**Agile Software Development Overview:**

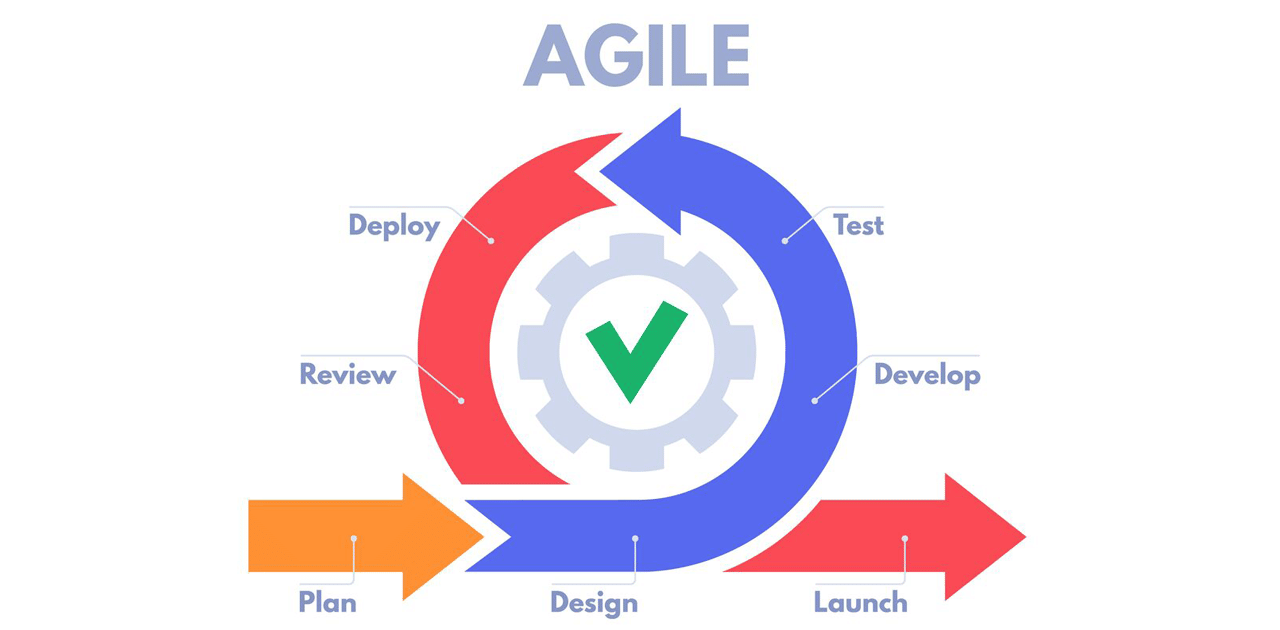
Agile is a flexible and iterative approach to software development that emphasizes collaboration, customer feedback, and adaptability. Key components include:

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 :

**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.

**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.