Domain I. Agile Principles and Mindset (9 tasks)

Here are my PMI-ACP notes in PDF form, taken from my PMI-ACP review.

Explore, embrace, and apply agile principles and mindset within the context of the project team and organization. I associated everything in domain 1 with all aspects, values, characteristics, and concepts of agile and all of its supporting or related methodologies.

Some key terms and concepts you want to keep in mind are:

Agile mindset

Defined processes vs. empirical processes: In empirical process control, you expect the unexpected. With defined process control, every piece of work is understood. In agile, an empirical process is implemented where progress is based on observation and experimentation instead of detailed, upfront planning like in defined processes.

Being agile vs. doing agile

Concept of ruthless prioritization, team collaboration, and servant leadership (following agile virtues) vs. just going through the relevant concepts of scrum (e.g., having defined roles, rituals, and artifacts). For example, you can follow scrum rituals like daily scrums (standup), sprint planning, sprint reviews, and retrospectives and have a proper sprint backlog. But if you're not prioritizing value for customers / management and are micromanaging your team instead of empowering them – you are not considered agile.

At the end of the day, you want to have the mindset, not just the rituals and artifacts. Really absorb the values and principles to unlock true agile.

Doing agile vs. being agile

The agile triangle

Comparing the traditional iron triangle to agile iron triangle of constraints. Where cost and time is considered fixed in agile, but scope is a variable (due to timeboxes and relevant artifacts). You can build on this further with the true agile triangle being value, quality, and constraints (with constraints encapsulating the agile iron triangle).

Agile inverted triangle vs. predictive and the offshoot of it

The four main principles of agile (agile manifesto)

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

Exam tip: Make sure you memorize these four main principles. They're super important and are at the core of every agile concept and each methodology. Also keep in mind that the manifesto is not saying to DISREGARD all documentation or contracts, but that working software and customer collaboration are more valuable. So think of it as a prioritization of one principle over another, not a complete disregard for one. There will be cases where you'll need documentation in agile (compliance, high technical infrastructure, or it's a requirement in the project).

The 12 agile principles

Here's my take on the 12 principles of agile.

ShortenedversionFull principle Highest priority = deliver value to customerOur highest priority is to satisfy the customer through early and continuous delivery of valuable software.Welcome changes, even late to harness competitive advantageWelcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

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

Business people and developers must engage dailyBusiness people and developers must work together daily throughout the project.

Build projects around a motivated, self-governing teamBuild projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Face-to-face (F2F) = best form of commsThe most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progressWorking software is the primary measure of progress.Sustainable development should be possible indefinitelyAgile

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

Continuous attention to technical excellenceContinuous attention to technical excellence and good design enhances agility.

SimplicitySimplicity – the art of maximizing the amount of work not done – is essential.

Best designs and architecture are created by self-organizing teamsThe best architectures, requirements, and designs emerge from self-organizing teams.

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

Exam tip: Honestly, a lot of practice exams tell you to memorize these, and while I did that, I feel like understanding them is more important. These 12 agile principles are the core foundation of most major agile methodologies (XP, KanBan, scrum, and even FDD), however they are based off of the four agile values stated earlier.

Agile methodologies

There are roughly 5 to 7 methodologies you should know "of" and 3 to 4 methodologies you should have a practical understanding of.

The "must-haves" include:

Scrum

Values (openness, commitment, respect, courage, focus), pillars (transparency, adaptation, and inspection), rituals (daily scrum (standup), sprint planning, sprint review (demo) and retrospective), roles and duties of these roles (scrum master, product owner, and team). Scrum **main** artifacts (sprint backlog, product backlog, potentially shippable increment), general concepts of sprints and timeboxes, and how they work.

Exam tip: Scrum was by far the most covered methodology in my experience. I would recommend knowing all of these concepts above, and be able to apply hypotheticals to them easily. For example, knowing that the product owner can cancel a sprint is extremely important, but knowing when and how to apply that concept to a question like "can a product owner (P/O) cancel a sprint" and answer "yes, a P/O can cancel a sprint especially if the sprint has lost value" is key. On my exam, there were a ton of hypotheticals similar to this, where you needed to decide what to do in a specific circumstance (situational).

KanBan

KanBan principles (visualize workflow, limit WIP, manage workflow, make process policies explicit, feedback loops, and improve collaboratively). KanBan boards and how they are a “pull” method vs. a push, KanBan and WIP limits and how WIP limit reduces WIP and therefore can increase throughput and overall velocity (two important terms), benefits of KanBan, and process tailoring with KanBan (how it works with other methodologies).

eXtreme programming

Five (5) principles of XP (communication, simplicity, feedback, courage, and respect), 13 practices of extreme programming and how they work (planning game (iteration and release planning), small releases, simple design, customer tests, collective ownership, metaphors, sustainable development, coding standards, paired programming, continuous integration, refactoring, and test driven development).

Some other aspects of XP that are important are: 10-minute builds (related to continuous integration (CI) and test driven development (TDD)), Slack, weekly and quarterly cycles, and incremental designing.

I would also have a brief understanding of roles in XP (customer, tracker, coach, team (developer and testers), their specific goals, and how they work. They are very similar to scrum, but differ in their paired programming aspects.

Exam tip: Roles in XP and scrum are very similar, which is why the synergy between the two is so high (scrum is more about time boxing and improving process, whereas XP is about efficacy of development standards). Also keep in mind the XP’s development cycle (I think this is the only reason I never got above target on this domain). Having an understanding of the concepts of planning, analyzing, designing, coding, testing and deploying cycle of XP is key as, at a high level, XP is similar to scrum but different in its scope.

Lean

Lean is a pretty straight forward concept. You’ll want to know the steps to lean (identify value, map the value stream, create flow, establish flow, and continuous improvement), the 7 principles of lean (eliminate waste, build in quality, create knowledge, defer commitment (last responsible moment), deliver incrementally, respect people, and optimize the whole). You’ll also want to know the 7 types of waste under lean as well.

Exam tip: Lean is at the basis of all agile concepts and so is reducing waste (as waste is an anti-value and the purpose of agile is to drive value). Therefore, you might not need to know EVERY single core value and principle, but I would memorize the wastes of lean as neutralizing waste is extremely important in agile.

Some more obscure and less important Agile methodologies that might have one or two questions on the exam are:

FDD (Feature Driven Development)

I knew the basics of how FDD worked (like the concept of the domain, and the five steps to FDD (develop an overall model, build a features list, plan by feature, design by feature, and build by feature)). Plus, the roles of FDD like the engineer lead and domain / class architects.

DSDM (Dynamic Systems Development Model)

Generally understand the concept of DSDM (its birth with a combination of rapid application development (RAD) and waterfall), and some of the roles associated with DSDM (there's a lot).

Crystal methodology

Just know how crystal works and how criticality and team size affects the color of the specific crystal methodology. The higher the criticality (chance for injury or level of importance to the customer's life or wellbeing) and the larger the team size, the darker and more opaque the crystal color.

Agile leadership

In my notes, I also had a short section on agile leadership vs. management and how agile is more about empowerment and enabling your team to do the right things (in addition to providing them the right direction vs. a traditional management style of command, control, and efficiency). While neither of these are wrong, it's important to note that this is an "agile" exam, so you never want to "direct" or "control" team members to do anything. It matters the type of project and execution you're building; traditional project management has its positives as does agile. Collaboration and empowerment is key in agile leadership.

Overall, this was my weakest domain (was on target) so I could have probably studied more for these questions. However, I would advise you not to memorize everything and grasp the four core agile values, 12 agile principles, and the roles and workflows for

each agile methodology I described. Understanding the soft skills of agile leadership and being agile is also very important.

Domain II. Value-Driven Delivery (4 subdomains, 14 tasks)

Deliver valuable results by producing high-value increments for review, early and often, based on stakeholder priorities. Have the stakeholders provide feedback on these increments, and use this feedback to prioritize and improve future increments.

What is value-driven delivery for the PMI-ACP?

Basically, projects are undertaken to do two things: Reduce risk (anti-value) and generate revenue. Lean and KanBan do a great job of this. This aligns with the first principle of agile (always try to drive value for the customer).

Concept of agile project accounting, risk management and compliance

The concept of agile project accounting is in incrementally providing value to the customer (in time-boxed sessions). You end up providing revenue throughout the project instead of at the end (thus mitigating risk and increasing cash flow for the customer).

This is an important concept for agile as it helps mitigate risk (since you can adapt, reprioritize your plan, and pivot to different things). Whereas with traditional project management, you could be stuck with a set of products or features that are now obsolete (due to the project being delivered all at once).

Note: A word about compliance projects in agile. As most projects revolve around either generating value or reducing risk, projects or features related to compliance are usually related to safety or security regulations instituted by a government regulation or an association. Compliances issues I've tackled and had to prioritize within my agile sprints include those surrounding the General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA). While meeting compliance may not generate any value per say, it can reduce your liability and exposure to prosecution. Therefore, it might be the case that you just need to ***label*** *the project a* ***must-do*** *or* ***must-have****. This being said, always try to earmark time in your cycles to ensure that you have room to prioritize tasks*

for compliance (set aside time when estimating your user stories or tickets with the development team for compliance issues).

Earned Value Management (EVM) for agile projects

Earned Value Management (EVM) is a broad and important concept in traditional project management that has been downsized and adapted for use in agile projects.

What EVM means to me is a number of formulas that allow you to calculate the feasibility and success of a project. While I was never asked to calculate the metrics on EVM on my exam, I did have a few questions related to knowing *how to work with* the following metrics:

ConceptFormulaResult Interpretation Internal Rate of Return (IRR)n/aSelect project with biggest IRRNet Present Value (NPV)NPV = Sum of Cash Flow / (1+i)^tSelect project with biggest NPVReturn on Investment (ROI)ROI = (Return – Cost – Investment) / InvestmentSelect project with biggest ROISchedule Variance (SV)SV = EV-PV< 0 is bad = 0 is good > 0 is goodSchedule Performance Index (SPI)SPI = EV / PV< 1 is bad = 1 is good > 1 is goodCost Variance (CV)CV = EV-AC< 0 is bad = 0 is good > 0 is goodCost Performance Index (CPI)CPI = EV / AC< 1 is bad = 1 is good > 1 is goodEVM equations and table

Here are detailed definitions for the equations:

- **Earned Value (EV):** The portion of the approved budget (or story points) that are allocated to a completed item.
- **Planned Value (PV):** The (budget / story point) value of items that were planned to be completed at a point in time.
- **Actual Cost (AC):** The amount of cost consumed at a point in time.
- **Cost Variance (CV):** The difference between the earned value and the actual cost at a point in time.
- **Cost Performance Index (CPI):** The earned value divided by the actual cost. A value of 1 indicates that the earned value is in line with the planned cost. A value greater than 1 indicates that the project operates at lower cost than initially planned, while values smaller than 1 show the opposite.
- **Schedule Variance (SV):** The difference between earned value and planned value.
- **Schedule Performance Index (SPI):** The schedule performance index is a relative expression of the schedule variance. A value of 1 shows that the project operates in line with the planned schedule. If the SPI exceeds 1, the project is ahead of the plan, while a value below 1 indicates that it is behind the schedule.

- **Value-Based Prioritization:** The concept of prioritizing based off of three factors: 1) benefit; 2) risk; and 3) dependencies.
- **Return on Investment (ROI):** This return on investment sets the returns of an investment in relation to the investment amount.
- **Net Present Value (NPV):** The sum of discounted net cash flows.
- **Internal Rate of Return (IRR):** The discount rate for which the present value (sum of discounted cash flows) equals 0. It is usually iteratively determined and there is no simple formula. Thus, the exam will focus on the understanding of the concept rather than require the IRR calculation.

The purpose of all of these is to understand if things are working out, and be able to surface to stakeholders, sponsors, and those related to the project how well the project is going. This being said, EVM has issues in that it doesn't fit the **best** with agile and progressive elaboration (explained further below). EVM requires a lot of data (costs, forecasted revenue, a dedicated team, salaries etc.), which you regularly don't have as an agile project or product manager.

Value prioritization

This is a core concept in agile and the PMI-ACP that you should understand fully. In a nutshell, customer value prioritization in agile is the concept of delivering the highest value possible, as early as possible in a project to ensure early cash flows to the customer and also mitigate risk from an ever-changing landscape. The product backlog is where most of this takes place, and therefore is a very important part of any agile team.

Here are some value prioritization schemes you may see on the exam:

Simple schemes: Rank from high to low (priority 1, 2, 3, etc.), based off a business significance and revenue / cost.

MoSCoW prioritization scheme: "Must have", "should have", "could have", "would like to have". Created for Dynamic Systems Development Method (DSDM) but s very popular way to prioritize features and stories. I love MoSCoW and I've actually used it in my roles as a project and product manager.

Kano analysis: Categorizing features and components into ways that affect the users that use them. There are a number of concepts to remember for Kano analysis:

- Delighters / Exciters: These are the features that people love, but didn't know that they would. A real world example of this is people not noticing Google Meetings'

new noise-cancelling feature within their software, which blocks out specific decibels and sounds (introduced in 2021).
- Satisfiers: The more of these things, the better – like chicken wings .
- Dissatisfiers: These are the things that will cause the user to dislike the product if they are not associated with the product itself, but will not necessarily raise satisfaction if they were. For example, the ability to close the carton top of a milk carton (leaving it exposed or closed). ***Note:*** *Having dissatisfiers are must-haves, and it took me a while to realize this.*

100 Points Method: Fairly simple concept, where everyone gets 100 points and they can vote on features using these points. The stakeholders can distribute their points as they see fit (100 points on one feature or 5 points spread across 20 features). Not to be confused with dot-voting or multi-voting.

Monopoly money: Basically, stakeholders will get a certain amount of money that represents the project budget, and they can **buy** features they think are the most important.

However, there are two issues with this:

1. If documentation is required, usually no one actually buys it.
2. Everyone must have a good understanding of what each feature does, so this can be difficult if the project is highly technical or specialized.

Karl Weber's prioritization model: A very interesting model, where the business team rates the benefit and penalty of not having a product or feature, and the development team rates the cost and risk of building the product. Based off of these calculations and others (in a matrix), you will use statistical weights to calculate the best feature / product to build. I've actually tried this model with a varying degree of success, and while I admit it's a tad confusing, it did help me organize the features I needed for the project I was building. Also, it allows for an extremely high level of collaboration between the delivery team and business stakeholders.

Concept of delivering incrementally: Another super important concept to grasp for agile. The purpose of incremental delivery is to ensure that there's regular deployments or builds towards a staging environment, so that there's an evaluation of and **validation** of a customer's requirements. There are a number of positives of incremental delivery, but one of the main ones is the "cost of change." One main issue with Waterfall development (explained earlier) is if there's an issue that's deployed at the end of the project, it could affect the whole program or product. Whereas with incrementally

delivery, you're releasing code or pieces of your product in small chunks and validating / verifying as you go.

Cost of change increases as development continues

Minimal Viable Product (MVP) and Minimum Marketable Feature (MMF): MMF can be MVP, but MVP might not be MMF. My definition of MVP is as follows: The minimum amount of work required in a product to enter it into the market, while providing value to the customer. MVPs are great for proof-of-concepts and is important for the "failing fast" mentality in agile.

In my mind, MMFs are a subset of MVPs, but can encompass the implementation of a feature onto a product as a means to gauge its adoption by users. A more modern (circa 2020) example of an MMF reminds me of LinkedIn's new "pronoun feature," where you can add "his / him / he" or "she / her" into your name. It's appropriate, since gender sensitivity is at an all-time high as I write this guide.

Some additional aspects of MMF and MVP are:

- A distinct and deliverable feature of the system that provides significant values to the customer (can be sold / used immediately).
- Chosen for implementation after value-based prioritization.
- Can reap return on investment instantly.

Agile contracting: Although I never saw this on my exam, I highly recommend knowing a few of the more used agile contracting methods.

Money for nothing, change for free: Allows the customer to "change" features as long as they deprioritize less important features and it doesn't balloon up the cost. Also allows for an out-clause if the customer doesn't see any more value in the product backlog.

Graduated fix price contract: If the customer finishes early, they get paid more. If they finish on time, they get paid the normal rate. If they're behind schedule, they get paid less.

Fixed-price work packages: The concept of paying per each story or feature for a fixed amount of money, and each component works individually of every other component. If the price increases on one feature or story, it does not affect the other ones.

Knowing the difference between verification and validation in agile: This was an important one for me to understand and it took a while to get a strong definition of it.

Verification is basically, “Did your product meet the acceptance criteria that was required of you?”, “Did you build what you entered an agreement to build?” **Validation** is how well your actual feature or product addresses these requirements. For example, “Did you build what the customer actually needed?” Further in this post, I’ll explain the concepts of Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD), which will highlight these further.

Understanding these concepts will also help grow your knowledge in the development process, but I never saw them on my exam: Continuous integration, exploratory testing, usability testing, smoke testing, TDD and ATDD.

These are all development concepts, and will empower you to work more effectively with XP / hybrid agile development teams. All of these concepts greatly assisted in my understanding of development workflows and will allow you to resonate more cleanly with your delivery team.

Domain III. Stakeholder Engagement (3 subdomains, 9 tasks)

Engage current and future interested parties by building a trusting environment that aligns with their needs and expectations, and balances their requests with an understanding of the cost / effort involved. Promote participation and collaboration throughout the project’s life cycle and provide the tools for effective and informed decision making.

Knowing stakeholder management

Knowing the overall definition of stakeholders, which is anyone who can have an impact on a project (which includes the actual delivery team, product owner, and project manager) – according to PMBOK. However, according to some agile documentation, the delivery team is not a stakeholder.

Make sure you have an understanding of agile modeling and establishing a shared vision with the stakeholders of a project and delivery team. This is a key concept of agile as the development team, testers, designers, and analysts usually have a direct line (two-way) of communication with the stakeholders. They (the delivery team and stakeholders / customers) should know the definition of “done” and what constitutes completion.

Agile chartering

Know what agile chartering is and how it works. PMBOK 6 has a good definition of agile chartering and how traditional project charters document the “how” to build things and are highly detailed, whereas agile charters will define usually the following:

1. Participants
2. Business cases
3. Key stakeholders
4. Benefits
5. Risks
6. Objectives and constraints
7. Communication plans

Each of these will have their own breakdown and could constitute a lot of information. Consider mocking up your own agile charter for fun!

Here are some soft project management skills that I think you should know for the exam:

Communication management

Knowledge sharing: Know how and what **information radiators** are, how they work, and that they are one of the best ways to convey information to stakeholders. (I saw a lot of questions on this on the exam). I will explain information radiators in more depth further down below.

Also know the different methods of knowledge sharing to stakeholders, which are limited to:

- Personas (for understanding users of the project)
- Wireframes (low-fidelity, high-touch examples of a product)
- Mockups (advanced diagram on how a product feels and works)
- Use-case diagram (how the product and it components relate to its users)

Face-to-face communication: The best way to communicate is face-to-face. If possible, get your team together, and always choose face-to-face communication if it’s feasible and you have the money. There will be questions on this on the exam.

Two-way communication: Use a two-way communication model, or a collaboration model to communicate. Don't use a hierarchical dispatching model, as information can be lost.

Exam tip: Know the definition of "done" and all of its intricate parts (tested, validated, error-free, and deployed).

Emotional intelligence: The four quadrants of emotional intelligence (EI) and understanding them (self-awareness, self-management, social awareness, and relationship-management). I would also try to find or brainstorm some scenarios where having strong EI is important, since I did see some EI questions on my exam.

Active listening: The three levels of active listening (internal listening, focused Listening, and global listening). They all grow off each other and are important.

Agile leadership style: Servant leadership

Traditional leadership and management emphasizes command-and-control (i.e. Theory X: "Workers are lazy and need to be monitored closely").

Servant leaders will lead by acting as servers to their team to ensure the needs of team members are met and roadblocks are cleared (i.e. "servant first, leader second" mentality) (i.e. Theory Y: "Team members are self-motivated").

An agile servant leader needs to:

- Protect the team from interference, distractions and interruptions
- Remove impediments to the team's performance
- Communicate and re-communicate project vision – maintain a common vision to drive the team to perform
- Carry food and water (i.e. provide all the resources for the team to perform, including motivating the team and providing trainings)

Understanding red and green zones

This is understanding how leaders work, how to live / work in a red zone (aggressive, defensive, non-collaborate) vs. the green zone (supportive, responsible, reactive in a righteous way), and how good leaders should only operate in the green zone.

Workshops

Have a good grasp on workshops; why you do them and what they are used for. This ends up being baked into the concept of brainstorming methods and collaboration games. For these methodologies, I would Google them and pull the information from the relevant sites. Each could have their own step-by-step article.

Some ideas for brainstorming methods:

- Quiet writing
- Round-robin
- Free-for-all

Collaboration games

- Remember the Future
- Prune the Product Tree
- SpeedBoat (Sailboat)
- Buy a Feature
- Bang for the Buck

Negotiation

An important concept on the ACP: That a healthy negotiation is not a zero-sum game, and that someone does not have to “lose”. It’s all about give and take; there should be an investigation of different options and alternatives. An outcome should be collaborative and thoughtful to all parties.

Conflict resolution

There are five stages of conflict – in the order of light to severe:

Level of conflictAgile leader responseA problem to solve – A problem occurs or is presented, very likely not heatedConsider letting the team work through the problem.**Disagreement** – Everyone tries to protect their own interests, could be a heated, passionate debateAs long as there’s no insults or sides being taken, let the team work through the disagreement.**Contest** – Sides begin to be taken, it’s a “you vs. me” scenario.Personal attacks may come out here, and lines will be drawn. Consider mediating the conversation and addressing concerns of all parties.**Crusade** – “Us or them” mentality, individuals may be out for bloodAs a leader, you’ll need to respond here. Try to keep the meeting’s objective at the top of everyone’s mind and remind team members of their working agreements with each other.**World war** – All-out battle, sides

are drawn, insults could be rampantCall a break, split up parties. Become intermediary between the parties.

Domain IV. Team Performance (3 subdomains, 9 tasks)

Create an environment of trust, learning, collaboration, and conflict resolution that promotes team self-organization, enhances relationships among team members, and cultivates a culture of high performance.

Understand the concept of “adaptive leadership” and what it entails

How to encourage emergent leadership within the team, and facilitate growth of senior and junior members to take the lead in projects and tasks. Also understand the purpose of an agile coach and project manager from a micro-perspective of a coach and mentor, instead of task management and controller as in traditional style.

Understand the roles of agile project managers, the delivery team and product owner

I would say this is an important one, and I saw at least 2 to 3 questions referencing or needing to know that the project manager should shield outside interference from the delivery team, or that the product owner can cancel a sprint.

Know the team terms

Colocation, self-directed, self-organizing, emergent leadership (remember physical and virtual (digital teams)), tacit knowledge, team space and osmotic communication are all important aspects to know. There will definitely be one or two questions on this on the exam.

Caves and commons

Knowing how caves (a colocated place where there’s a quiet space to work) and commons (a place to collaborate with potential conferencing screens, multi-monitors, information radiators, and maybe even video games) work.

Development mastery models

Know the three developmental mastery models.

1. Dreyfus

The concept of being at different levels. As you grow and understand concepts and rules, things become more intuitive until you are a master of that specific skill. Created by Stuart and Hubert Dreyfus in the 1980s, there are five stages of expertise in this concept:

1. **Novice:** You are learning the rules, but you mostly have no true understanding of why you are using them, and are applying them based off of concepts learned or seen. For example: To build a website, you need to have pages, a domain name, and to host it somewhere like GoDaddy. To do this, you follow online guides on how to set up a website.
2. **Advanced beginner:** You have a better grasp of the rules and can apply them in certain situations with a relative understanding of the concepts. For example: To build a website, I have HTML pages with CSS, JS, and potentially some sort of backend language. On top of that, I use the same HTML template for all of my pages to save time.
3. **Competent:** You are comfortable with the rules, and you know what do when the time comes. For example, to build a website, I can use a number of Content Management System (CMS)s and frameworks, and I can tweak them myself, add plugins / add-ons, and slowly adjust the infrastructure for performance.
4. **Proficient:** You know what to do in almost all circumstances, and can deduce the best outcome and strategy to tackle any problem with little to no research. For example: You're a webmaster and you know how to build a website in many different ways. If an issue arises, you need very little research to solve the problem or none at all. You can also come up with custom solutions, which might be better than predefined rules.
5. **Expert:** Everything is intuitive – you know what do, how to do it, and the best answer to almost every problem. For example: You're an owner operator of a website; a master of the internet with multiple competencies in different adjacent disciplines. You intuitively know how to handle issues that come your way and very likely set the pace of competencies in your industry.

2. Shu, Ha, Ri

Similar to Dreyfus' adult skill development, except it's broken down into three concepts:

1. **Shu:** Understand the rules and be bound by them. Use these rules as guidelines and don't sweat understanding their underlying meanings.
2. **Ha:** Things become intuitive; the team begins to get into the motions and understand why things are done (essentially mastering the rules).
3. **Ri:** Master and ascend the rules, potentially tailoring as you understand the underlying meanings and practices.

3. Tuckman model

Forming, storming, norming, performing, and adjourning / mourning.

1. **Forming**: When the team is coming together for the first time and getting to know each other. They're not very efficient. The project manager should be very delegatory and may be very hands on with the team.
2. **Storming**: This is where the team begins to jive together, but there is a jockeying for power. The project manager should remain delegatory but begin to provide autonomy to the team.
3. **Norming**: The team is organized and begins get into the flow of work. This is where the project manager should start to step back from the day-to-day tasks, and become more supportive in their role.
4. **Performing**: The team encapsulates the essence of a true team. This is where you want your team to be. They are very likely self-organizing and self-governing.
5. **Adjourning / Mourning**: The team goes through a phase where it's broken up or they lose a team member. This can be a transition period back to any of the other stages.

Tuckman's model of group development

Understand concepts of mentorship and coaching

Know and understand the concepts of mentorship and coaching. More particularly, how to relate them to Tuckman's model. For example, in the forming and storming phases of Tuckman's model, an agile coach will need to be highly "supportive" and almost provide direction to the team to help them make it to the norming stage. Once there, they can move into more of a mentorship / true supporting role – allowing the team to self-direct and self-organize.

Domain V. Adaptive Planning (3 subdomains, 10 tasks)

Produce and maintain an evolving plan, from initiation to closure, based on goals, values, risks, constraints, stakeholder feedback, and review findings.

Team velocity

Velocity is a capacity planning tool used in agile projects, usually defined as the number of story points that are completed in an iteration.

Velocity usually increases gradually over the first few iterations as the team becomes more “performing”, but stabilizes as the product becomes more complicated (more bugs, documentations, dependencies, etc.).

Cycle time and throughput

Cycle time is the time necessary to get a single item of work done from start (idea) to finish (as a shippable product that delivers value). Cycle time can be reduced by shortening iteration time (breaking down task sizes), limiting work In progress, and reducing wastes.

Throughput is the number of things that can be done in an iteration.

Cycle Time = Work in Progress (WIP) / Throughput

Defect cycle time: The time between defect injection and defect remediation. The shorter the defect cycle time the better.

Remaining work: Likely cost remaining = Salary burn rate x remaining weeks left.

Likelihood of completion: Remaining work / rate of progress.

Estimating

Know concepts of estimating pulled from Mike Cohn’s book, “Agile estimating and planning”. I believe estimation and planning is one of the most important concepts in this domain. Know and understand the following terms:

Affinity estimating: Estimating based off of topic and size.

Initial velocity: Know the ways to estimate velocity: Historical, ideal time, affinity, and by running a few sprints (the best way).

Ideal time: Estimating in ideal time, which is the perfect amount of time a development team can devote to working on an increment (if it's two week sprints, potentially 75 hrs per developer).

Relative sizing: Estimating tickets based off of other tickets of similar size.

T-shirt sizing: Sizing tickets based off a broad ticket sizing like t-shirts (XL, L, M, S).

Planning poker and wideband delphi: Two methods of estimating tickets, where it's a group collaboration session and the delivery team will individually vote on the size and scope of each specific story.

Planning

Planning over the life span of a project is a hugely important concept in agile. Contrary to popular belief, there's usually more planning in an agile project than a traditional one, all else being equal. Also, you need to realize that the cost of change (how much it costs to change something) increases as development goes on. So, it's better to catch a defect or an issue in a paired programming session vs. a demo to your clients.

Some concepts you'll want to know are:

Story points: Related to estimation and the fact that you are looking at the level of effort for the work – not how much time it takes to actually finish a ticket.

Backlog refinement (or grooming): The reprioritizing of stories in the backlog, which is usually done by the P/O and the development team.

Slicing stories: The concept of cutting stories more finely to include all aspects of development to ensure that value is being driven to the customer.

Spikes: Two types of spikes: Architectural spike (spikes done to test new software or infra) and risk-based spike (spikes to see if something can work or not).

Failing fast: Concept of trying something and failing at it quickly to reduce the amount of waste. This helps foster innovation.

Daily standups: Setting the ground rules, roles, who can attend, as well as the three questions (what did you do yesterday, what are you going to do today, and are there any

impediments). The purpose of standup (daily scrum), and how it's the smallest level of planning and feedback.

You should also refresh your knowledge of product roadmap, story maps, user stories, and acceptance criteria.

Burn-up charts:

A chart that dictates the amount of sprints on x-axis and story points on the Y axis. It dictates how much work is being done over a period of time – great for forecasting project completion. If the story points increase, it could be an increase in scope for the project.

Weeks by story point

Burn down charts:

Similar to burn-up charts but in an inverse fashion (with the amount of story points descending until the project is done.

Timeline by Tasks

Domain VI. Problem detection and resolution (5 tasks)

Continuously identify problems, impediments, and risks; prioritize and resolve in a timely manner; monitor and communicate the problem resolution status; and implement process improvements to prevent them from occurring again.

Most of the material I studied revolved around understanding specific metrics, like lead time, cycle time, escaped defects, and EMV. However, you should know a lot about trend and variance analysis.

Trend analysis: Constitutes using metrics in the past to forecast the future. It's important especially for determining velocity. Key terms to know for trend analysis is leading (future) and lagging (past) metrics.

Variance analysis: The concept of common cause and special cause variance. So, knowing that there's a level of general variance for projects and accepting it (in common cause's case) and investigating it and getting to the root cause of a special cause variation.

One key thing to know is **value stream mapping** (it's a lean concept of mapping out non-value added and value added and reducing those wastes) and concepts like **kaizen** (which is also a lean concept), where you want to optimize at a basic level your individual work processes as an employee, instead of a giant optimization that happens in the West. All of this bakes into lead time, throughput, productivity, and process cycle efficiency (the metric you're trying to optimize for value stream mapping).

Knowing how to handle risk and calculate / prioritize was something that I studied for, but didn't see on my exam.

The three ways to handle or track risk are:

1. **Risk adjusted backlog:** Sounds like it is; a backlog that has risk-related stories prioritized within them.
2. **Risk burndown charts:** Sounds like it is , a burndown chart for risk instead of story point completion.
3. **Risk severity:** Calculating risk based off it's monetary value x the probability of it happening, or some ordinal metric (like 1, 2, 3) so that you don't get blinded by the numbers.

Finally, knowing that a team is the best way to solve problems. Since the development team is the closest to the project, always try to include them in problem solving as they will likely have the best answer!

Domain VII. Continuous improvement (product, process, people) (6 tasks)

Continuously improve the quality, effectiveness, and value of the product, the process, and the team.

Last domain of notes, but can basically be summed into a few sentences and topics.

Process tailoring and agile hybrid models

Know how process tailoring works and how / when to do it. Know some types of process tailoring (Scrumban or ScrumXP) for personal use, and to always tailor your process for a good reason when data is available.

For example: My agile team went remote due to COVID-19. I instituted more user story workshops because it takes longer to write user stories remotely when everyone is

yelling over a digital conference (every workshop had gone 30 minutes over for the last five sprints). Use those lagging metrics to define your plan!

Retrospective

Ahhh, the good old retrospective. Know the phases of retrospective, including: 1) set the stage; 2) gather data; 3) generate insights; 4) decide what to do; and 5) retrospect the retrospective.

Here's how I usually imagine each stage:

1. **Set the stage:** Set the ground rules and purpose of the retrospective. You should have invited everyone that's appropriate if you're a scrum master. Also, you'll want to gauge the temperature by doing an anonymous survey called an ESVP survey (explorer, shopper, vacationer, and prisoner). Based off that, if the survey is negative you may need to reset the room.
2. **Gather the data:** Personally, I've had this done before with specific team members to expedite the process and used digital dashboards like easyretro.io to facilitate the knowledge transfer. However, PMI wants there to be a comprehensive data dive with all of the relevant team members in the room together.
3. **Generating insights**: Next, you want to generate insights on the issues that have come up. You can use the 5 Why's, fishbone analysis, "mad glad sad" voting, and many other tools to find the root causes of issues.
4. **Decide what do to do:** Figure out what you want to do by everyone voting or classifying everything into "stop doing", "keep doing" and "start doing". Make sure the P/O is there to record these tasks and turn them into tickets in the product backlog.
5. **Close the meeting**: Thank everyone for coming and retrospect the retrospective. Also, have a quick cheer for how you did as this is usually a two-hour ordeal.

Systems thinking: I actually don't remember this on my exam, but it's a great way to think about how to approach complex problems with agile. It's more of a high level look and feel of what strategy you should take with a project based off its complexity. Generally, it's the concept of thinking how pieces of a concept interact with each other.