Agile Project Management

This document is a quick review of the agile project management Iterative process.

1. **Introduction to Agile**

   - Manifesto for Agile Software Development

     We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

     - Individuals and interactions over processes and tools
     - Working software over comprehensive documentation
     - Customer collaboration over contract negotiation
     - Responding to change over following a plan

     That is, while there is value in the items on the right, we value the items on the left more.

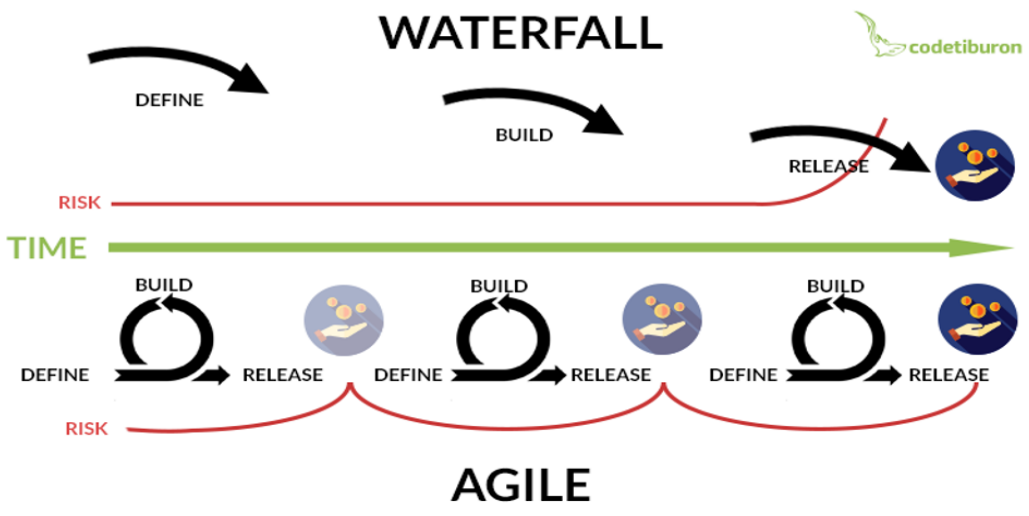
   - What?

     Agile project management is an iterative approach to managing software development projects that focuses on continuous releases and incorporating customer feedback with every iteration.

   - Why?

     With today's customers and businesses requiring rapid responses and changes, agile provides the flexibility to adjust and iterate during the development process.

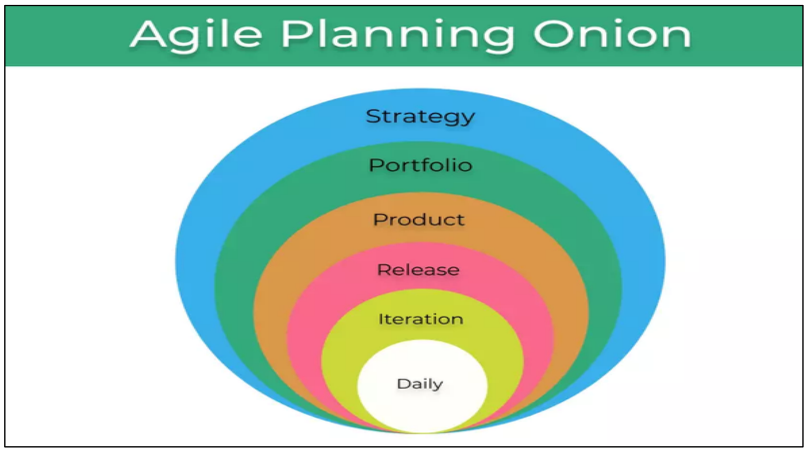
   - Agile vs Waterfall
     The waterfall project management approach entails a clearly defined sequence of execution with project phases that do not advance until a phase receives final approval. Once a phase is completed, it can be difficult and costly to revisit a previous stage. Agile teams may follow a similar sequence yet do so in smaller increments with regular feedback loops.



- Agile Principles
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.
  - Business people and developers must work together daily throughout the project.
  - Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
  - The 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 progress.
  - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
  - Continuous attention to technical excellence and good design enhances agility.
  - Simplicity--the art of maximizing the amount of work not done--is essential.
  - The best architectures, requirements, and designs emerge from self-organizing teams.
  - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

2. Planning

- Agile can be applied to every level of the project. It is fundamentally essential to grasp the idea that Agile Planning is applicable to every slice of the onion. You don't just do Agile on the team level (Day, Iteration, Release). You can have an Agile product management service, Agile portfolio management, an Agile strategy, etc. Being agile in your strategy is what defines business agility in general. Agile planning is iterative. This means that you develop and adjust your plan multiple times, as you find necessary.



3. Estimating (relative units vs. time units)

Estimating time or effort related to a user story or task card helps us prioritize.

- Relative Estimates (Story point as a unit)

  “It is better to be roughly right than precisely wrong.” (John Keynes)

  A story point is a metric used in agile project management and development to estimate the difficulty of implementing a given user story, which is an abstract measure of effort required to implement it. In simple terms, a story point is a number that tells the team about the difficulty level of the story. Difficulty could be related to complexities, risks, and efforts involved.

- What?

  Agile estimating uses relative sizing to provide a realistic way for teams to forecast work. There are two parts that play the role in relative estimations:

  1.The size of a task or story is what must be estimated. It’s made up of three factors:

  a. Effort. How much work is required to complete this task?

  b. Complexity. How difficult or complicated is this task?

  c. Uncertainty. Do we know exactly what must be done to accomplish this task, or will we need to learn as we go?

2. The size is relative to the other stories your team may have on its plate.

   It’s easier and more effective to compare tasks and determine which is larger or smaller, rather than assign numbers or sizes to tasks independently without a reference point.

- Which Scale?

  In most cases a story point uses one of the following scales for sizing:

  - 1,2,4,8,16
  - X-Small, Small, Medium, Large, Extra-Large (known as “T-Shirt Sizing”)
  - Fibonacci sequence: 1,2,3,5,8,13,21
  - Zoo scale (ant, cat, goat, hippo, ..)
  - Vehicles (skateboard, bicycle, car, bus, train, plane, shuttle)

- How?

  1. List all the stories to be sized
  2. Put them in order from smallest to largest
     - Take the first user story
     - Then take the second user story
     - Decide which is bigger and put the bigger one above
     - Then take the next one and decide where it fits relative to the other two
     - Then take the next one and do the same
     - Repeat the process until all the stories are now in the list (in a sequence from smallest to largest)

- Size the stories

  You can also play some planning poker (eg https://planningpokeronline.com)

- Time Estimates (time as a unit): Typically, we use minutes and hours as unit.

- Actual (time as a unit): As we track the task to completion it is help to record the actual time used.
- Don't be embarrassed if you were way off! Learn from this.

4. Applying 20%, 80% Rule

- Pareto Principle

  The Pareto Principle, commonly referred to as the 80/20 rule, states that 80% of the effect comes from 20% of causes.

  Consultants, economists, business leaders, and time-management experts continue to find more examples of the 80/20 rule in many areas of life. For example:

  - 80% of the teacher's time spent on 20% of students
  - 80% of sales come from 20% of customers
  - 20% of your product experience will lead to 80% of all support cases
  - 20% of the tasks on your to-do list will generate 80% of the benefit from the entire list
  - 80% of all stress comes from only 20% of all types of stressors

  There are several ways agile product managers can use the 80/20 rule. Below are the two use cases in supporting Agile Best Practices:

  1. Breaking out your long-term vs. short-term focus

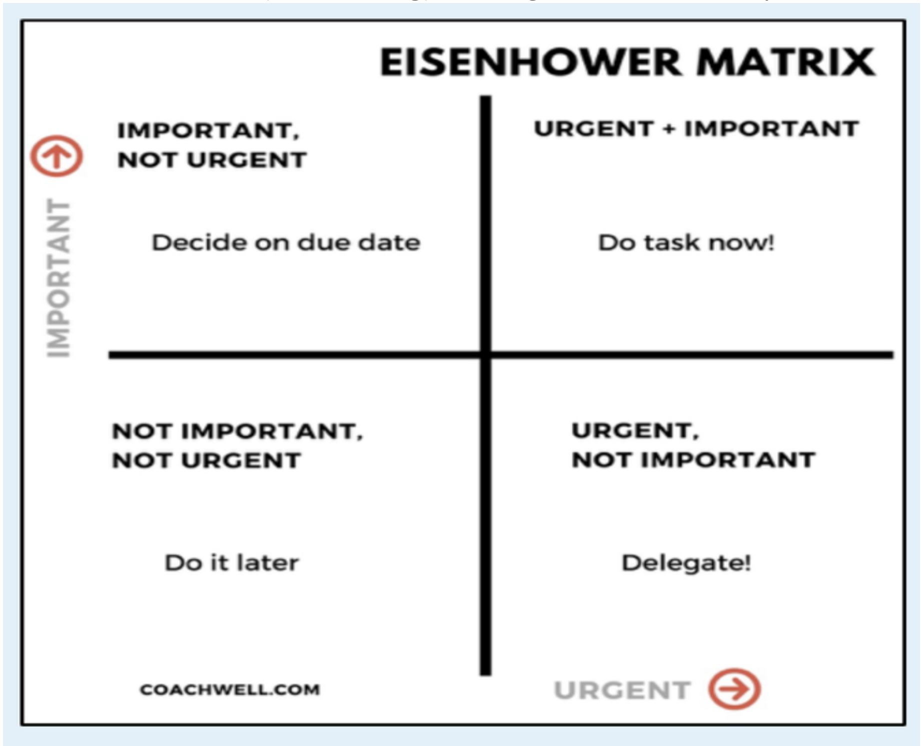
     80% of your time on long-term strategic preparation and 20% on the tactical and logistical details.

  2. Prioritizing the most strategically valuable features

     Focus on those "vital", 20% of the core features on your backlog and give them 80% of your team's time that will meet the need of 80% of the users.

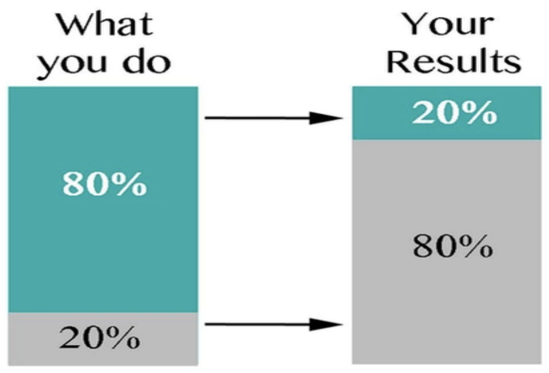
5. Prioritizing

- Once we have a list (aka backlog) of things to do, we must prioritize what to do first!



Eisenhower Matrix from Coachwell

- What items are urgent?
- What things are important?
- What things give us most gain? (Given limited time.)
- 20/80 rule: “What would be best bang for your buck?”



20% of what you do
leads to 80% of your results.

Do the 20% that matters and forget the rest.

6. SMART (again!)

You will systematically, and progressively, break down what needs to be done for your project. Here's the progression:

- EPICS: captures persona's life goals, experience goals (relative estimation)
- FEATURES: captures user's task goals, interaction goals (relative estimation)
- TASKS: captures what developers will do (SMART - actual estimation)

When it’s time to write sprint level task, each task you identify must be:

i. Specific, i.e., you’re not building a database, you’re developing the schema for five tables.
ii. Measurable, i.e., we can measure whether it is incomplete, complete, etc. This is particularly important for Research. It only counts if you made a prototype or a proof of concept or some sort of deliverable to share with your team.
iii. Achievable, i.e., it’s something a developer can do with the tools at your disposal
iv. Relevant, i.e., the task must be directly relevant to completing one or more of the user stories you identified
v. Timeboxed, i.e., each task can be completed in one hour (this may be difficult to estimate at first, and that’s okay, you will get better!)

7. Pair Programming

Pair programming is an Agile software development technique originating from Extreme programming (XP) in which two developers’ team together on one computer. The two people work together to design, code, and test user stories.

- Pair programming styles:

  i. Driver/Navigator Style:

     - In this style, one programmer (the driver) handles the mechanical tasks, like writing the code, while the other (the navigator) focuses on the strategic elements, like reviewing the code.
     - The roles are often switched between the two programmers, and this style works well when pairing a novice with an expert.
     - The navigator’s role can vary from a reserved oversight to a more hands-on tactical approach.

  ii. Unstructured Style:

     - Most pair programming sessions fall into this style, where two programmers collaborate loosely without a set structure.
     - This style works best when both programmers have similar skill levels.
     - A common variant is the unstructured expert-novice pair, where an experienced programmer is paired with a less experienced one.

iii. Challenges of Unstructured Style:

- This approach is hard to maintain discipline and tends to struggle in long-term projects, especially in remote settings.
- Unstructured pair programming can be considered when the team is unsure about which style will work best for the project.

iv. Ping-Pong Style:

- In this approach, one developer writes a test, and the other makes the test pass.
- The two programmers alternate roles regularly, ensuring that no one developer dominates the workflow.
- This style is often used alongside Test-Driven Development. (https://www.techtarget.com/searchsoftwarequality/definition/test-driven-development).

▪ Pair programming skill pairs:
Development teams should also choose pair programming styles that align with the skills of the programmers involved. Potential skill pairs include:

a. Expert/Expert Pairs:
   - Two experts can generally work within any pair programming style.
   - Advanced programmers may prefer the ping-pong style, as it allows them to have even participation.

b. Novice/Novice Pairs:
   - Two novices together may have difficulty in the driver/navigator style, because no one is experienced enough to take charge.
   - The unstructured approach may be difficult for beginner programmers.

c. Expert/Novice Pairs:
   - The most common skill combination is an expert programmer working with a less experienced person.
   - Experts rely on their depth of knowledge to direct the activity, while the novice can learn more from the expert.

▪ Advantages of Pair Programming:

a. Fewer Coding Mistakes:
   - Another programmer is looking over the driver's code, which can help reduce mistakes and improve the quality of the code.

b. Knowledge Sharing:
   - Knowledge is spread among the pairs.
   - Junior developers can pick up more skills from senior developers.

    - Those unfamiliar with a process can be paired with someone who knows more about the process.

  c. Reduced Effort to Coordinate:
    - Developers will get used to collaborating and coordinating their efforts.

  d. Increased Resiliency:
    - Pair programming helps developers understand each part of a codebase (https://www.techtarget.com/whatis/definition/codebase-code-base), meaning the environment will not be dependent on a single person for repairs if something breaks.

- Challenges of Pair Programming:

  a. Efficiency:

    - Common logic might dictate that pair programming would reduce productivity by 50% because two developers are working on the same project at one time.
    - According to a blog post (https://raygun.com/blog/how-good-is-pair-programming-really/) on Raygun, pairs work about 15% slower, which is an improvement but is still less than the productivity of two separate programmers.

  b. Equally Engaging Pairs:
    - If both developers do not equally engage in the project, then there is less chance that knowledge will be shared, and it is highly probable that one developer will participate less than the other.

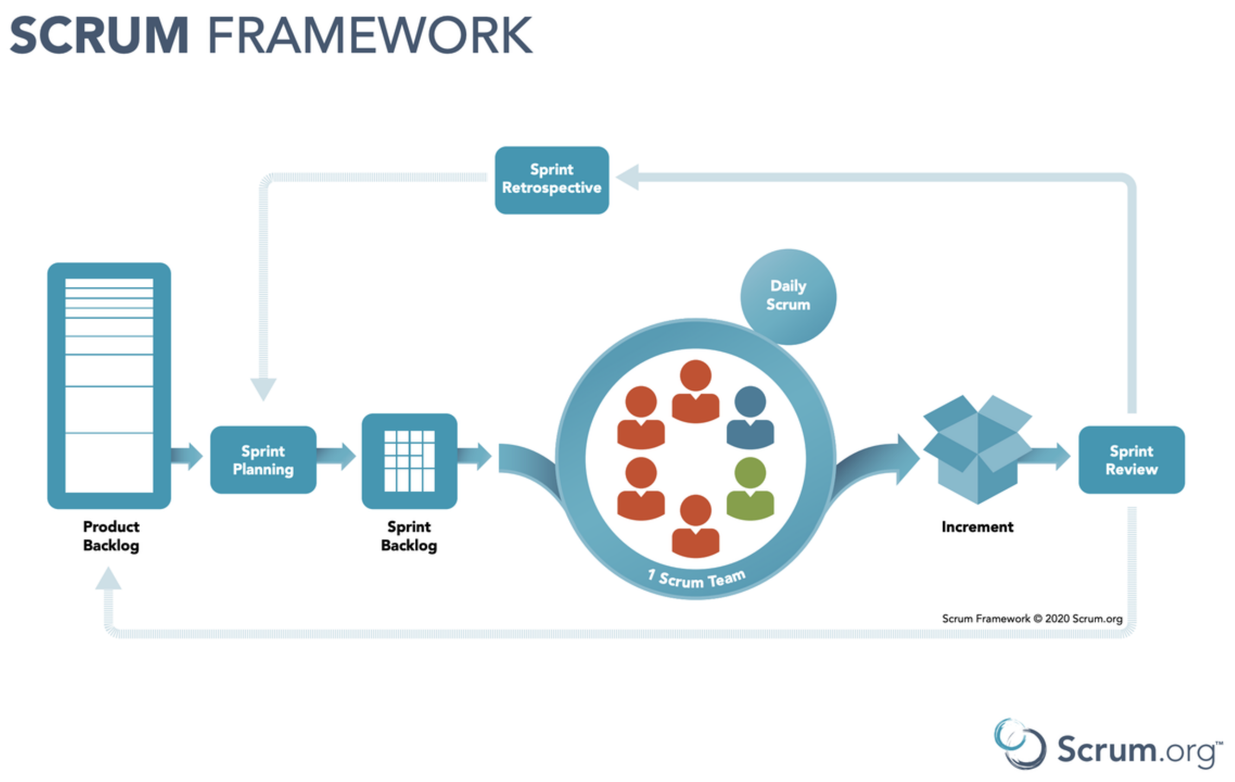
  c. Social and Interactive Process:
    - It is hard for those who work better alone.
    - Pairs that have trouble together might be better suited to work by themselves; forcing them to collaborate may hurt their work ethic.

  d. Sustainability:
    - The pace may not be suited to practicing hours at a time.
    - Likely, developers will need breaks at different times.

8. **Scrum**
Scrum (named after a rugby team's huddle) is a lightweight framework for doing Agile project management.

- Basics
The core of scrum is simple - the three pillars:
  1. Transparency: All aspects of the project must be visible to those responsible for the outcome. Everyone shares their progress and challenges openly.
  2. Inspection: Regularly examining the progress toward the sprint goals. Everyone inspects each other's work to ensure quality and alignment.

3. Adaptation: Based on the results of inspections and feedback, adjustments are made to improve future performance.



- The Scrum Framework
  The Scrum Framework consists of several key components:

  1. Product Backlog: The product backlog is a prioritized list of features, bug fixes, and technical improvements to be completed. The Product Owner manages and prioritizes the backlog to ensure the highest value items are worked on.

  2. Sprint Planning: In Sprint Planning, the Scrum Team selects items from the product backlog that they commit to completing during the sprint. This creates the Sprint Backlog, which guides the team's work for the sprint.

  3. Sprint Backlog: The Sprint Backlog is the specific set of product backlog items that the team selects for the sprint. It includes tasks and user stories that the team believes can be completed within the sprint timeframe.

  4. Daily Scrum (Stand-ups): The Daily Scrum or "Stand-up" is a short, daily meeting (typically 15 minutes) where team members provide updates. Each member answers three questions:

        i. What did I accomplish yesterday?
        ii. What will I work on today?
        iii. Are there any impediments preventing me from making progress?

5. Increment: At the end of the sprint, the team delivers a working, shippable product increment. This increment must meet the Definition of Done to ensure that it is ready for release.

6. Sprint Review: At the end of the sprint, the team presents the completed increment to stakeholders in a Sprint Review. Feedback is gathered, and new items may be added to the product backlog.

7. Sprint Retrospective: After the sprint review, the team holds a Sprint Retrospective to discuss what went well, what didn't, and how the process can be improved for the next sprint.

8. Scrum Team Composition
   A Scrum Team consists mainly of:
   - Product Owner: Manages the product backlog and maximizes the value of the product.
   - Scrum Master: Ensures the team adheres to Scrum principles, facilitates meetings, and removes obstacles.
   - Development Team: Developers responsible for delivering the product increment.
   - Stakeholders: These are external individuals or groups that have an interest in the outcome of the product but are not directly involved in day-to-day operations. This includes investors, customers, etc.

▪ Scrum Process Cycle
The Scrum Process Cycle is a continuous loop of planning, execution, review, and adaptation that ensures iterative delivery and improvement of a product. The cycle consists of several key stages:
   - The Product Backlog feeds into Sprint Planning, where items are selected for the Sprint Backlog.
   - The team works on the sprint backlog, holds Daily Scrums, and delivers a Product Increment at the end of the sprint.
   - The sprint ends with a Sprint Review and Sprint Retrospective to inspect, adapt, and improve the process for the next sprint.

▪ In a nutshell, this framework requires:

- One Product Owner responsible for managing the product backlog.
- Scrum Team members, typically developers, who deliver increments of work.

- Prioritizing and selecting a list of tasks from the Product Backlog for a Sprint iteration.
- The Scrum Master facilitates daily quick updates (also known as “Stand-ups”) during the sprint.
- Reassessing and reiterating for the next sprint cycle.

9. YouTube Link

- Please check the youtube link
- This is how the weekly meeting is supposed to be conducted
- https://youtu.be/LHKe9KbiDgA?si=5wIkV_pWLPejE623

10. Trello

- Please check the youtube link
- https://youtu.be/geRKHFzTxNY?si=6qoqnfPHLNOmYGI
- Use the Trello to manage your project (https://trello.com/)

11. References:

1. https://agilemanifesto.org/principles.html
2. https://www.atlassian.com/agile/project-management
3. https://kanbanize.com/agile/project-management/planning
4. https://www.visual-paradigm.com/scrum/what-is-agile-software-development/
5.[https://www.techtarget.com/searchsoftwarequality/definition/Pair-programming#:~:text=Pair programming is an Agile, code and test user stories
6. https://www.netsolutions.com/insights/how-to-estimate-projects-in-agile/
7. https://www.productplan.com/learn/80-20-rule-agile/
8. https://clichq.com/the-urgent-important-matrix/