Agile Project Scheduling and Monitoring

Wouldn't it be great if projects managed themselves once you got them started? But this isn't the case. Successful Agile project management involves adequate scheduling and monitoring, which helps identify any adjustments that need to be made, and ensures effective time management.

In this course, you'll learn about managing projects with Agile project scheduling activities like setting work in progress limits and implementing project buffers. This course also covers recommended project time management processes for monitoring and tracking progress at both the iteration and project levels. This course also introduces you to key risk management and product quality, testing, and integration strategies you can use when managing Agile projects to help ensure your project creates and maintains the intended value as it progresses.

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

Agile Project Scheduling and Monitoring

[Course title: Agile Project Scheduling and Monitoring. The presenter is Barbara Waters, PMI-ACP.] Projects require a solid plan from the very beginning, to ensure the project work focuses on the end goals. But once project iteration work begins, you'll need to monitor and track progress to ensure everything goes as planned. And identify when things don’t, so you can fix it fast. In this course, you'll learn about Agile scheduling methods. As well as good practices for measuring and monitoring your Agile project progress.

Work in Progress Limits

[Topic title: Work in Progress Limits.] As project work progresses, it's critically important to achieve an effective state of flow and productivity for the Agile Project Team. Setting work in progress limits, or WIP limits, help do this.

There are a number of benefits to setting and utilizing WIP limits. For example, by setting WIP limits we help prevent bottlenecks. Once we have a limit on how many things we can be doing at one given time, we start to see where might be areas that are bottlenecks and start to remove them. And by doing so, we improve throughput.

Also just by the fact that we're starting to look at what our work flow looks like, we start to see steps in our work flow that may not be adding value to the end product, which also improves throughput. By setting WIP limits,

we also look at what is the minimum and maximum amount of work that we want the team to be working on concurrently. We want the limit to be realistic so that we don't start to try and multitask within the team which ultimately causes more damage than it does good. Usually, we'll set those limits and agree on them during the planning parts of the project.

Also, by having WIP limits, we can start to identify areas of idleness or overload. So for example, if we start to see an iteration that has an overload of tasks associated with it that aren't progressing, we know that there's something going wrong. And there's something that's contributing to either idleness in one area or too much to do in another area. And it may be, for example, we don't have enough developers, or QA testers to help move the process along. By identifying those areas of idleness or overload or inefficiencies, we can start to fix those issues and help the team get what they need in order to get the work done.

So the goals of setting WIP limits are having consistently sized tasks. By setting a limit on how many things can be done at one time, we want them to be similarly sized so that it makes sense to have a limit on, for example, two items at one time or four or five. Also, we want to make sure that those individual tasks don't take longer than a couple of days.

Another goal of setting WIP limits is to reduce idleness. So for example, if we have as a team, an overall and collective WIP limit and we start to see that we're not able to move things forward because there are some things that are stuck, then team members can collaborate and help each other to move those items. And start to help other team members in getting their work done. Another goal is to protect the quality of work. So we want to ensure that team members aren't rushing through work and multitasking. By setting those limits, we're trying to give the team the feeling that you should be spending your time as you're doing these tasks and doing the quality work so that you can move it along and then pick up a new item and focus on that new item. So we're establishing focus and we're establishing a state of flow within the team.

In summary, setting WIP limits helps prevent bottlenecks, improves throughput, helps determine the minimum and maximum amount of work, ensures limits determined and agreed upon, and helps you identify areas of idleness or overload.

Scheduling Buffer Types

[Topic title: Scheduling Buffer Types.] There are a number of challenges associated with creating a schedule for your project.

One of those challenges is achieving speed and reliability. So in order to achieve both speed and reliability, we want to make sure that we have a schedule that doesn't give us too much time to execute and implement tasks. But at the same time, doesn't squeeze us into too little time to execute or implement those tasks. And we still want to try to achieve early finishes. Since as a project manager, or someone responsible for the outcome, it would be great if we could save costs and free up resources, both for the team, but also for us to be able to innovate and start to think about other considerations for our customers.

Another challenge is that some project managers find it difficult to manage progress of the team once the team members start to become more self-organizing. This is especially true for team members that are just starting to understand the concept of self-organization, or that haven't worked in an Agile environment for long enough. Figuring out how to introduce the right buffers into your project schedule can be challenging, especially since once we try to introduce buffers, we start to see some phenomena occur. One such phenomena is Parkinson's Law, *[A graph displays. The x-axis depicts Time. The y-axis depicts Effort.]* which states that work expands to fill the time available. So that means if we actually allow for too much time for tasks, or for a project, we can start to see that the work will actually expand to fill that time. That can manifest in procrastination, or anything else that starts to cause us to take more time than it is necessary to complete tasks.

Also Murphy's Law, which is that whatever can happen, will happen. With Murphy's Law, we start to expect delays, especially if we start to squeeze time and not allow enough time for projects or for tasks. And lastly,

Student Syndrome. With Student Syndrome, this basically means that team members leave the work until the last possible moment, *[A graph displays. The x-axis depicts Activity time. The y-axis depicts Effort.]* especially with tasks that are difficult or tasks that they haven't actually implemented before. This can cause delays *[The graph exceeds Milestone Due Date.]* that result in missed deadlines. So the question is, how do we introduce the right amount of buffer into our schedule that both protects our release date, but also gives us enough time to complete tasks? When people start to introduce buffers into a project's schedule, they can sometimes inadvertently cause additional problems by allocating too much work. When we allocate too much work to people, we hope to create a sense of urgency. But actually, that strategy backfires. Instead of creating a sense of urgency and keeping everyone busy, we're more likely to miss deadlines and start to compromise the quality of our work, because people are rushing to get done.

There are, however, four common types of buffers that can be used effectively in Agile projects. The first such type are feature buffers. With feature buffers, we introduce a buffer into the implementation of each feature. So we add extra time than was originally estimated for the implementation of a feature. Another type of buffer is a project buffer. A project buffer is extra time added to the end of a project's schedule that is intended to protect the release date. This extra time added to the end of the project's schedule is supposed to protect us from any delays, or anything that slipped along the way as we're implementing the project. Thirdly, there's a feeding buffer, which helps ensure that task or cross team dependencies don't result in delays. A feeding buffer is very similar to a project buffer, but it's a buffer that gets introduced in between any types of cross team hand offs. So if one part of the project is primarily being implemented by one team, and that team hands off to another team, we introduce a buffer in the hand off time. And lastly, we have a resources buffer. The resources buffer requires people and skills in place for longer than it's estimated they'll be needed. So for example, *[A Resource calendar for the months June, July, and August displays. On the left-hand side of the calendar are listed three resources: Human resources, Material resources, and Skill resources.]* if we need somebody in place for a certain portion of the project, we'll add a certain amount of time to that original estimated time, just to give some extra flexibility.

In summary, there are four common types of scheduling buffers you can use in your Agile project. They are feature buffers, project buffers, feeding buffers, and resources buffers.

Calculating a Project Buffer

[Topic title: Calculating a Project Buffer.] There are four statistical concepts and practices you should understand before calculating a Project Buffer.

The first one is Standard Distribution of Task Duration. This standard distribution is determined using random variances in the actual completed task duration. So you'll calculate the expected estimate for the task duration and then the variances, and you'll figure out the standard distribution between the actual estimate and those variances. Another concept is Estimating at 50% Confidence. This means that we come up with estimates that will likely be accurate at a 50% probability, so we're 50% confident that those estimates are correct. The third concept to keep in mind is Estimating at 90% Confidence. This means that those estimates will likely be accurate 90% of the time, where we're 90% confident of those estimates. Finally, there is the practice of Using Both 50% and 90% Estimates. When we use both the 50% and 90% confidence estimates we can come up with a value or a range that's called the local safety, and it's where our estimates will probably fall somewhere between what we think is 50% most likely to happen and 90%.

Two common methods for calculating buffer are the Critical Chain Project Management, or CCPM method, and the Square Root of the Sum of the Squares Method. Critical chain revolves around the resources needed to complete the project tasks. And we create a critical chain by looking at the sequence of the longest series of dependent tasks in our project. Then we add buffers at the end of the project in order to absorb any delays. And this helps protect the project completion date.

The Square Root of the Sum of the Squares Method was developed by Mike Cohn, and is often more accurate than the critical chain method, especially for agile projects. This method assumes that the difference between the

50% and 90% confidence estimates for a task is approximately two standard deviations from the actual value. So to calculate the Square Root of the Sum of the Squares Method, first you take each task's 90%, or worst case estimate. Next, subtract its 50%, or average estimate. In the third step, you divide the result by 2, and square that result. In step 4, you calculate the sum of the results for all the tasks. Finally, the fifth step is to calculate the square root of that result and multiply by 2 for the final result.

There are two main guidelines to follow when calculating the Project Buffer. The first is, plan a buffer for projects with more than ten user stories. Using the Square Root of the Sum of the Squares Method produces the most adequate buffer. The second guideline is ensure that the Project Buffer represents at least 20% of the total project duration. If it's smaller than 20% it may not provide sufficient protection against delays.

In summary, to calculate the Square Root of the Sum of the Squares Method, first you take each task's 90% or worst case estimate. Next, subtract its 50%, or average estimate. In the third step you divide the result by 2, and square that result. In step 4, you calculate the sum of the results for all the tasks. Finally, the fifth step is to calculate the square root of that result, and multiply by 2 for the final result.

Monitoring Iteration Progress

[Topic title: Monitoring Iteration Progress.] Monitoring your project progress is critical for keeping track of how things are going, and there are several monitoring tools and techniques you can use to do so. Two ways you can monitor progress are Daily Standups and Backlog Grooming.

Daily Standups are regularly scheduled meetings that agile teams usually participate in, that are pretty short and simple. The objective of Daily Standup meetings is to share what's been done since the last standup, what team members will be doing until the next standup, and to talk about what impediments are blocking the team from being able to make progress on any given task or issue. These meetings are intended to be short and simple, and to only talk about those three things, and to avoid long-winded discussions about technical challenges, or technical implementation. A good scrum master or facilitator of this meeting will identify which conversations need to happen offline, or outside of this meeting, and will help the team to table them and identify them for later discussion. Standup meetings are typically effective in single or smaller project teams, and not when combining a number of teams together.

The process for Organizing Daily Standups is straightforward. First, set a daily meeting at a time when the entire project team is available. To keep the meeting short, set a time limit, such as 15 minutes, and have members remain standing. Team members should speak in turns, and a physical task list should be updated accordingly, for instance, on a whiteboard. The three standard questions should be covered. These are, what's been done since the last standup? What will be done until the next standup? And what are the impediments or blocking issues? Follow-up conversations can be scheduled.

Backlog Grooming, also known as Product Backlog Refinement, provides an opportunity for the development team to revisit a product backlog. This allows the product owner to reassess the priority of user stories, and the team to ask clarifying questions related to the user stories. The team can also refine the estimates for upcoming work. The backlog refinement meeting is derived from the road map and its requirements. During backlog refinement or grooming, we're prioritizing the list of work with our business owners, so it allows for that interaction along the course of a project, and not only at the release planning phase. Once we've done the Backlog Grooming, we ensure that the most important items are listed at the top of the backlog after each one of these meetings, so the team knows what to pick off of the backlog, work on, and deliver first.

The first step in Backlog Grooming includes gathering and analyzing the user and customer feedback from our previous iterations that have been delivered. Next, we integrate any learnings that have been incorporated based on feedback or input or even insight into certain data and analytics. After that, we determine next steps. Then we develop small stories by either breaking up upcoming stories or creating new stories based on what we've learned. Finally, we prepare the stories. So we ensure that those stories are clear and ready, acceptance criteria is included, and that we've discussed with the product owner what the success criteria is for those stories.

There are some considerations to keep in mind when grooming. First, it should be a collaborative effort between the product owner and the development team. Grooming should be done prior to or during development. Also, Backlog Grooming should constitute between 5 to 10% of the effort in each sprint. It should be kept as simple and short as possible.

In summary, Daily Standups and Backlog Grooming are two effective ways for monitoring project progress. The steps for Backlog Grooming are gather and analyze user and customer feedback, integrate learnings, determine next steps, develop small stories, and then prepare stories.

Tracking Progress

[Topic title: Tracking Progress.] Tracking project progress is important so you have a record of how the work being performed aligns with what was originally planned. One item that should always be kept updated is the project release plan.

A release plan typically contains the following components. A list of high-level deliverables, the release goals, the high-level user stories, a set of priorities for the release, an estimate of the number of iterations, and a project completion date. Release plans are updated when work is completed to show which items have been developed, tested, and are ready for release. The release plan is also updated when changes are made to requirements and priorities. A customer's requirements and priorities may change over time, and product backlog items will also change. Finally, changes to project estimates also need to be updated in the release plan. Estimates may be changed after the first few iterations are complete.

Tools for tracking and communicating progress at the iteration level include, iteration burnup and burndown charts. These tools are specific to the current iteration. Tools for tracking and communicating progress at the project or release level include release burnup and burndown charts, parking lot charts, and defect reports.

Release burnup and burndown charts indicate the amount of work outstanding in a full project, instead of in a single iteration. They plot the number of story points in a project against the number of iterations expected to complete the work. *[A Parking lot chart displays. It contains six segments: Releases, Sprint, Themes, Tags, Assigned, and Story type. The Themes segment is open.]*

The parking lot chart tracks project progress at the levels of themes. It provides a high level, but comprehensive overview of the scope of the project. Parking lot charts include *[Theme 22 is zoomed in.]* the name of the theme, *[that is Theme 22]* the number of user stories in the theme, *[that is 56 stories]* the total number of story points assigned to stories in the theme, *[that is 75/100 story points completed]* the percentage of the product that the theme represents, *[that is 14% Percentage of the product and 20% Percentage of work completed]* and the percentage of work completed. *[that is 75%]*

Defect charts include different types of information. *[A Defect Report chart displays. The x-axis depicts Date. The y-axis depicts Defects.]* They contain information about open or unresolved defects in a project, as well as the daily open or kill rates. They also show open defects measured over time and according to their status. Lastly, they show open defects measured by priority over time, or how often a build fails and when. For example, this could be at the start or end of development.

In summary, there are several progress tracking tools you can use for your project. At the iteration level, you can use iteration burnup charts, and iteration burndown charts. At the project level, you can use release burnup charts, release burndown charts, parking lot charts, and defect reports.

EVM for Agile Projects

[Topic title: EVM for Agile Projects.] Projects are supposed to create value. And you need an established way to calculate that value. Earned Value Management, or EVM, is a project management technique that's used to

measure our performance in terms of cost and schedule in our projects. Metrics are another way to measure and provide early warnings of any performance issues that allow us to make timely adjustments. And to also communicate progress to stakeholders.

Agile EVM uses the Scrum framework artifacts as input. The traditional EVM calculation is used and expressed in traditional EVM metrics. *[A graph displays. The x-axis depicts Time. The y-axis depicts Cost.]* Which basically is the schedule performance index or SPI and the cost performance index or CPI. The required parameters in order to calculate the Agile earned value management metrics are actual project costs, an estimated backlog in story points, a release plan so that we have a total of our story points that are estimated for the entire project and an assumed velocity for the team.

So when we did our planning, we expected that over time we would be achieving this many story points per unit of time. *[that is t1 and t2 on the x-axis.]* Another is actual cost, or AC. *[A purple line depicting Actual cost displays, which increases from 0 to t2.]* So how much have we spent so far along the course of the project in order to achieve the completed story points? The orange line shows us the earned value. *[Earned value increases from 0 to t2. Earned value is greater than actual cost till a point that lies between t1 and t2. At that point, earned value intersects actual cost. And beyond that point, earned value is lower than actual cost. Beyond t2, earned value increases and becomes greater than actual cost.]* So how many story points have we actually completed at any given point in time? *[A blue line depicting Planned value displays. It is a 45 degree straight line from 0.]*

When we talk about value in earned value management, we really mean how much work has been completed. And we calculate that and express it in terms of value. In Agile projects, earned value management is actually a good way to calculate how many story points have been completed over time. It is quite easy to see whether story points are completed or not. In some software projects, if we're not dividing up the work into the concept of a story point or a user story, it may be much harder to calculate earned value. So in this example, if you look at the point in time that's indicated as t1, you can see that our earned value, shown by the orange line, is higher than the planned value. And that means that we are ahead of schedule with our project. Also, if you look at the purple line, our actual cost is less than our earned value. So we've spent less than the value we've created. And those are good things. If you move on to the point t2 in time you can see that the actual cost, which is the purple line, intersects with the planned value. But the earned value, represented by the orange line, is now below both the actual cost and planned value. So at this point in time, we've spent more than the value we have created. And we are also behind in terms of schedule.

Again, earned value management is a good way for us to determine whether or not we are on track, in terms of cost and schedule for our projects. One concern with earned value management calculations for Agile projects is that they require a lot of calculation and tracking in terms of cost and actual hours for our projects. So it may start to feel not so agile. However, if we're just using high-level calculations and plotting them in this kind of a graph, it's a good way to see some visibility into how we're doing in terms of cost and schedule.

Key performance indicators, or KPIs, are another way for us to measure how we're doing in terms of performance on our projects. KPIs are aggregate metrics that are tied to a strategic objective. There are two types of KPIs, leading and lagging. Leading KPIs measure activities bearing a significant effect on future performance. Two examples of leading KPIs are the product owner's time with team and the depth of ready for dev items in the backlog. Lagging KPIs measure the output of previous activity. Three examples of lagging KPIs are the estimated effort divided by actual effort, the number of defects per iteration, and the actual velocity.

In summary, earned value management, or EVM, is a project management technique that's used to measure our performance in terms of cost and schedule in our projects. The required parameters for EVM are actual project costs, an estimated backlog in story points, a release plan, and assumed velocity for the team. Key performance indicators, or KPIs, are another way for us to measure how we're doing in terms of performance on our projects. KPIs are aggregate metrics that are tied to a strategic objective.

Reviews and Feedback

[Topic title: Reviews and Feedback.] Reviews and feedback on both the project work and the product iterations as it progresses are equally important. There are four review and feedback methods that can be used in Agile projects to gain critical insight into how work and the final product are progressing. The four methods include retrospectives, Five Whys, fishbone diagram analysis, and product feedback loops.

Agile retrospectives are special meetings that take place at the end of a period of work, usually at the end of an iteration. But they can also be conducted at the end of a release. These meetings are team driven, which means that even though they may be facilitated by a Scrum Master or somebody else, the team is in large part contributing to the content of the meeting. The purpose of retrospectives is to examine the work that we've been doing together, analyze the results, and then identify ways to improve. And it's more focused on the dynamics of a team, versus the way that we've been doing specific technical tasks. Agile retrospectives have a particular framework. The first step is to set the stage and prepare the team to engage in the retrospective. The next step is to gather data to create a shared pool of data. The third step is to generate insights by observing work patterns. Following this, the team decides what to do by having a discussion about what actions to take. The final step is to close the retrospective and identify ways to make future retrospectives better.

The Five Whys is a retrospective technique that we use to get to the root cause of a specific issue or symptom. Sometimes we might want to isolate specific symptoms or issues that have happened in our iterations that may not have an obvious solution. The goal of Five Whys is to clearly understand the situation at hand. Maybe not necessarily to solve the problem, but to arrive at the real root cause of issues that are happening. The Five Whys is a powerful technique that's used outside of Agile projects as well. And it is a great way to get at the root cause of things that we may not understand at face value. One approach for conducting a Five Why session is first to display the problem statement, for example on a whiteboard. So we may say, the build broke last night. Then we'll divide the team into separate groups with no more than five people per group. Individually, each group will answer why number one happened. So the first statement is the build broke last night. Each team will come up with an answer for why did the build break? The response will be numbered as number two. We'll discuss number two, put the one that seems the most common on the board, and then individually answer why number two happened, and so on, until we get to number five. Whatever it is, at the end, we want to come up with five different answers for why, and drill down into, and try to find the root cause for why number one happened. We'll characterize the chain of responses, and then we'll discuss and consolidate the chains into a single chain that we all agree is probably the most likely.

A fishbone diagram is also known as an Ishikawa diagram, and it expands the reach of the Five Whys by allowing us to get into different categories of causes of issues that we're seeing in our projects. It helps reveal key relationships between categories of causes and the issues that we're seeing. It also provides additional insight in a very visual way. When constructing a fishbone diagram, the name of the problem or potential issue is entered at the right, and a line representing the backbone of the analysis is positioned to its left. Possible causes of the problem are drawn as bones coming off of the main backbone. Each of these will represent a particular cause category. Next possible causes are added to the main cause category bones of the diagram. The practical length is typically four or five levels. The completed diagram represents all the possibilities contributing to the potential root cause of the problem.

Continuous feedback loops are a key part of an Agile framework, such as scrum and kanban, and they're based off of the Deming Cycle, which is plan, do, check, act. The Deming Cycle was introduced by Dr. W Edwards Deming. Dr. Deming made significant contributions to quality and the total quality management, or TQM movement. Through continuous feedback loops, the Deming Cycle allows us to continuously inspect our performance, our interactions, our teamwork, our process, and our progress, and the effectiveness of all these things and how we can continuously be improving. The first stage is plan. During this stage, a plan is created with the information we have now. The next stage is do. During this stage, part of the plan is implemented or executed. During the third stage, which is check, we measure performance, effectiveness, and so on. Finally, the fourth stage is act. During this stage, we take action, change course, and adapt based on what we've learned.

In summary, there are four common methods for reviews and feedback for your Agile project, retrospectives, Five Whys, fishbone diagram analysis, and product feedback loops.

Risk Management

[Topic title: Risk Management.] All projects carry risk, regardless of size, product, or methodology. When we're talking about risks, risk refers to any uncertainty that could positively or negatively affect the outcome of a project, such as threats and opportunities. Negative risks are anti-value and minimize the value being delivered. There are a number of risks that need to be managed. These include productivity variation, scope creep, specification breakdown, personnel loss, and poor estimation.

Productivity variation is the variation between planned and actual performance. Specification breakdown refers to a lack of consensus on what should be implemented. Personnel loss refers to the unexpected loss of human resources. Risk assessment evaluates the probability of a risk occurring and the impact that it may have. Risk severity is calculated by multiplying the risk probability by the risk impact. There are several risk categories. Business risks threaten the business value of the project. Technical risks relate to the technologies being used and the technical implementation of the project. Logistics risks relate to funding, scheduling, and so on. Finally, there are other types of risks, such as political risks.

Inactivity to run with your team on an agile project in order to ensure a higher possibility of success and reduce risk is the pre-mortem. Pre-mortem activities are performed prior to release. And help prepare for the upcoming release or challenge. The pre-mortem approach is to simulate an already failed case and then identify all potential reasons for why the failure occurred. The reasons for the failure are then eliminated.

The first step in a pre-mortem is to brainstorm and list all potential problems. Next, document problems that even have a remote chance of occurring. No potential problems should be left off the list. Discussing or detailing possible solutions should be avoided at this point, as it draws attention from identifying problems.

The second step in a pre-mortem is to identify the top problems. During this step, select between five and ten problems to focus on going forward. The focus should be on problems that are critical to the project. Minor issues should be avoided. Identify issues that will likely happen and tackle problems that are inevitable. Discard problems that you cannot control, such as external risks. Instead, focus on problems that can be corrected.

The third step in a pre-mortem is to create solutions. You should spend approximately one hour reviewing the top problems and then creating a list of actions. The actions should create proactive solutions to existing problems and define a backup plan. Backup plans are ideal for problems that haven't happened yet.

Variance is defined as the difference between actual results and the expected results. The process of analyzing these results is called variance analysis. And it's another risk management strategy useful in agile projects. A variance analysis can yield two different types of analyses. These are a favorable variance and an adverse variance.

A favorable variance is when the actual results are better than the expected results. An adverse variance is when the actual results are worse than the expected results. In order to determine the significance of the results of a variance analysis, identify the standard being used. And determine the level of interdependence between variances. This analysis helps maintain control. For example, it may help maintain control of a project budget. It also helps spot trends, identify opportunities, and identify threats. There are a wide range of values that can be used for variance analysis. These include purchase price, selling price, labor efficiency, variable overhead efficiency, material yield, fixed overhead spending, and labor rate.

Introspectives are another risk management activity you can use in your agile project. An introspective is a planned event and isn't regularly recurring like retrospectives. They take a system's level view at the organization and its processes. Introspectives help sustain organizational agility by assessing the effectiveness of its processes, functions, tools, rules, and technology.

The first step in the introspective process is to identify the survey participants. Typically, it's the consumers of the product that will be the primary source of information. The next step is to send the survey to the identified

participants. Following this, the survey results are summarized and distributed. This is a preparation step for the introspective where the participants of the introspective session are identified. These are usually the consumers of the product or service, and the stakeholders. Finally, the introspective session is run. This involves a discussion related to the findings, and coming up with an implementation plan. An introspective session requires a trained facilitator, and a note taker. These sessions can take up to three hours. Upon completion, the prescribed changes are approved by stakeholders, and implemented to return immediate results. Any successes resulting from the session are broadcast throughout the organization.

Value stream mapping is a lean process tool used to analyze the flow of information and activities from inception through to the delivery of value. It describes the flow of ideas and actions and helps uncover bottlenecks, ques, and silos. The value stream mapping process starts with identifying the criteria to be mapped. These could be activities, symptoms, or products. The next step is to map out the current state. It's then possible to identify where there are areas of waste. At this point, a future state map is created. Finally, the future state map is implemented. Value stream mapping has a number of benefits. It provides a clearer picture of processes. And its focus on time allows time-based decisions to be made over those that are cost-based. It also identifies areas of waste and provides a common discussion point. In addition, value stream mapping identifies required changes, improves efficiency, and assists with standardization. Some disadvantages of value stream mapping are that there are no immediate significant monetary measures for value. And that stakeholders are responsible for differentiating between value and waste.

In summary, pre-mortem activities, variance and trend analysis, introspectives, and value stream mapping are common risk management strategies you can use in your agile project to identify and mitigate risk.

Product Quality, Testing, and Integration

[Topic title: Product Quality, Testing, and Integration.] Agile products are only considered successful if they have the expected level of quality that was planned. Agile quality processes and practices introduce quality into Agile projects. Quality processes and practices include quality verification, validation, and continuous integration.

Verification is used to determine if the right product is being built. It involves compliance and requirements checks, and ensures the proper use of data. Verification is a high-level activity. And includes things like integration checks, code reviews, and peer reviews. Its main purpose is to determine if a product meets customer needs.

Validation is used to determine if a product is being built right. It's used to review consistency, standards, and techniques. Validation is a low-level activity consisting of exercises such as walkthroughs, feedback, inspections, unit testing, and functional testing. Validation demonstrates the level of consistency, correctness, and completeness.

And the objectives of continuous integration are to minimize the effort and overall duration required for each integration. And to deliver product versions suitable for release, continuous integration uses version control tools, such as the Concurrent Versions System, or CVS. It provides an automated build and product release process. And requires a number of unit and acceptance tests.

Agile testing is a software testing practice with a number of advantages, including that it saves us time and money, requires less documentation than traditional testing techniques, helps us gather regular feedback, and identify issues in advance. The principles of Agile testing are that testing is performed continuously throughout the project. Testing moves the project forward. So you want to view testing and perceive testing as an activity that helps us move forward, rather than holds us back. All teams should always be required to test. No teams are exempt from testing.

Agile testing also helps support gathering continuous feedback. Coding issues are regularly fixed within Agile testing frameworks. We're producing less documentation. But we're also testing throughout the implementation

of our code.

Agile testing is an exploratory, scripted approach, with a rapid development pace. Its emphasis is on autonomy, skill, and creativity. Agile testing is complementary to test automation, and helps identify possible issues with the product. Agile testing has a usability focus. It's an exploratory technique in which the best results come from understanding the end user and their work flow. It involves observing end user interactions without intervening. The user's actions are documented by taking notes, video recording, or using screen capture. The difficulties encountered by users are then examined during a post analysis.

In continuous integration, developers integrate code several times a day, using a shared repository. The objectives of continuous integration are to identify problems early to minimize effort and duration. And to deliver a product suitable for release at any given moment. Continuous integration involves a number of practices. These include maintaining a single source repository, automating the build, and making builds self-testing. In addition, everyone commits to the mainline, every day. And every commit should build the mainline on an integration machine. Other continuous integration practices include keeping the build fast. Testing in a clone of the production environment. And making it easy for anyone to get the latest executable. Another practice is making sure that everyone can see what's happening. Finally, automating deployment.

In summary, quality processes help introduce quality into Agile projects. And Agile testing helps insure quality in the project and has a number of characteristics. Agile testing characteristics include that it is performed continuously, is mandatory, exploratory, and scripted. It also emphasizes autonomy, skill, and creativity, and is focused on usability. The practice of continuous integration also contributes to project quality.