# **Agile Software Development Overview:**

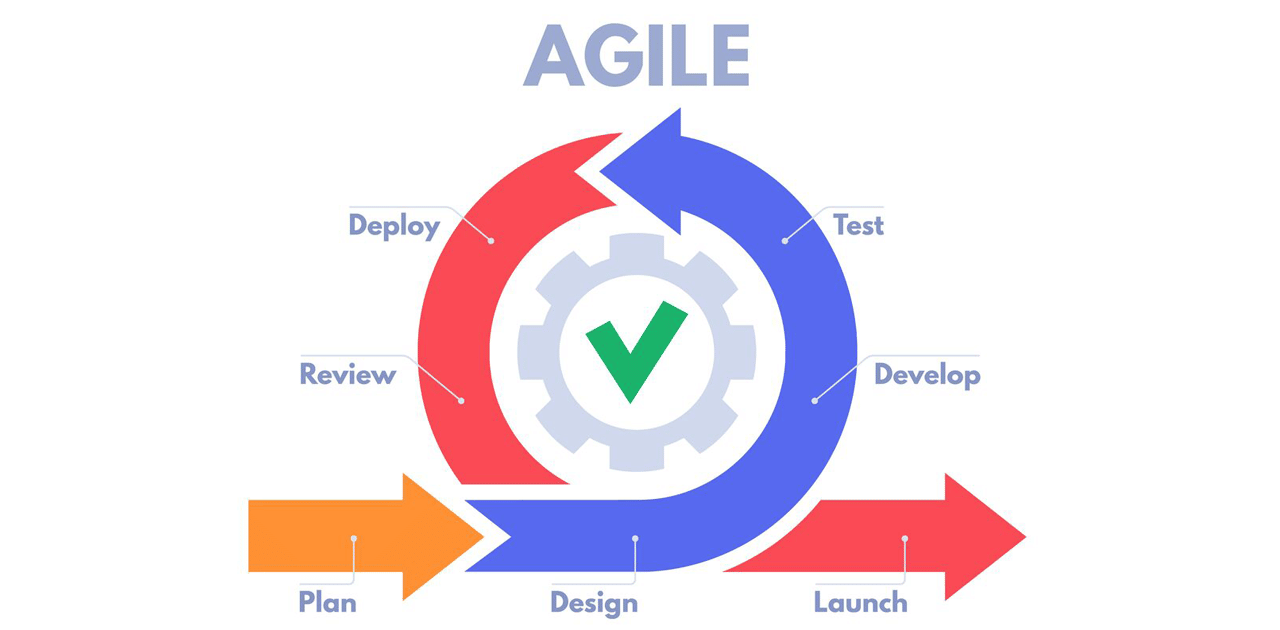
Agile is a flexible and iterative approach to software development that emphasizes collaboration, customer feedback, and adaptability. Key components include:

1. **Iterative Development:** Work is divided into small, manageable increments called sprints (usually 1–4 weeks long).
2. **Collaboration:** Continuous communication between developers, stakeholders, and users.
3. **Adaptability:** Changes in requirements are welcomed, even late in the development cycle.
4. **Deliverables:** At the end of each sprint, a functional and testable product increment is delivered.
5. **Principles:**
   * Focus on delivering value.
   * Keep the process simple.
   * Prioritize individuals and interactions over processes and tools.

### Stages in Agile:

1. **Concept/Initiation:** Define the project scope and objectives.
2. **Sprint Planning:** Identify what can be achieved in the next sprint.
3. **Execution and Daily Meetings:** Develop and test features.
4. **Review and Retrospective:** Evaluate the sprint’s outcomes and plan for the next.

### Agile Model:



### Water Fall Model:

A diagram of a software development process

Description automatically generated

### Resources for a Deeper Dive:

1. [Agile Manifesto](https://agilemanifesto.org) – The foundational principles of Agile.
2. **Books:**
   * *Scrum: The Art of Doing Twice the Work in Half the Time* by Jeff Sutherland.
   * *Agile Estimating and Planning* by Mike Cohn.
3. **Online Courses:**
   * [Agile with Atlassian Jira (Coursera)](https://www.coursera.org/learn/agile-with-atlassian-jira).
   * [Introduction to Agile (LinkedIn Learning)](https://www.linkedin.com/learning/topics/agile).
4. **YouTube Videos:**
   * *Agile Development in 10 Minutes* by Simplilearn.
   * *What is Agile?* by Atlassian.

# **Project Development Plan:**

## Sprint 1 :

Date: 12/11/2024

**First Sprint Goal: Build the Foundation**

**Duration:** 1–2 weeks  
**Objective:** Set up the project structure, understand software planning basics, and implement task creation and data storage.

**Sprint Tasks:**

**1. Project Setup (Day 1–2):**

* **Install necessary tools:**
  + ~~IDE for C++ development (e.g., Visual Studio or Code::Blocks).~~
  + ~~A beginner-friendly GUI framework like~~ [**~~Qt~~**](https://doc.qt.io/qt-6/qt-for-education.html) ~~or~~ [**~~wxWidgets~~**](https://www.wxwidgets.org/)~~.~~
* **Initialize Git Repository:**
  + ~~Set up the folder structure with directories for source code, resources, and saved data.~~
  + ~~Create a .gitignore file to exclude unnecessary files from commits.~~

**2. Task Management Basics (Day 3–5):**

* **Understand User Stories:**
  + Example: *"As a user, I want to create a task with a name and description so I can track its progress."*
* **Implement Core Functionality:**
  + ~~Write a simple console-based program to accept task names and descriptions.~~
  + ~~Save tasks to a local text file (backend logic).~~

**3. GUI Basics (Day 6–9):**

* **Learn GUI Framework Basics:**
  + ~~Study how your chosen framework handles windows, buttons, and text fields.~~
* **Create the First GUI Screen:**
  + ~~Design a simple window with input fields for task name and description.~~
  + Add a "Save Task" button.

**4. Testing and Review (Day 10–12):**

* ~~Test the GUI and backend to ensure tasks are saved correctly.~~
* ~~Push changes to GitHub.~~
* ~~Document what you’ve learned and challenges you faced.~~

## Sprint 2 :

Date: 16/11/2024

**Sprint Duration:**

7-10 days (adjustable based on your pace)

**Sprint Goals:**

1. Enhance GUI aesthetics and usability.
2. Replace current file management with JSON for better structure and scalability.
3. Introduce the multi-task feature with task descriptions.
4. Continue learning CSS to improve GUI design skills.

**Sprint Tasks:**

**1. Learning & Research**

* **Timeframe:** 1-2 days (concurrent with other tasks)
* **Tasks:**
  + Continue the **CSS full course** (target key sections: layout, animations, and responsive design).
  + ~~Research JSON basics and understand its structure (e.g., key-value pairs, arrays).~~
  + ~~Explore examples of GUI layouts that can manage multiple tasks effectively (inspiration for PyQt5 widgets).~~

**2. GUI Enhancements**

* **Timeframe:** 2-3 days
* **Tasks:**
  + ~~Improve the~~ **~~current GUI layout~~** ~~using CSS styles:~~
    - ~~Focus on button designs, font styles, color schemes, and overall alignment.~~
    - ~~Use PyQt5's QSS (Qt Style Sheets) to implement CSS-like styles for widgets.~~
  + Add a **dynamic list view** for displaying multiple tasks with their descriptions:
    - Each entry includes a task name, streak count, and description.
    - Include edit and delete buttons for each task.
  + Explore transitions or animations (e.g., smooth updates when tasks are added or deleted).

**3. JSON File Management**

* **Timeframe:** 2-3 days
* **Tasks:**
  + ~~Replace the current~~ **~~text file-based system~~** ~~with~~ **~~JSON~~** ~~for:~~
    - ~~Storing tasks, streaks, and descriptions in a structured format.~~
    - ~~Making file management more efficient and scalable.~~
  + ~~Implement a function to:~~
    - ~~Load data from a JSON file when the app starts.~~
    - ~~Save updated data to JSON when changes occur (e.g., new task added, streak updated).~~
  + ~~Validate data integrity during JSON read/write operations.~~

**4. Multi-Task Functionality**

* **Timeframe:** 2-3 days (overlaps with JSON implementation)
* **Tasks:**
  + ~~Create a~~ **~~Task Manager GUI Section~~**~~:~~
    - ~~Add a form for users to~~ **~~create new tasks~~** ~~(fields: Task Name, Description).~~
    - Display tasks in a list with the ability to **select and log streaks** for each task.
  + ~~Update~~ **~~taskmanager.py~~** ~~methods to:~~
    - ~~Handle multiple tasks (add, update streak, delete).~~
    - ~~Interact with the new JSON-based backend.~~
  + ~~Integrate with the GUI (buttons to add, update, and delete tasks).~~

**Deliverables:**

1. A CSS-enhanced GUI with a polished design.
2. A JSON-based file management system.
3. Multi-task management functionality.
4. Updated README documentation reflecting new features.

**Stretch Goals (Optional Enhancements):**

* Add a **search/filter feature** for tasks.
* Explore adding **notifications** for streak milestones or missed logs.
* Start experimenting with animations for visual rewards.

**Daily Plan:**

* Dedicate at least **2-3 focused hours daily** for these tasks.
* Track your learning streak with your app to stay motivated!

## **Sprint 3**

Date: 30/11/2024

The focus of Sprint 3 will be **documentation**, **testing integration**, and **enhancing task details display**. We will also ensure the project is optimized for scalability to support future features, laying a strong foundation for future development.

### Goals for Sprint 3

1. **Document and Optimize Codebase**
   * Document key features, modules, and workflows for easy understanding and future collaboration.
   * Refactor and optimize code for readability, maintainability, and scalability.
2. **Integrate Testing**
   * Learn pytest basics and write tests for critical functionalities.
3. **Enhance Task Details Display**
   * Add functionality to display detailed information about tasks in the Add/Remove window.
4. **Research Notifications**
   * Start exploring notification systems to enhance user engagement.

### Sprint 3 Breakdown

| **Task** | **Description** | **Estimated Time** | **Dependencies** | **Priority** |
| --- | --- | --- | --- | --- |
| **Code Documentation** | Write comments, docstrings, and a README file to explain the purpose of each module and its functionality. Include dependencies and setup instructions. | 2 days | None | High |
| **Code Optimization** | Refactor the code for clarity and scalability. Break down large functions, remove redundancies, and follow PEP 8 guidelines. | 2 days | None | High |
| **Learn pytest Basics** | Understand unit testing with pytest. Study fixtures, test cases, and coverage tools. | 2 days | Internet for research | High |
| **Testing Integration** | Write and implement basic unit tests for streak calculations, JSON file handling, and GUI element interactions. | 2 days | Completion of pytest learning | High |
| **Task Details Display** | Add a section in the Add/Remove window to display selected task details dynamically. | 3–4 days | Completed GUI integration | High |
| **Notification Research** | Explore Python notification libraries like plyer or OS-specific solutions to understand how to integrate notifications in the app. | 1–2 days | Internet for research | Medium |

### Task-Specific Details

**1. Code Documentation**

* **Purpose**: Improve code clarity for you and future developers.
* **Steps**:
  + Add docstrings to all functions and classes, explaining their input, output, and purpose.
  + Update the README with the app's description, features, and setup instructions.
  + Include a roadmap for planned features to showcase future goals.

**2. Code Optimization**

* **Purpose**: Enhance performance and readability.
* **Steps**:
  + Analyze the codebase to identify repetitive patterns or redundant code.
  + Refactor large functions into smaller, reusable ones.
  + Follow PEP 8 standards using tools like flake8 or black.

**3. Learn pytest Basics**

* **Purpose**: Introduce automated testing to ensure reliability and maintainability.
* **Steps**:
  + Understand pytest syntax and write simple test cases.
  + Learn how to use fixtures for setup and teardown in tests.
  + Explore tools like pytest-cov for code coverage reports.

**4. Testing Integration**

* **Purpose**: Ensure the app is reliable and free from critical bugs.
* **Steps**:
  + Write tests for streak calculations, ensuring edge cases are handled.
  + Test JSON file handling (read, write, and update).
  + Explore testing GUI interactions, focusing on user input and output behaviors.

**5. Task Details Display**

* **Purpose**: Improve the Add/Remove window’s usability and provide users with more information.
* **Steps**:
  + Add a QTextEdit or QLabel widget to display task details.
  + Update the combobox or list view to trigger a function when a task is selected.
  + Dynamically fetch and display the selected task's details from the JSON database.

**6. Notification Research**

* **Purpose**: Lay the groundwork for integrating notifications in the app.
* **Steps**:
  + Research libraries like plyer for cross-platform notifications.
  + Explore platform-specific options (e.g., Windows Toast notifications).
  + Draft a plan for how and when notifications should be triggered (e.g., streak achievements, reminders).

### Timeline

| **Day** | **Task** |
| --- | --- |
| Day 1 | Start code documentation. |
| Day 2 | Complete documentation; begin optimization. |
| Day 3 | Finalize optimization; start learning pytest. |
| Day 4 | Complete pytest basics; start writing tests. |
| Day 5 | Write and implement more test cases. |
| Day 6 | Begin enhancing task details display. |
| Day 7 | Complete task details display; research notifications. |

### Post-Sprint 3 Reflection

After completing Sprint 3, you should have:

1. A well-documented and optimized codebase ready for scalability.
2. Basic testing integrated with reliable unit tests.
3. A more user-friendly Add/Remove window displaying task details.
4. A clearer understanding of how to integrate notifications in the future.

## Documentation:

**Why Choose wxWidgets Over Qt?**

1. **Simplicity and Lightweight**:
   * **wxWidgets** is often considered easier to use for beginners compared to Qt. It follows more closely to native OS look-and-feel, which means it tends to have a simpler setup and fewer abstractions.
   * It provides a more "native" experience for different operating systems, as it wraps native controls and doesn’t require you to learn a complex framework.
2. **Licensing**:
   * **wxWidgets** is open-source and more permissive under the LGPL license, which can be more suitable for certain projects if licensing is a concern.
   * **Qt** uses a more restrictive license, especially in its commercial versions, which could become a roadblock for projects that want to avoid licensing issues.
3. **Ease of Use**:
   * **wxWidgets** is often easier to get started with for small to medium-sized projects, especially for a beginner.
   * **Qt** is a bit more complex and designed for larger-scale, feature-rich applications. It comes with many advanced features that could add complexity to a simple project.

**Complexity Difference: wxWidgets vs Qt**

1. **Learning Curve**:
   * **wxWidgets** has a **gentler learning curve**, which makes it more beginner-friendly. It’s more intuitive for those who are just starting with GUI programming.
   * **Qt**, on the other hand, is **more feature-rich** and designed for more complex applications. The learning curve is steeper, but it comes with a broader set of tools for development (e.g., Qt Quick, QML for UI, etc.).
2. **Documentation and Community**:
   * **Qt** has more extensive documentation and a larger community, which can be helpful when you need advanced features.
   * **wxWidgets** has decent documentation but not as extensive as Qt’s, especially when it comes to more advanced topics.
3. **Advanced Features**:
   * **Qt** provides more built-in features for complex GUI elements, such as **animation, networking, and advanced graphics rendering**, which wxWidgets doesn't focus on as much. Qt is often preferred for **highly interactive or visually complex applications**.
   * **wxWidgets** focuses on being lightweight, so it doesn't have as many of the advanced features that Qt offers out-of-the-box.

**Will You Need to Change a Lot of Code to Transition from wxWidgets to Qt?**

1. **Fundamental Code Changes**:
   * **Yes**, transitioning from **wxWidgets to Qt will require significant changes in the code**.
   * **UI elements** (like buttons, panels, text fields, etc.) are handled differently in both frameworks. wxWidgets uses wxButton, wxTextCtrl, etc., while Qt uses QPushButton, QLineEdit, etc.
   * The **event handling mechanism** is different. wxWidgets uses event tables for event handling, whereas Qt uses **signals and slots** for event handling.
   * The **layout management** in both frameworks differs, so you'd have to rewrite how your GUI components are arranged.
2. **Code Reuse**:
   * There might be **some reusable logic** such as backend functions (for your streak tracking, database management, etc.), but the GUI code will largely need to be rewritten due to the differences in how the two frameworks structure and manage UI components.
3. **Transition Complexity**:
   * The transition complexity is **medium to high**, depending on how much of your application depends on the GUI. If your application has a very simple GUI, transitioning will be easier.
   * For more **complex GUI layouts**, the transition will require more work because of differences in layout management, widgets, and event handling.

**Summary: Should You Stick with wxWidgets or Switch to Qt?**

1. **Choose wxWidgets** if:
   * You are a beginner and need a framework that’s easier to learn.
   * You want a framework that’s simpler to use for small to medium-sized projects.
   * You prefer the licensing and open-source nature of wxWidgets.
   * You are aiming for a lightweight, simple application without a need for advanced features.
2. **Choose Qt** if:
   * You expect to build a more feature-rich, complex, or high-performance application.
   * You want advanced features like animations, better graphics support, and easier handling of complex UI elements.
   * You’re comfortable with a steeper learning curve and can handle the additional complexity for future scalability.

**In Conclusion:**

* **Transitioning from wxWidgets to Qt** will **require significant code changes**, especially in the GUI section (widgets, events, layouts, etc.).
* If you start with wxWidgets and later want to switch to Qt, the transition will not be trivial, but it's feasible if you design your project in a modular way, separating the core logic from the UI.
* **Qt** will offer more advanced features in the long term but at the cost of increased complexity, whereas **wxWidgets** will let you build your application faster and with less overhead.

If you’re aiming for a relatively simple project, wxWidgets will likely be a better starting point. You can always revisit Qt later if you need more advanced features.

**What Are User Stories?**

User stories are short, simple descriptions of a feature from the user's perspective. They are written in a way that explains *who* the user is, *what* they want to achieve, and *why* they want to achieve it. User stories are a fundamental part of Agile project management and help break down a project into manageable, user-focused tasks.

**Format of a User Story:**

**As a [type of user], I want [some goal] so that [reason].**

This format ensures that every feature is designed with the user's needs in mind.

### Your Project's User Stories

Based on your project's description, here are some possible user stories:

1. **Task Management**:

As a user, I want to create a task with a name and description so that I can track its progress.

1. **Streak Tracking**:

As a user, I want to track streaks for each task so that I stay motivated and maintain consistency.

1. **Reward System**:

As a user, I want to receive visual rewards when I reach certain milestones so that I feel accomplished and motivated.

1. **Task Validation (AI Feature for the Future)**:

As a user, I want the system to validate that I've completed a task (e.g., by answering questions or scanning images) so that the rewards feel meaningful.

1. **Local Data Storage**:

As a user, I want my tasks and streak data saved locally so that I can access them offline and keep track of my progress.

### Overview of File Handling in Python

1. **Opening Files**: Python provides the open() function, which allows you to open files in different modes:
   * "r": Read (default mode).
   * "w": Write (overwrites the file if it exists).
   * "x": Create a new file; raises an error if the file exists.
   * "a": Append to the file.
   * "b": Binary mode.
   * "t": Text mode (default).
   * Combine modes like "rb" for binary read.
2. **Reading Files**:
   * read(): Reads the entire file.
   * readline(): Reads one line at a time.
   * readlines(): Reads all lines and returns them as a list.
3. **Writing to Files**:
   * write(): Writes a string to the file.
   * writelines(): Writes a list of strings to the file.
4. **Closing Files**: Always close files with close() or use a with statement to manage files efficiently and avoid resource leaks.

### Recommended Resources

1. **W3Schools** offers an easy-to-follow tutorial covering the basics of file handling with examples on reading, writing, and appending to files. [Learn more on W3Schools](https://www.w3schools.com/python/python_file_handling.asp)​

[W3Schools.com](https://www.w3schools.com/python/python_file_handling.asp)

.

1. **FreeCodeCamp** has a detailed guide explaining file handling with examples of how to use read(), write(), and advanced concepts like seeking positions within files. Visit FreeCodeCamp's tutorial​

[FreeCodeCamp](https://www.freecodecamp.org/news/file-handling-in-python/)

.

1. **Programiz** is another beginner-friendly resource with clear explanations of different file handling modes and operations. Check out Programiz​

[Pynative](https://pynative.com/python/file-handling/)

.

1. **Real Python** offers a comprehensive tutorial with advanced topics like working with JSON files and file handling best practices. Explore Real PythonksforGeeks\*\* covers a wide range of examples, including file manipulation, error handling during file operations, and file attributes. Read the GeeksforGeeks guide.

### JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) is a lightweight data-interchange format widely used for storing and exchanging data between systems, applications, and APIs. It is easy for humans to read and write and simple for machines to parse and generate. JSON is language-independent but uses conventions familiar to programmers of languages like C, C++, Java, Python, and JavaScript.

### Key Features of JSON

1. **Lightweight and Compact**: Unlike XML, JSON uses minimal formatting, making it more concise.
2. **Readable**: Its structure, similar to JavaScript objects, makes it easy for humans to understand.
3. **Easy to Parse**: Most programming languages have libraries or built-in support to parse JSON into objects or data structures.
4. **Data Representation**: Ideal for representing hierarchical data such as configurations or API responses.

### JSON Structure

JSON represents data as key-value pairs and supports:

* **Objects**: Encapsulated within {}, where each key is a string, and the value can be a string, number, boolean, array, object, or null.

json

Copy code

{

"name": "John",

"age": 30,

"isStudent": false

}

* **Arrays**: Ordered collections of values, enclosed in [].

json

Copy code

["Python", "JavaScript", "C++"]

### Use Cases

1. **Web APIs**: JSON is the standard for transmitting data between a client and server (e.g., RESTful APIs).
2. **Configurations**: Used for configuration files due to its simplicity (e.g., .eslintrc files in JavaScript projects).
3. **Data Storage**: Common in NoSQL databases like MongoDB.

### JSON vs Other Formats

* **JSON vs XML**: JSON is more lightweight and easier to read/write compared to XML.
* **JSON vs YAML**: YAML is more human-friendly but less strict and robust in parsing than JSON.

### Python and JSON

In Python, the json module provides functions to work with JSON:

* json.dump() and json.dumps() to convert Python objects to JSON.
* json.load() and json.loads() to parse JSON into Python objects.

Example:

python

Copy code

import json

# Convert Python dictionary to JSON

data = {"name": "Alice", "age": 25, "isStudent": True}

json\_string = json.dumps(data)

print(json\_string)

# Parse JSON back to Python dictionary

parsed\_data = json.loads(json\_string)

print(parsed\_data)

### Resources for Learning JSON

1. [JSON Official Website](https://www.json.org)
2. [Python JSON Module Documentation](https://docs.python.org/3/library/json.html)
3. w3schools JSON Tutorial
4. [MDN Web Docs on JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON)

 **JSON in Web Development**:

* What is JSON in API

 **JSON in Embedded Systems**:

* JSON for Embedded Systems and IoT

Let me know if you’d like guidance on working with JSON in Python!

# **Book Summary :**

## Chapter 1 :

Agile is method of planning used to divide project to small chunks of project each one of them will have a certain plan and undergo certain development cycle to convey information to customer and adapt to change that comes along the way reducing risk in developing the unwanted project or features and decreasing uncertainty by defining more and more about needed resources to achieve the project requirements , supporting better decision making on which feature to cancel and which one should be done.

## Chapter 2 :

Traditional planning was focused on activity rather than feature however features are considered the point of value to customer not the activity. There are three reasons why traditional planning fails: first of all, finishing project early, The individual will stretch their work by doing unnecessary tasks to avoid being assigned another work or expected to do work in shorter duration which can add pressure to them. Second reason is that late is passed down to the later tasks as most of activities are independent which is the third reason. Delay in finishing one task will cause all the other task to miss their schedule. There are multiple other reasons like multi-tasking which give illusion of speed however decrease time spent on adding value due to time consumed in switching between multiple tasks, priority assigned by developer may not be similar to that with respect to user and customers causing many feature to drop which was of value to customer , Ignoring uncertainty by believing that time assigned to project will a certain despite risk , changing features or adding new ones that may occur along the way and finally thinking that estimate is commitment which make customer think that if he was given a certain estimate then the project must be done in this time period.

## Chapter 3 :

1. **Core Values of Agile**:
   * **Individuals and Interactions**: Prioritizing a team’s collaboration and problem-solving over strict adherence to tools and processes. You grasped the essence here—team dynamics often matter more than tools.
   * **Working Software Over Documentation**: While documentation is important, it should not hinder progress toward a functional product. Iterative progress leads to better results.
   * **Customer Collaboration**: Agile treats the customer as a partner, aiming for shared goals instead of adversarial contract negotiation.
   * **Responding to Change**: Embracing uncertainties and adapting plans ensures the project remains relevant and realistic.
2. **How Agile Teams Work**:
   * **Work as One**: Agile breaks silos—each role (product owner, customer, developer, project manager, etc.) collaborates to align goals and work seamlessly.
   * **Short Timeboxed Iterations**: Timeboxing ensures that teams deliver incremental progress in a manageable way.
   * **Deliver Something**: Aiming for a shippable product or significant progress after every iteration keeps motivation high and helps assess usability early.
   * **Focus on Business Priorities**: Completing high-priority features in each release ensures value delivery.
   * **Inspect and Adapt**: Continuous learning and flexibility are critical as new insights or changes emerge.
3. **Agile Planning Layers**:
   * **Release Planning**: Focuses on defining the scope and timeline of a major deliverable (e.g., your first version of the streak tracker).
   * **Iteration Planning**: Breaks releases into smaller, actionable goals for each sprint (e.g., adding GUI elements in the next iteration).
   * **Daily Planning**: Keeps the team on track, identifying what needs to be done each day to achieve iteration goals.

## Chapter 4 :

**Story Points:**

* Story points measure the relative size and complexity of user stories in Agile projects.
* They are **not absolute units** but are used to estimate effort, complexity, and potential challenges in developing a feature.
* **Two approaches** to assign story points:
  1. Identify the smallest task, assign it 1, and compare other tasks relative to it.
  2. Choose a medium-sized task (e.g., 5 points) and estimate whether others are larger or smaller in comparison.

**Handling Unclear Requirements:**

* Agile often begins with incomplete requirements, making assumptions and guesses about user stories.
* Details are uncovered progressively during iterations.

**Velocity:**

* Velocity is the sum of story points completed in an iteration (typically 2–4 weeks).
* It helps estimate project duration by dividing total story points by the team's velocity.
* Since velocity can vary between teams, projects, and even within the same project, it must be recalibrated periodically to refine time estimates.

**Benefits of Story Points:**

* **Effort-based Estimation:** Story points focus on relative effort instead of absolute measures.
* **Separation from Time:** They decouple effort estimation from duration estimation, though effort and schedule remain interrelated.
* Provides a flexible, iterative way to estimate project timelines and adapt to changes as the project evolves.

A diagram of a diagram

Description automatically generated