Agile Principles and Methodologies

Agile projects use short work iterations and incremental development of products that focus on business priorities and customer value.

In this course, you'll learn about Agile concepts that are fundamental when managing projects, including the eight Agile values and twelve Agile principles. This course also covers the five phases of the Agile project management model, and introduces you to the most common Agile methodologies and frameworks. Finally, this course introduces key activities for managing Agile projects, including creating a product vision and project charter, and best contract and documentation types.

This course is one of a series in the Skillsoft learning path that covers the objectives for the PMI Agile Certified Practitioner (PMI-ACP)® exam. PMI-ACP is a registered mark of the Project Management Institute, Inc.

Table of Contents

1. Agile Principles and Methodologies
2. Understanding Agile
3. Agile Values and Principles
4. Agile Project Management Model
5. Agile Methodologies
6. Scrum Framework
7. Adopting an Agile Approach
8. Initiating an Agile Project
9. Creating Vision and Charting a Project
10. Agile Contracts
11. Agile Documentation

Agile Principles and Methodologies

[Course title: Agile Principles and Methodologies. The presenter is Barbara Waters, PMI-ACP.] Agile Projects are characterized by the use of short work iterations and incremental product development. In this course, you'll learn about the core values and principles outlined by the Agile Manifesto. You'll also learn about some of the more common Agile methods and practices, how to create an Agile product roadmap, and the unique aspects of Agile contracts and documentation.

Understanding Agile

[Topic title: Understanding Agile.] While there are countless ways you can manage a project, most common project management methodologies fall into one of two model types. Either a defined and linear process model, or more of an empirical and iterative process model.

In a defined and linear process model, there's a very linear approach planned for the project work. And work gets carried out following that plan through distinct and complete project phases. You complete one phase of a project, then move on to the next phase, each phase needs to be completed before moving to the next.

In an empirical and iterative model on the other hand, you use actual and empirical data gathered as project work is performed, working in iterations, building on what's been done, and introducing or changing elements as you go. Sometimes performing work in cycles to create the end result. *[A diagram of Empirical and Iterative Model*

displays. Based on empirical data gathered, or requirements, a project is built. Building a project includes the following three stages: Design and Development, Testing, and Implementation.]

Agile Project Management is considered an incremental model for project management. So, instead of project work being completed in a linear model with one final delivery phase, your project work is divided up into increments. And each increment of project work goes through requirements, design, development, and then testing and delivery, before moving on to the next piece of project work. So you're taking it one piece at a time and delivering each piece.

Agile Project Management is additionally considered an iterative model. So not only are you building the pieces of your final product incrementally over time, you're also iterating. *[An example depicting multiple iterations displays. Iteration 0 is Project Setup Plan, Iteration 1 is Plan, Develop, and Test Feedback, which is repeated as needed, and Iteration n is Develop and Test Release Product.]* Meaning in the first increment, you might initially build the minimum features that are required for your product, such as login functionality. *[An example showing login functionality displays. It has a username text box, a password text box, and a Sign in button.]* Then once you've tested that functionality, in the next increment of work you may build more robustness into that feature. *[A Forgot Password button is added to the login functionality.]* So you're incrementally adding different features to your end product, and adding robustness to your features with each iteration.

With Agile being iterative and incremental, the focus is on delivering the highest value items first. So if you're starting with highest value items first, then you know that you've at least tackled the things that are the most important to your stakeholders.

Another characteristic of Agile Project Management, is that you identify issues much earlier. Because testing is ongoing with each iteration rather than just in one testing phase at the end of the project life cycle, issues are realized much earlier in the development process. Feedback is also obtained early and often because at the end of each iteration, you're getting feedback from the project stakeholders. And then incorporating that feedback into the next iteration. Once that feedback has been received, and since you haven't moved too far ahead in the project work, it's a lot easier to make changes as well.

In summary, the Agile Project Management approach is incremental and iterative. Other characteristics of Agile are that it helps focus on the highest value requirements first. Issues get identified earlier in the project life cycle. Stakeholder feedback is received earlier and often. And changes are easier to implement.

Agile Values and Principles

[Topic title: Agile Values and Principles.] So where does Agile come from, and what is it based on? Well, in February of 2001, a group of 17 independent software practitioners came together and authored what is called the Agile Manifesto.

The Agile Manifesto was born out of a need for a software development methodology that was driven by the features and end product, rather than being driven by traditional project management methods that focused more on the project management process itself, and less on the final product. The Agile Manifesto is comprised of 4 values and 12 principles.

The Agile Manifesto contains four values, which are further broken down into two sets, primary values and secondary values. *[The four Primary values are: Individuals and interactions, Working software, Customer collaboration, and Responding to change. The four Secondary values are: Processes and tools, Comprehensive documentation, Contract negotiation, and Following a plan.]* Essentially, all of the values contribute to the success of your project, but the primary values are the values that contribute most to successful projects, and are valued over the secondary values.

So the first Agile value says that we value individuals and interactions over processes and tools. Sometimes in our corporate environments or in team environments, we value certain processes and working with certain tools

more than we value the individuals and interactions that create success. What that means is if we have a tool, for example, that we're using that costs us a lot of overhead, we should be thinking twice about using that tool. We should be thinking more about, how do we foster more communication? How do we foster more collaboration? And if there is a process that ultimately seems wasteful and doesn't really create a lot of optimization and people being creative or doing their best work, then we should rethink that process and focus more on having individuals interact in ways that bring out the best ideas.

The second Agile value says that we value working software over comprehensive documentation. What this means is working software really should be our measure of progress. The ability to start a project and quickly prototype something, or come up with something that works, is a lot more valuable than working for weeks and weeks, and in some cases, months, on documenting everything very comprehensively, especially for software development projects, since there's a lot of change to the final product across the life of the project.

The third value is that we value customer collaboration over contract negotiation. So if you've worked in an environment where you have a defined project plan, it sometimes becomes a big pain to actually have your stakeholders request a change. That change becomes a big contract negotiation. A change request goes in, and then you have to identify whether the change is worth it. In an Agile environment, it's understood that change is constant. Change is going to happen. And so the focus should be on collaboration with the customer to make sure that we understand the impact of the change, the need for the change, and the value that comes out of the change, instead of trying to negotiate whether or not to make changes.

The fourth and final Agile value is that we value responding to change over following a plan. This one is pretty self-explanatory. We need to be able to respond to change and have a system that allows us to do so dynamically, versus following a linear defined plan, where we try to define everything upfront, which is typically going to change over the course and lifetime of a project anyway, particularly in a software development project.

In addition to the four Agile values outlined in the Agile Manifesto, there are 12 principles. These principles should drive all work on your project. The first six principles are satisfy the customer, welcome change, deliver software frequently, work together, motivate individuals, and use face-to-face communication.

It is important to satisfy the customer. The highest priority is to satisfy the customer through early and continuous delivery of valuable software. It's also essential to welcome change, even late in development. Agile processes harness changes for the customer's competitive advantage. It's also important to deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale. And working together is vital. Businesspeople and developers must work together daily throughout the project. Motivating individuals is also key to your project's success. It's important to build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. And face-to-face communication is the most efficient and effective method of conveying information to and within a development team, particularly when time scales and development require the quick transfer of information.

The remaining six principles are that working software equals progress, constant pace, technical excellence, simplicity, self-organizing teams, and reflection. This means that working functionalities in the product are your signs of progress along the project timeline, not the succession of project phases in a spreadsheet. Constant pace is also needed. Agile processes promote sustainable development. The sponsors, development team, and users should be able to maintain a constant pace indefinitely. In addition, continuous attention to technical excellence and good design enhances agility. Simplicity is key. Keeping it simple is an art. Sometimes we tend to think that the more features we build or the more complexity we build into our features, the better, when actually, the best products are the simplest ones that have all the features and function the customer wants, but remain uncomplicated. In addition, the best architectures, requirements, and designs emerge from self-organizing teams. And finally, the principle of reflection brings success, and carries it through future work. At regular intervals, the team should reflect on how to become more effective, then tunes and adjusts working behaviors and procedures accordingly.

In summary, the Agile Manifesto outlines four values that should be kept in mind for project management. The values are, value individuals and interactions over processes and tools. Value working software over

comprehensive documentation. Value customer collaboration over contract negotiation. And value responding to change over following a plan.

The Agile Manifesto also outlines 12 principles that should guide your project. They are, satisfy the customer, welcome change, deliver software frequently, work together, motivate individuals, use face-to-face communication, working software equals progress, constant pace, technical excellence, simplicity, self-organizing teams, and reflection.

Agile Project Management Model

[Topic title: Agile Project Management Model.] While the Agile manifesto values individuals and interactions over processes and tools, and working software over getting caught up and focusing on comprehensive documentation, there is still a recognized value in a model outline for successful Agile project management.

Jim Highsmith, one of the authors of the Agile manifesto, developed the Agile project management model. The model consists of five phases, and it's worth noting that these five phases do not have a linear progression, but rather they are cyclical in nature. Agile is really a cycle and a set of iterations or loops where we're doing a lot of different activities within each cycle, rather than working through a prescribed path with distinct step.

The first phase of the Agile project management model is the envisioning phase. During this phase, you envision what this product is going to be, what type of value you should be delivering to your end customers, and what the vision is for the overall project or product.

The second phase is the speculating phase. In the speculating phase, you start thinking about how to implement different features or functionality that will actually fulfill that vision that you created in the envisioning phase.

Next is the exploring phase. In the exploring phase, you're performing iterations of learning. This is the phase in which you're developing code, you're developing software, you're testing it, you're getting feedback on it, and you're figuring out the way to fulfill that vision.

The next phase is the adapting phase. In Agile, in every cycle, and in every iteration, there is adaptation. As you learn, you adapt and change the plan. You might change your idea of what is higher priority. You can even be changing the way you work in order to optimize and come out with the best ideas, as well as a really effective strategy for working together as a team.

And finally, the last phase is the closing phase. Because the Agile project management model is iterative, that could mean closing a specific iteration, or the whole project. *[An example depicting multiple iterations displays. Iteration 0 is Project Setup Plan, Iteration 1 is Plan, Develop, and Test Feedback, which is repeated as needed, and Iteration n is Develop and Test Release Product.]*

In summary, the five phases of the Agile project management model are, envisioning, speculating, exploring, adapting, and closing.

Agile Methodologies

[Topic title: Agile Methodologies.] There are what seems like countless methodologies that you can use for managing your Agile project. You're specific project type will help determine whether an Agile methodology is suitable and which methodology should be used. You need to consider both criticality and safety and security requirements. You can implement an Agile model rigidly with no deviations, partially tailoring it to specific needs and in combination with another methodology as a hybrid model. Regardless of what you select, there are a number of common ones that you should be aware of and consider for your Agile project.

The first is the extreme programming, or XP methodology. XP is one of the first frameworks that could be considered Agile. *[The iterations involved in Extreme Programming methodology are: Release Plan, Iteration*

Plan, Acceptance Test, Stand-up Meeting, Pair Negotiation, Unit Test, and Pair Programming.] XP is a software engineering centric model and was developed by a number of software engineers who wanted to improve the way they worked together as teams, the way they worked with their stakeholders and the quality of the end product that they were building. XP focuses on the ongoing rapid delivery of software through quick, short releases with a recommended one week iteration. Each iteration results in production ready code.

Lean principles and tools are another approach to Agile project management. Lean principles and tools don't prescribe specific development methods, but instead provide guidelines for streamlining the development process. *[The various Lean Principles are: Identify Value, Map the Value Stream, Create Flow, Establish Pull, and Seek Perfection.]* The guidelines are driven by seven key principles to accomplish this, eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people, and optimize the whole.

Kanban, another Agile framework, is derived from Lean principles and tools. Kanban principles include visualizing a workflow, limiting work in progress or WIP, focusing on the workflow and continuous improvement.

Crystal Methodologies are a family of Agile methodologies named after the colors of crystals of different hardness. *[A sample diagram of Crystal Methodologies displays.]* The Crystal Methodology you should apply depends on the complexity or hardness of a specific software development project. As the size and criticality of the project increases, a methodology with more prescribed steps and artifacts is required to keep control of the development process.

Feature-driven Development, or FDD, provides a client-centric, architecture-centric and pragmatic software development process. All aspects of the software development process are planned, managed, and tracked at the level of individual software features. Five main activities in FDD are develop an overall model, build a features list, plan by feature, design by feature and build by feature.

The Dynamic Systems Development Method, or DSDM is another framework for Agile development. It reflects a business perspective rather than a technical one and has three phases, pre-project activities, project activities, and post-project activities. DSDM requires a large number of artifacts and designates a large number of roles, but has flexibility in the way activities are incorporated. *[A diagram depicting Dynamic Systems Development Method displays. In the Pre-project phase, the initiation of the project takes place. Feasibility is a short phase to assess the viability and the outline business case. Foundation is a key phase for ensuring the project is understood and defined well enough so that the scope can be baselined at a high-level and the technology components and standards agreed, before the development activity begins. Exploration is an iterative development phase during which teams expand on the high-level requirements to demonstrate the functionality. Engineering is an iterative development phase where the solution is engineered to be deployable for release. In the Incremental Deployment phase, for each increment of the project the solution is made available. The Post-project phase assesses the accrued benefits.]*

Model-driven Development, or MDD, requires extensive models prior to production. The Agile version of MDD, or AMDD involves creating incremental models just to add sufficiently detailed support throughout the project life cycle. A team can begin by developing a high-level model of the project requirements, and then develop more low level models for each iteration requirement. AMDD principles include assuming simplicity, modeling with a purpose, valuing content above representation, creating quality work, communicating openly and traveling light.

Disciplined Agile delivery, or DAD, is a process decision framework that is a people first, learning-oriented hybrid Agile model. It has a risk value delivery cycle and it is goal-oriented, enterprise-aware and scalable. DAD is not prescriptive. Rather, DAD provides the guidelines for fitting together appropriate strategies and approaches for a successful outcome.

Test-driven Development, or TDD is a design technique that emphasizes unit testing with tests written before code. TDD components include test cases, test fixtures, test suites and test harnesses. TDD is useful for specification and validation rather than overall system design. *[A flowchart depicting Test-driven Development*

displays. The idea is that before adding a functionality, you write an automated test about how this new functionality that you want to introduce to your system (new code) has to behave. You then run the test and you wait for it to fail. You then again update the functional code to the specification, you write the simplest code, for it to pass. You run the test again and you make sure that this time it passes. At the end you have to refactor (even the simplest code has undesirable properties) and make sure that you have a clean code.]

And finally, Behavior-driven Development, or BDD is often iterative and aims to improve a team's ability to deliver prioritized, verifiable business value using a shared language known as ubiquitous language.

In summary, there are numerous Agile methodologies you can use for your project. Some of the most common ones include, Extreme Programming or XP, Lean Principles and Tools, Kanban, Crystal Methodologies, Feature-driven Development, or FDD. Dynamic Systems Development Method, or DSDM, Agile Model-driven Development, or AMDD, Disciplined Agile Delivery, or DAD, Test-driven Development, or TDD, and Behavior-driven Development, or BDD.

Scrum Framework

[Topic title: Scrum Framework.] As a lightweight framework for Agile project management, we say that Scrum is a framework versus a methodology since Scrum does not prescribe how to implement certain things such as reporting, source control, or even code. Scrum utilizes an iterative, incremental approach which allows for predictability and better risk management. So, Scrum is centered around iterations or sprints that are typically anywhere between one to four weeks in length. Most commonly these days, many teams are implementing a two-week sprint length.

Each sprint or iteration development period has a clear goal consisting of an agreed set of work items to implement during that iteration. So before starting any sprint, the team gets together and agrees on what work items they will complete within the course of a sprint. And at the end of each sprint, the goal is to have some kind of product increment that can be inspected and adapted, gaining feedback from stakeholders.

Each successive sprint then builds on the last. And planning occurs in between the sprints. So, although we do have a general high-level plan for the overall release, sprints also allow us to inspect and adapt and potentially change the plan for the next upcoming sprint based on feedback or based on something we've learned during the course of a sprint.

Inspecting and adapting are central to the success of Scrum. And Scrum includes four events that provide opportunities to inspect and adapt. These are sprint planning, daily Scrum, sprint review, and sprint retrospective.

The sprint planning meeting is the initial meeting where the team will commit to a set of deliverables for the sprint. The sprint planning meeting is an opportunity to inspect and adapt because at this meeting, the product owner is able to provide the team with some feedback based on either the last sprint or based on things that have changed since the last sprint. It's an opportunity to potentially reorder or reprioritize the requirements backlog.

The daily Scrum is another opportunity to inspect and adapt as a team. Everyday there's a Scrum meeting, also known as a daily stand up. Typically 15 minutes maximum in which team members discuss what they've done, what they are going to work on and any roadblocks or impediments to their work. Each team member answers three main questions. What work have I completed since the last Scrum and why? What do I plan on completing between now and the next Scrum? And do I have any roadblocks or problems that the team can help me overcome? On many occasions, the why part of the first question is skipped since everybody should know why they've worked on specific items. Those items typically line up with sprint goal and have been agreed on at the beginning of the sprint. However, in the case that something has delayed or changed priorities, it's good to make sure that everybody understands why priorities changed with work items and what challenges are being experienced?

The sprint review, which happens at the end of a sprint, allows the team to review and demo the product of their work to their stakeholders and to the product owner. It's also another opportunity to inspect their deliverables, to inspect what they've been working on, and adapt if necessary.

And lastly, the sprint retrospective is the fourth event that provides opportunities to inspect and adapt. The sprint retrospective is a meeting in which the team can get together and talk about what went well, what didn't go well, and what action items they can take in order to improve as a team.

In summary, the primary goal of the Scrum framework is to have opportunities to inspect and adapt as work progresses in your Agile project. The four Scrum events that provide opportunities for this are, sprint planning, the daily Scrum, the sprint review and the sprint retrospective.

Adopting an Agile Approach

[Topic title: Adopting an Agile Approach.] When considering the adoption of an Agile approach, there are some factors to consider. The organizational structure, the project type, a customer's role, and the team composition all influence how Agile can be adopted. It's also important to consider how an organization approaches change. An organization that focuses on controlling change to minimize potential disruption may struggle with Agile methodology. If the focus is on welcoming change, an Agile methodology will be easier to support.

Regardless of the individual factors within your organization, the ADAPT steps developed by Mike Cohn identify five necessary requirements for successfully transitioning to Agile. These five steps are create awareness, increase desire to support change, develop ability, promote successes, and transfer the Agile mindset throughout the organization.

The first step of the ADAPT process is awareness. Your organization needs to be aware of the problems that it has currently. What are some of the process issues that are occurring? What are some of the inefficiencies? What are some of the issues that we want to improve upon? Also, awareness around how Agile methodologies can help us achieve those goals or fix those issues is key knowledge to have.

The second step is desire. There definitely needs to be a desire from the organization, or from enough people in the organization, to make the change and to make the shift towards Agile methodologies.

Ability is the next step. The actual ability for an organization to support an Agile approach is vital. In some cases, organizations may not have the ability to change because of certain factors. There might not be enough co-location of teams, or the engineering structure may not allow for things like test-driven development or unit testing. It could also come down to an organization just not currently having people with the right skill sets to adopt Agile.

Next, after deciding whether you have the ability to actually move to an Agile methodology, you want to start promoting it. This includes promoting Agile across the organization, not just specific to your project. And it also includes talking about the benefits that you expect to gain.

And the fifth and final step is transfer. This is the step where you finally make the transition and start implementing Agile methodologies or frameworks in your organization.

Moving to Agile is definitely not an easy thing to do. It includes change and requires successful change management. It includes a lot of understanding of why you're moving, and why you're trying to change the way that you've done things. It involves cooperation with governance, constraint and feedback, and organization and learning. However, the benefits for organizations that have moved to Agile have been very clearly measured, and show a lot of improvement in areas that people and organizations care about.

In summary, the ADAPT steps are required when transitioning to an Agile approach. The steps include awareness, desire, ability, promote, and transfer.

Initiating an Agile Project

[Topic title: Initiating an Agile Project.] When initiating any project, it's a good idea to first make sure that you understand the opportunity and whether it makes sense for your business to pursue that opportunity. So the key activity for initiating an Agile project is establishing a business case or business justification.

The purpose of having a business case is multifold. A business case helps ensure that everyone understands what they are trying to achieve. By establishing a business case for the end product, it becomes very clear across the organization from executives to managers, to marketing, to sales, to the development team, what the goal is for developing this product or working on this project.

The business case also helps simplify decision-making by setting clear objectives and parameters. By understanding those parameters, you can also start to create a framework for making decisions around the product. A business case also helps keep the project team focused on meeting the customer's overall objectives for a project. By defining the value that the customer sees or that the customer is seeking, it becomes a lot easier for the team to focus on those things, versus perhaps features or functionality that the team considers as valuable.

Finally, the business case provides the key criteria for judging the success of a project.

So what exactly is included in a business case? Recommended components of a business case include an opportunity, goals, strategy, project vision, milestones, investment, and expected payback. Some components of a business case can include an opportunity. So an opportunity is a chance to create value or meet a business need. For example, in a hypothetical project, the stated opportunity might be to update the website with the company's revised corporate image, to improve functionality, and bring it in line with industry standards.

Another component in a business case are the goals which provide the reason why a project should be implemented. They also inspire a team and they need to be specific, measurable, attainable, and realistic.

Strategy is another component that should be included in a business case. Strategy is the plan that will help the customer achieve the specified goals.

The project vision, which specifies what the customer hopes a project will have achieved once it's completed, should also be included in the business case. The project vision also helps an Agile team understand the purpose of the project.

Milestones, which identify significant points in the development and shipping of a project or product, should be in your business case. Milestones help focus the team and make it easier to track progress.

The investment specifies what a customer will need to invest in a project based on each milestone.

And finally, the expected payback is included. The expected payback identifies both the financial benefit or return on investment that a customer can hope to gain from a project. As well as what the payback is for the company as a result of the project.

In summary, there are seven different components that should be identified in a business case when you are initiating an Agile project. They include opportunity, goals, strategy, project vision, milestones, investment, and the expected payback for both the customer and the organization.

Creating Vision and Charting a Project

[Topic title: Creating Vision and Charting a Project.] Any project requires directions to steer the team toward success. Creating the product vision and developing the project charter are two key activities that help guide the project team to a accomplishing the goals of your Agile project.

The product vision is a compelling statement about why you're building a product, the benefit the product brings, who you're building it for and why you are uniquely positioned to develop it. A product vision also describes how a project can capitalize on the opportunities and fulfill the goals of the business case. A product vision should provide all the stakeholders, including the development team, with a common understanding of what's required, without limiting that team's creativity in finding solutions.

An important part of creating a product's vision is to interview the people who are stakeholders of the product about their vision for the product. The interview should focus on establishing minimum and maximum requirements for the product and investigate the relationship between the project's potential risks and benefits. The vision statement doesn't just describe a product's features, it focuses on how a product will add value. It should motivate the development team and help them envision the final product.

Project charters are great way to ensure that the intentions, outcomes and priorities are clear to both the stakeholders and the team. It's a good way to try to establish alignment on key elements of a project, especially early on. A project charter can contain many elements. But one of the most common ones is the vision statement, which defines the why of the project. This is the purpose, or the reason, why this project exists. The objectives or mission are the what of the project, and these state what will be done in the project to achieve the vision. It should also identify who the customer is and what problem is going to be fixed with the end product.

The success conditions, or success criteria, are a set of outcomes or results that define success of the project for management or stakeholders. A charter can also contain priorities and compromises. It's a good idea to understand what a stakeholder will and will not accept as a compromise before starting out on a project. And it's also important to know what risks the project has. As well as to think about the acceptable workarounds. Risks and mitigation strategies should be included in the project charter. Team roles should also be included. Who will be doing what should be documented, along with a high-level overview of what each person is responsible and accountable for.

The most important thing to note about a project charter in an Agile environment is that it's a one page document. It needs to include these high-level essential items that talk about what creates success for a project. But it shouldn't go on and on for many, many pages.

In summary, a project charter should be no more than a page long. But should still include the following elements. A vision statement, objectives or a mission of the project, customer, problem, success conditions or criteria, priorities and compromises, risks and mitigation and team roles.

Agile Contracts

[Topic title: Agile Contracts.] Contracts help organizations manage their risks and resources. In traditionally managed projects, the most commonly used contract is a fixed-price contract.

Fixed-price contracts aren't generally the best for Agile projects, as they assume a fixed scope. They also place risk on the team implementing the project. Because if the work ends up exceeding the estimate or if requirements change, which they frequently do, the team has to bear that loss and still continue with the promised scope. Fixed-price contracts may also take focus away from what's best for the product and customer. And instead, developers become bound by legal limitations that are no longer in the customer's best interest.

In theory, a possible solution is for a contract to specify the overall business goals that a project team must enable a customer to meet. Instead of specifying detailed technical requirements. However, this isn't always practical. Because in terms of legal requirements, business goals need to be defined in monetary terms, which can be difficult.

So what types of contracts are suitable for Agile projects? You can use different contract types for each of the Agile development phases. So for example, in the initiation phase, you may use a service contract. Instead of

measuring a specific deliverable like product development, a service contract may measure the time, cost, and materials spent on coming up with for example, a roadmap or a story map within that project.

The next phase, or the development phase, may be a series of fixed-price contracts for each iteration. Some variants on the fixed-price contract include fixed-range contracts and fixed-price per user story.

Cost-reimbursable or time-and-materials contracts operate on a pay-as-you-use basis. So it's in the client's hands to decide whether or not to pay the team to spend more time or more resources on developing additional requirements, making changes or enhancements, or fixing issues.

A not-to-exceed with fixed-fee contract can protect both the development team and the customer. It's made up of a fixed-fee base as well as a not-to-exceed condition. So it may be, for example, that our fixed-fee base for this project is x dollars and then the not-to-exceed condition is we will not exceed 15% of that base.

An incentive contract can also be used and can motivate a development team by offering rewards for good performance. The types of incentive contracts include, fixed price with incentive, cost-reimbursable with award fee, percentage increase or decrease based on time, and a fixed price user story with additional hourly rate.

In summary, while regular fixed-price contracts are not suitable for Agile projects, many other types of contracts are. They include service contracts for the series of fixed-price contracts, cost-reimbursable or time-and-materials contracts, not-to-exceed with fixed-fee contracts, and incentive contracts.

Agile Documentation

[Topic title: Agile Documentation.] One of the key principles of the Agile Manifesto is to emphasize Working Software over Comprehensive Documentation. So that means we prefer having Working Software be our primary measure of progress versus Comprehensive Documents. Agile approaches focus on keeping documentation light or having just enough documentation to help you move forward and make progress. Most agile teams use specific forms of documentation. And they favor those that are simple and highly visual.

So for example, a Project Backlog Iteration, Sprint Backlog User Story Cards, and Burnup and Burndown Charts are very common documentation forms used in Agile teams. All of these are very simple to use and are highly visual, and it's shown that visual communication is a lot more effective than communication via pages of text.

Sometimes people feel that documentation is required for every process in every project. Reasons might be that it's needed for Managing risk, Following a process, Information sharing or Detailing requirements. These are not valid reasons for generating comprehensive documentation, and can actually hinder, rather than help in Agile project success.

In an Agile environment, you manage risk better by focusing on delivering the highest value items first, versus by creating very comprehensive documents. And you can still follow a very specific and very effective process without having comprehensive documentation. In addition, sharing information among teams can happen in collaborative environments where people are talking verbally and communicating without the need for documents. And from an Agile perspective, thoroughly detailing requirements early in a project is wasteful. It's too soon at that point for customers to know what they really want at a detailed level. So requirements are likely to change.

We know that early on in a project is the point where we know the least that we are ever going to know about that project. That's the worst point in the project life cycle to try and create very detailed comprehensive documents about what's required.

With all of that being said, there are of course valid types of documentation that are quite useful and contribute to the success of an Agile project. For example, the Vision Statement defines the ultimate goals of a project and the why. Why we're building this project.

A Project Overview summarizes critical information about a project such as the Vision, Technologies used, and Operating processes. It should be just a few pages long.

Requirements Documentation, of course, are helpful, even essential to have. Product requirements may be documented in many different forms, including for example the Product Backlog.

Acceptance Testing Documents are also helpful. Acceptance criteria is used by testers to determine whether the requirement has been met. Acceptance Testing Documents specify those acceptance criteria, and they help tell developers whether their work is complete, and whether it meets expectations of stakeholders.

Support Documentation is designed for those who will become responsible for supporting a project and a product once it's released. It may include Problem Escalation Procedures, a list of Important Contacts, and a Troubleshooting Guide.

User Documentation may include Training Manuals, User Manuals and Quick Reference Cards. It's designed to equip users to use a product and to find answers to their questions about it without having to call support staff.

So in Agile environments, there are still documents that are generated. However, the focus is on creating really effective communication, not documenting what doesn't need to be documented. And not spending too much time documenting things that you know will change later on in the project.

In summary, helpful documentation to create for an Agile project includes the Vision Statement, the Project Overview, Requirements Documentation, Acceptance Testing Documents, Support Documentation, and User Documentation.