Crosswind Learning

Version 6.1 Updated for the 2021
Project Management Professional (PMP)® Exam

Crosswind Success Series: PMP® Exam Bootcamp Manual

www.crosswindpm.com

Tony Johnson, MBA, CAPM, PMP, PgMP, PfMP

Version 6.1 aligned with the Project Management Institute, *A Guide to the Project Management Body of Knowledge, (PMBOK® Guide)* - Sixth Edition, Project Management Institute Inc., 2017

8.5.15. Planning Poker

Planning Poker is an estimation technique that is effective because it relies on multiple expert opinions. Each estimator (developers, programmers, testers, database engineers, analysts, designers, etc.) is given a deck of cards. The sequence of numbers that planning poker is based on came from the Fibonacci set (each number is the sum of the previous two numbers). Each card in the deck contains one valid estimate. The moderator, usually the product owner/customer or an analyst, presents a user story or theme and answers any questions the estimators may have. Each estimator selects a card and turns it over so that everyone can see the card. If the estimates vary, the estimators explain their selections and another selection takes place. This continues until the estimates converge. Planning Poker should be played before the project begins and as new stories are introduced.

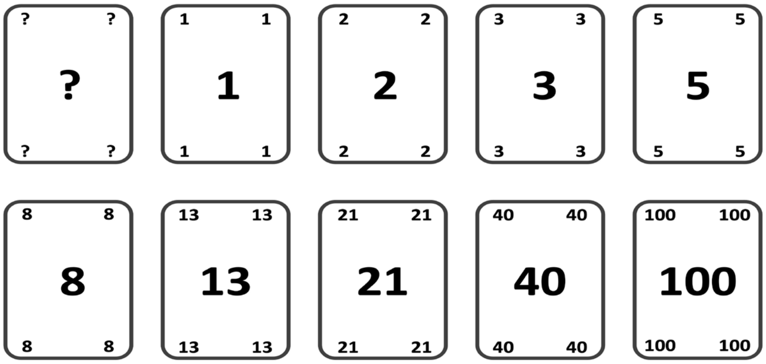


Figure 8-8: Planning Poker

8.5.16. Agile Modeling

Agile modeling is a set of practices and principles that will be applied to the modeling and analysis of requirements.

Agile modeling espouses the use of barely sufficient modeling: model only enough to proceed with the work. It considers modeling a form of communication.

A typical modeling session might include a diagram drawn on a white board while the team discusses the concepts the diagram is capturing. In areas where the team thoroughly understands the concepts, the diagram is executed at a very high level; in areas where the team's understanding of the concepts is not as thorough, the diagram is executed with more detail. Pictures are taken of the diagram, distributed to the team as a reminder of the discussion, and the white board is erased.

8.5.17. Product Roadmap

The purpose of the product roadmap is to depict the evolution of the product over the next three or four releases or time periods. The roadmap depicts the features or themes that will be delivered during each release or time period and designates the targeted product owner/customer, supporting architecture for the feature, and the anticipated business value for the release or time period.

The product owner/customer, who owns the roadmap, meets with the Agile project manager, the architect, and executive management to develop or revise the roadmap at specified intervals.

The newly developed or revised roadmap is presented to the entire team and to the stakeholders in a meeting, typically a release planning meeting, that encourages feedback.

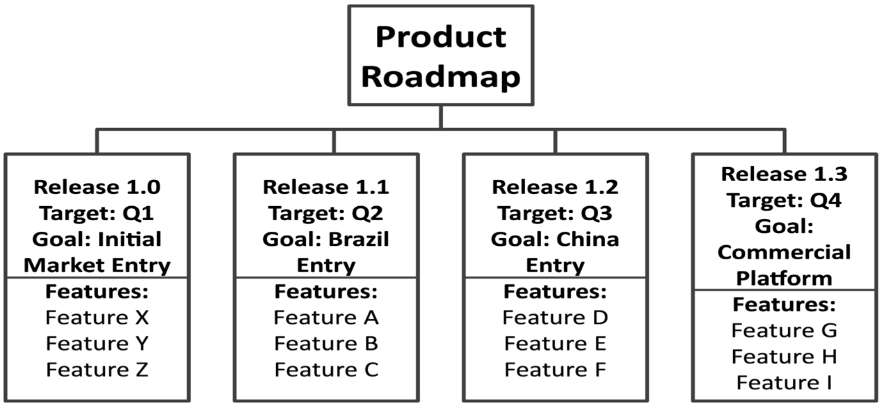


Figure 8-9: Product Roadmap

8.5.18. User Story/Backlog

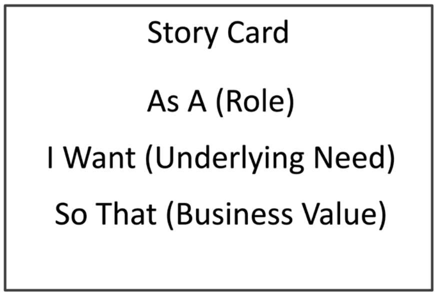
A user story, which is written in business language, is a short description of a piece of functionality the user wants the system to include. Although there is no mandatory format, a user story typically lists the role that needs the functionality with a brief description of the functionality and the reason the functionality is needed: "As a salesperson, I want to be able to sort my contacts by last name, first name, affiliated company, or last contact date so that I can easily access information for a specific contact."

A mnemonic to remember the characteristics of a good user story is **INVEST**:

- **I**ndependent (self-contained and not dependent on another user story)
- **N**egotiable (can be changed until in an iteration)
- **V**aluable
- **E**stimable
- **S**mall
- **T**estable

The user story is written on a card or, for large or distributed teams, typed into a computer application. **It is the basis for a dialogue between the development team and the users** (conversation and confirmation).

The story card, conversation, and confirmation are the Agile alternative to detailed requirements and is included in the acceptance criteria.



Story Card

As A (Role)

I Want (Underlying Need)

So That (Business Value)

Figure 8-10: Story Card

The user story backlog is comprised of a list of user stories that have not been completed and are organized by priority (high to low).

8.5.19. Personas

To make a user story vibrant and more meaningful, associate it with a persona as the primary user of the features represented by the story.

A persona is an archetype (a generic representation of a user personality). Before creating a persona, ensure that the person selected is representative of the target audience: rely on solid market and demographic research to determine this.

Once a person is selected, write the persona definition on a piece of paper. Since some background information will be included, as well as history of employment with the company and a description of the work the person does, a note card will not be large enough. Adding a picture of the person to the persona paper is an excellent practice.

After determining applicable user roles and a persona or two, write the user stories substituting “the user” with the name of the applicable persona or role.

8.5.20. Definition of Done (Acceptance Criteria)

The definition of done in relation to a user story is that the functionality expressed in the story is ready to deploy.

Many teams use a checklist to ensure that the definition of done is understood by everyone on the team.

A sample checklist might be:

- All unit, integration, and product owner/customer tests have been completed
- The code has been written and all tests have been passed
- The team has refactored the code: refactoring involves reviewing the structure of the code and making changes necessary to improve its quality
- The code has been integrated into the software
- The build scripts have been amended to include any new modules
- The story expressed by the code is included in the automated installer
- The code has been migrated and database schema has been updated as appropriate
- The product owner/customer has reviewed the functionality expressed by the story and has confirmed that it met expectations
- Each error has been fixed or scheduled as a separate story
- The product owner/customer has accepted the code

8.5.21. Continuous Integration

Traditional software development integrates code infrequently, often resulting in integration issues that are discovered so late in the project cycle that their source is difficult to find and correct. Late discovery often leads to increased costs required to meet the delivery date and/or a late delivery date.

Agile methods espouse frequent code integration (every few hours is recommended) so that integration issues may be found and corrected quickly.

Frequent integration is more efficient than infrequent integration. Locating issues is a much simpler process and a coder is not as likely to step on another programmer's code.

Integration is a rehearsal for a release, that is, all the steps involved in deploying a release are practiced during deployment of an iteration so everyone is fully practiced by the time the release is ready to deploy.

To effectively integrate code into the software, the following practices are recommended:

- The coder uses a source code management system, which keeps the source code for the project in a single repository, to check out the current copy of the integrated source code to his machine
- The coder alters the source code to incorporate his code and adds or changes automated tests
- The coder carries out an automated build on his machine, which compiles the code into an executable and tests it
- If the test is successful, the coder must check to ensure that the source code used has not been altered; if it has, then the coder's working copy must be updated with those alterations and a new build made; any issues that are detected in that build must be fixed
- The coder can then commit the successfully tested build to an integration machine (which contains the current copy of the integration source code) and do a build on that machine; if there are no errors, the build is done

To effectively apply continuous integration, the following practices are recommended:

- Use a single source repository
- Make builds automatic and self-testing
- Require all coders to commit to the build on an integration machine everyday
- Test the build in a production software copy
- Make the most recent executable readily available to everyone
- Make it easy for everyone to know the current state of the system and what changes have been made
- Automate deployment of the executable

8.5.22. Frequent Verification and Validation

In a traditional project, verification is the act of determining that the project result fulfills all requirements and validation. In Agile project management, verification is the act of determining that the project meets the needs of the product owner/customer.

In Agile project management, both verification and validation occur frequently and at various levels: verification occurs at the task level, the user story level, the iteration level, the release level, and the completed product level; validation occurs at the user story level, the iteration level, the release level, and the completed product level. This approach ensures that variances in expected results are found and corrected early on.

Verification is determined primarily through automated testing and validation is determined primarily through user acceptance testing.

8.5.23. Emotional Intelligence

Emotional intelligence is the ability to achieve goals through self-management and the influence of others. The underlying character traits are self-awareness, self-discipline, external awareness (influence and empathy) that extends to persons and situations, and ethics. If devoid of ethics, the influence aspect of emotional intelligence is manipulation.

Emotional intelligence is a key leadership trait and is vital to conflict resolution, building and maintaining a team, and making effective decisions.

8.5.24. Adaptive Leadership

Adaptive leadership is the ability to alter one's leadership style to fit the situation or stage of team development (for instance, the project manager may need to direct activities when the team is first formed, but will be totally disengaged from the decision-making process when the team has matured and is performing autonomously). It is a key skill needed to ensure that the project is aligned to the business goals of increased profitability and market share. The underlying character traits of adaptive leadership are adaptability, creativity, and analytical thinking.

To ensure profitability, quality must be successfully managed and the simplest things possible to achieve customer satisfaction must be done, for instance eliminating features that increase cost without adding value. Teams that are self-directed, skilled, and invested with authority, should be encouraged to carefully analyze metrics, even at the lowest levels, to address waste.

To ensure market share, market changes should be addressed quickly to maximize opportunity and minimize constraints that negatively impact the project.

8.5.25. Servant Leadership

Servant leadership is the ability to lead by serving. It is a skill that relies on the underlying character traits of humility and business ethics.

A project manager practicing servant leadership should be willing to, and capable of, removing obstacles and providing resources. Sometimes referred to as "moving rocks and carrying water," this skill is often critical to the success of the project.

An example of the project manager moving rocks (removing obstacles) is coaching an aggressive team member to respect other members of the team. An example of the project manager carrying water is convincing a production manager to free up time for an employee to dedicate more time to his role as a key subject matter expert.

8.5.26. Minimally Marketable Feature (MMF) Prioritization

Minimally marketable feature (MMF) prioritization establishes priority based on the smallest increment of functionality that has value for the users.

8.5.27. Relative Prioritization/Ranking

Relative prioritization/ranking measures the value of functionality relative to the value of the other functionalities and establishes priority based on value.

8.5.28. Minimally Viable Product (MVP) Prioritization

Minimally viable product (MVP) prioritization establishes priority based on the version of the product that has the least amount of features that can be released, yet:

- Has sufficient value for people to use the product or for consumers to purchase the product
- Demonstrates enough future benefit to retain early users
- Provides early user feedback to guide future development

The features in the product are prioritized to maximize business value.

8.5.29. MoSCoW Prioritization

The MoSCoW method of prioritization requires the product owner/customer to designate each feature as a **M**ust have, **S**hould have, **C**ould have, or **W**ould have.

M -	**M**ust have The requirement is critical to the success of the current delivery timebox (fixed time period allocated to a delivery) and must be delivered
S -	**S**hould have The requirement is important, but since it is not critical to the success of the current delivery timebox, it is typically delivered at a later time
C -	**C**ould have The requirement is desirable, but since it is not necessary, it is typically only included if time and resources are available
W -	**W**ould have (Nice to have) The requirement is not considered critical to the success of the project (because it is among the least-critical or lowest return project requirements) or is inappropriate for the current delivery timebox: it is typically either postponed until a later delivery timebox or removed from the project altogether

8.5.30. Kano Prioritization

The Kano prioritization method is based on the Kano Model of Customer Satisfaction, developed by Professor Noriaki Kano. It requires the product owner/customer to designate each feature as a **threshold**, **linear**, or **exciter and delighter**.

A **threshold** feature is required for the product to be successful. It meets a basic need and does not impact customer satisfaction. First priority should be given to threshold features. Often, it is acceptable to only partially implement threshold features, because increase in customer satisfaction drops after a base level of the threshold feature is developed.

A **linear** feature is a feature whose quality directly impacts customer satisfaction. Providing more of the feature or improving its performance increases customer satisfaction somewhat. Secondary priority should be given to linear features. Typically, unless the project is suffering from feature bloat, as many linear features as possible should be included.

An **exciter and delighter** feature is a feature that does exactly as described and increases customer satisfaction dramatically. Tertiary priority should be given to exciter and delighter features, but if there is any time available, including a few is an excellent idea. Kano features can migrate over time. What was an exciter and delighter feature can evolve to become a linear feature, and then a threshold feature.

8.5.31. Velocity/Throughput/Productivity

Velocity is the measure of story points that can be completed during an iteration. A story point is a unit of measurement that represents the size of a user story relative to the other user stories in the project. For example, a three-point user story is three times larger than a one-point user story.

To estimate velocity for the initial iteration, use traditional estimating techniques. To estimate velocity for the next iterations, simply total the story points that were "done" in the previous iteration. Reliable velocity is predicated on a strict timebox for the iteration and a strict definition of "done." You cannot allow the deadline for the iteration to slip. You must define done as "ready to deploy."

Slack is also a consideration when planning an iteration. For the last two days of an iteration, you might want to schedule work that is useful, but not critical time-wise. That way, if more time is needed to complete the stories, the scheduled work can be set aside.

Creation of a velocity chart, which charts the story points in all previous iterations, provides an easy reference to determine the project's progress and status. As the project progresses, the velocity of the iterations tend to become more stable provided the iteration lengths do not vary and the team members remain the same. The velocity chart can even be used for future release planning provided future changes in teams and priorities and goals are considered.

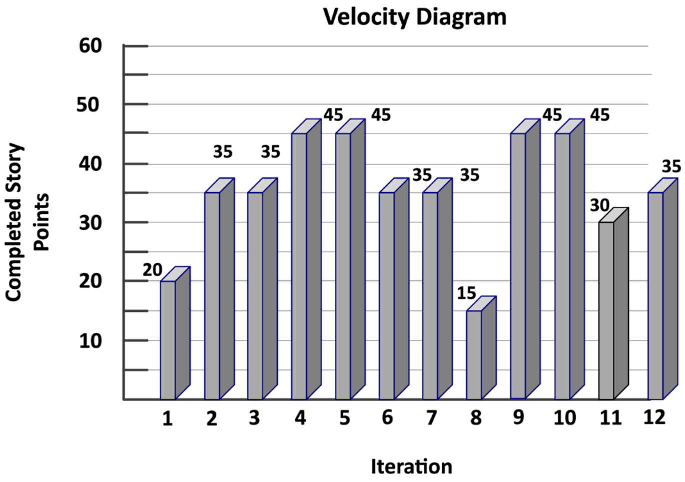


Figure 8-11: Velocity Chart

Throughput is the number of user stories that can be done in a given period of time. Since throughput relies on actual start and finish times, the team can derive a predictive measure for forecasting without any estimating. Note: if user stories are not of a similar size, the standard deviation will be larger. This will result in a larger range for your completion date predictions.

Productivity is measured as output per unit of input: Productivity = Output / Input. Many Agile experts state that measuring acceleration, the increase in productivity over time, is more useful than measuring productivity.

8.5.32. Value Stream Mapping

Value stream analysis is a technique for determining that a project represents the best use of the enterprise's time and resources. The value stream is the stream of activities that occur between a product owner/customer request and delivery of value. Both value added and non-value added activities are included. To calculate total cycle time, add total value-added time and total non-value added time. To calculate process efficiency, divide value added time by total cycle time; the higher the percentage, the greater the efficiency.

The primary tool for analyzing the value stream is value stream mapping. It was initially developed for manufacturing to determine bottlenecks in a process, but has been adapted by Lean to use with an Agile software project.

In an Agile project, the focus is on outcomes rather than processes. Basically, it involves creating graphic representations of the process streams required to complete a process, and then analyzing the streams to determine if there are any problems.

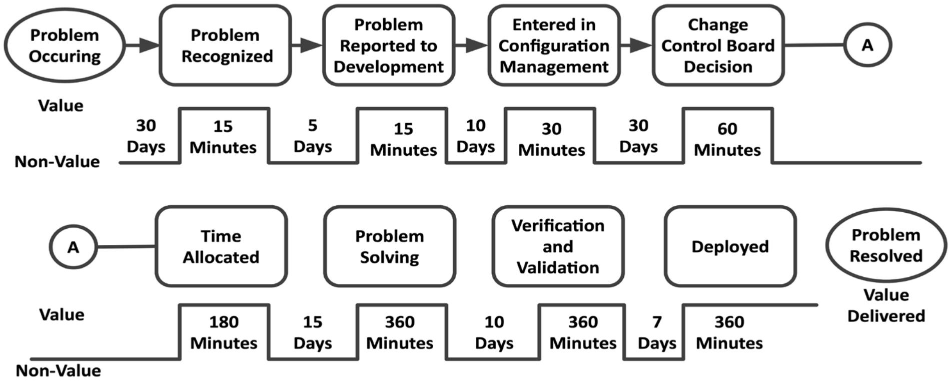


Figure 8-12: Value Stream Mapping

Once a problem is located, the team is questioned until the root cause of the problem can be determined. In Lean, this method is called the Five WHYs, although the actual number of WHYs is dependent upon the complexity of establishing the root cause of the problem. The initial WHY is always "Why has this problem occurred?" Once that answer is given, the team is asked, "Why did that happen?" That question continues to be asked until the root cause is determined.

8.5.33. Retrospectives

A retrospective is a meeting where individual members of the team critically evaluate the team in terms of performance and behavior. Performance and behavior are rated as: above standard, at standard, or below standard.

In an Agile project, to insure that the retrospective is valuable, it is typically held at the end of an iteration rather than at the end of the project.

The performance evaluation typically considers what went well, what didn't go well, and what can be done to improve performance. The behavior evaluation typically considers the discussion participation, meeting attendance, and accountability of individual team members; the management style of the project manager; and the team's understanding of the basis for key decisions made during the iteration.

8.6. Traditional Projects

The traditional approach to projects uses a set of developed techniques for planning, estimating, and controlling activities in order to reach the desired result on time, within budget, and in accordance with specifications.

8.7. Hybrid Projects

The hybrid approach to projects uses elements from the Agile approach and the traditional approach. This is done to accommodate a specific project or situation.

8.8. Agile and Hybrid Management Formulas and Variables

There are no formulas for this chapter. See the Cost chapter for earned value management formulas.