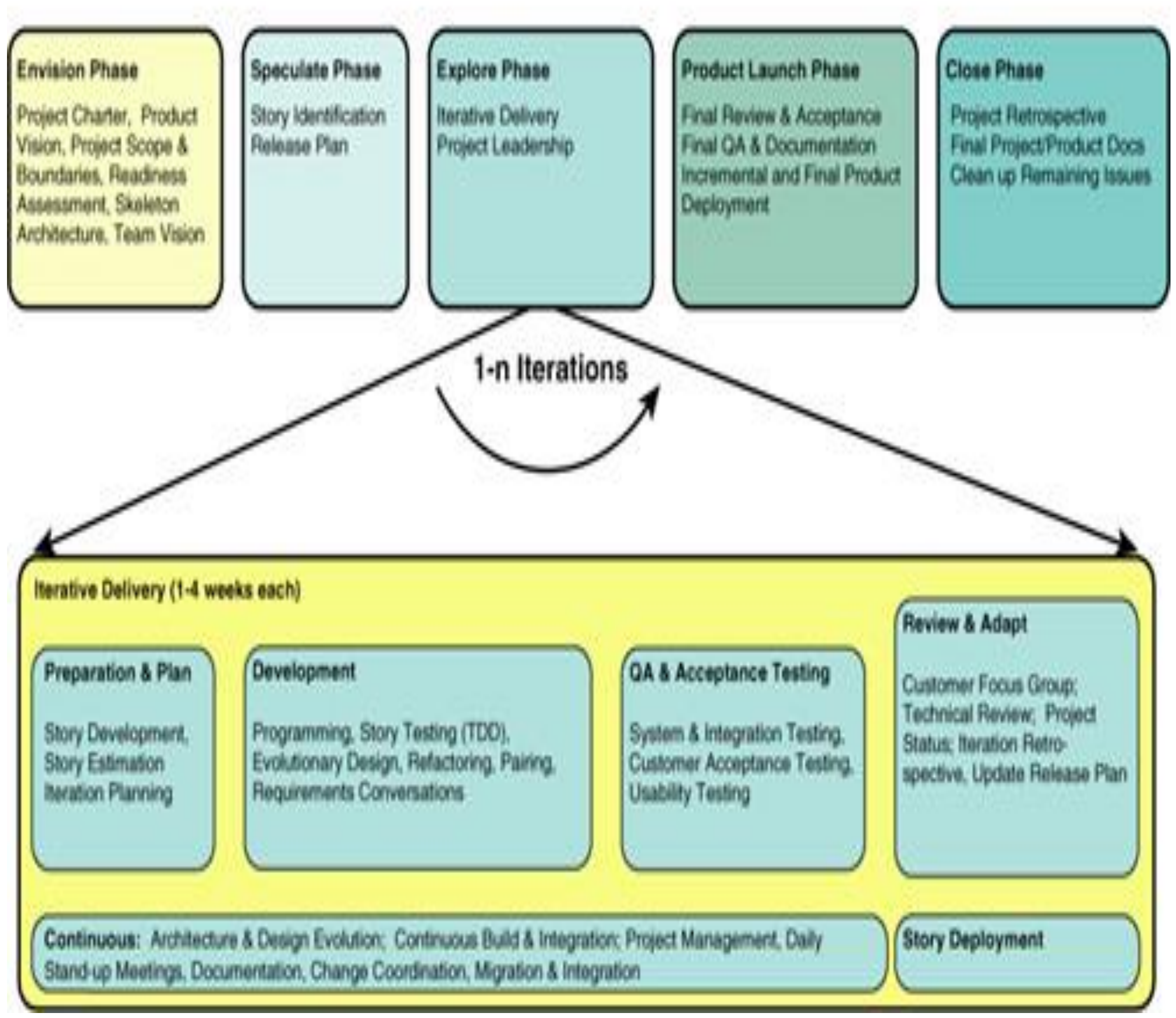
After the Envision phase, the loop will generally be Speculate-Explore-Adapt, with each iteration successively refining the product. However, periodically revisiting the Envision phase may be necessary as the team gathers new information.

Phase: Close

Projects are partially defined by the presence of both a beginning and an end. Many organizations fail to identify a project's end point, often causing perception problems among customers. Projects should end—with a celebration. The key objective of the Close phase, and the "mini" close at the end of each iteration, is learning and then incorporating that learning into the work of the next iteration or passing it on to the next project team.

An Expanded APM Delivery Framework
[View full size image]

An Agile Development Lifecycle


Envision Phase
Project Charter, Product Vision, Project Scope & Boundaries, Readiness Assessment, Skeleton Architecture, Team Vision
Speculate Phase
Story Identification
Release Plan
Explore Phase
Iterative Delivery
Project Leadership
Product Launch Phase
Final Review & Acceptance
Final QA & Documentation
Incremental and Final Product Deployment
Close Phase
Project Retrospective
Final Project/Product Docs
Clean up Remaining Issues
1-n Iterations
Iterative Delivery (1-4 weeks each)
Preparation & Plan
Story Development, Story Estimation Iteration Planning
Development
Programming, Story Testing (TDD), Evolutionary Design, Refactoring, Pairing, Requirements Conversations
QA & Acceptance Testing
System & Integration Testing, Customer Acceptance Testing, Usability Testing
Review & Adapt
Customer Focus Group; Technical Review; Project Status; Iteration Retro-spective, Update Release Plan
Continuous: Architecture & Design Evolution; Continuous Build & Integration; Project Management, Daily Stand-up Meetings, Documentation, Change Coordination, Migration & Integration
Story Deployment

Iteration 0

Iteration 0 helps teams balance anticipation with adaptation. The “0” implies that nothing useful to the customer—stories, in other words—gets delivered in this time period. However, the work is useful to the team. Recognizing that iteration 0 doesn’t deliver customer value pressures the team to keep it short.

Example of work Done in Iteration 0:

- A project to develop a large business software application which has integration with other business applications, may require some data architecture work to adequately define the interfaces with those other applications.
- Teams utilizing unfamiliar technology may need time for training prior to a project's launch and time to establish a development environment.
- Creating a requirement documented demanded by customer prior to signing a contract.

The need for an iteration 0 is usually obvious before the project gets underway and may be included as part of the Envision phase. The key to effectively utilizing iteration 0 is to balance the possible advantages of further planning with the growing disadvantage of lack of customer-deliverable stories. There are always tradeoffs to balance.