# Planning logic

**A Sprint** fixed period or duration in which a team works to complete a set of tasks

An **Epic** is a **big task or project** that is too large to complete in one sprint. It is broken down into **smaller tasks (stories)** that can be completed over multiple sprints.

A **Story** is a small task . It is part of an **Epic**.

A **Story Point** is a number that represents how much effort a story takes to complete. (usually in form of Fibonacci series)

1. Very Easy task
2. Easy task
3. Moderate task
5. Difficult task

**Sprint 1: (5 Days)**

Data Collection

Collection of Data **2**

Loading Data **1**

Data Preprocessing

Handling Missing Values **3**

Handling Categorical values **2**

**Sprint 2 (5 Days)**

Model Building

Model Building **5**

Testing Model **3**

Deployment

Working HTML Pages **3**

Flask deployment **5**

**Total Story Points**

Sprint 1 = 8

Sprint 2 = 16

Velocity= Total Story Points Completed/ Number of Sprints

Total story Points= 16+8 =24

The planning logic for the Solution Architecture Diagram involves a structured approach to designing and documenting the solution's architecture. This includes:

1. Define the scope and objectives of the solution.

2. Identify the key stakeholders and their requirements.

3. Develop a high-level overview of the solution's architecture.

4. Define the components, interfaces, and data flows.

5. Document the solution's architecture using standardized notation.

6. Review and refine the diagram with stakeholders.

7. Update the diagram as the solution evolves.

1. Business Requirements: Aligning the solution's architecture with business objectives and requirements.

2. Technical Constraints: Considering technical limitations, such as infrastructure, scalability, and security.

3. Stakeholder Needs: Incorporating feedback and requirements from stakeholders, including end-users, developers, and business leaders.

4. Industry Standards: Adhering to industry standards and best practices for solution architecture.

5. Flexibility and Scalability: Designing the solution's architecture to accommodate future growth and changes.

6. Risk Management: Identifying and mitigating potential risks and dependencies.

7. Cost-Benefit Analysis: Evaluating the costs and benefits of different architectural approaches.

Effective planning logic for the Solution Architecture Diagram also requires a iterative and incremental approach. This involves breaking down the architecture design process into smaller, manageable chunks, and continuously refining and updating the diagram as new information becomes available. By adopting an iterative approach, stakeholders can ensure that the Solution Architecture Diagram remains accurate, relevant, and effective in communicating the solution's architecture throughout the development lifecycle.

The planning logic involves considering various factors, including business requirements, technical constraints, stakeholder needs, industry standards, flexibility, scalability, risk management, and cost-benefit analysis. An iterative and incremental approach is also essential, breaking down the architecture design process into smaller chunks and continuously refining and updating the diagram.

By adopting a structured planning logic approach, stakeholders can ensure that the Solution Architecture Diagram is accurate, comprehensive, and effective in communicating the solution's architecture. This, in turn, enables the development of a solution that meets business requirements, is aligned with industry standards, and provides a strong foundation for future growth and development.

When developing the planning logic, several key considerations must be taken into account. These include business requirements and constraints, technical feasibility and limitations, stakeholder needs and expectations, industry standards and best practices, and risk management and mitigation. By carefully evaluating these factors, stakeholders can ensure that the Solution Architecture Diagram is well-informed and effective.

To ensure the success of the planning logic, several best practices should be followed. These include using standardized notation and terminology, documenting assumptions and dependencies, reviewing and refining the diagram regularly, collaborating with stakeholders and subject matter experts, and considering multiple scenarios and alternatives. By adopting these best practices, stakeholders can ensure that the Solution Architecture Diagram is a valuable tool for guiding the development of the solution.

The planning logic should also consider the scalability and maintainability of the solution, ensuring that it can adapt to changing business requirements and user needs over time. This involves designing a flexible and modular architecture that can be easily updated and expanded as needed.

# Project Planning Phase

## Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

| | |
|---|---|
| Date | 15 February 2025 |
| Team ID | |
| Project Name | |
| Maximum Marks | 5 Marks |

**Product Backlog, Sprint Schedule, and Estimation**

**Product Backlog:**

- A prioritized list of features, user stories, and requirements for the product

- Managed by the Product Owner, who ensures it is up-to-date and refined

- Contains items that are estimated, prioritized, and refined regularly

- Used to guide the development team's work during sprints

**Sprint Schedule**:

- A time-boxed period (usually 1-4 weeks) during which the development team works on a specific set of tasks

- Begins with Sprint Planning, where the team commits to delivering a set of work items

- Ends with Sprint Review and Retrospective, where the team reviews progress and identifies improvements

- Provides a regular cadence for the team to deliver working software

**Estimation:**

- The process of assigning a relative level of effort or complexity to each item in the Product Backlog

- Used to help the team understand the scope of work and make informed decisions about prioritization

- Can be done using various techniques, such as Story Points, T -Shirt Sizes, or Hours

- Enables the team to plan and commit to delivering specific work items during a sprint

In Agile project management, the Product Backlog plays a crucial role in guiding the development team's work. It's a prioritized list of features, user stories, and requirements for the product, managed by the Product Owner. The Product Backlog is continuously refined and updated to ensure it remains relevant and accurate. This list serves as the foundation for the development team's work during sprints, providing a clear understanding of the tasks to be completed.

A Sprint Schedule is a time-boxed period, typically ranging from one to four weeks, during which the development team focuses on a specific set of tasks. Each sprint begins with Sprint Planning, where the team commits to delivering a set of work items from the Product Backlog. At the end of the sprint, the team conducts a Sprint Review and Retrospective, reviewing progress and identifying areas for improvement. This regular cadence enables the team to deliver working software and make steady progress toward the project's objectives.

Estimation is a critical process in Agile project management, allowing the team to assign a relative level of effort or complexity to each item in the Product Backlog. This process helps the team understand the scope of work and make informed decisions about prioritization. Various estimation techniques can be employed, such as Story Points, T-Shirt Sizes, or Hours. By estimating the effort r required for each task, the team can plan and commit to delivering specific work items during a sprint, ensuring a smooth and efficient development process.

**Product Backlog, Sprint Schedule, and Estimation (4 Marks)**

Use the below template to create product backlog and sprint schedule

| Sprint | Functional Requirement (Epic) | User Story Number | User Story / Task | Story Points | Priority | Team Members |
|---|---|---|---|---|---|---|
| Sprint-1 | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | 2 | High | |
| Sprint-1 | | USN-2 | As a user, I will receive confirmation email once I have registered for the application | 1 | High | |
| Sprint-2 | | USN-3 | As a user, I can register for the application through Facebook | 2 | Low | |
| Sprint-1 | | USN-4 | As a user, I can register for the application through Gmail | 2 | Medium | |
| Sprint-1 | Login | USN-5 | As a user, I can log into the application by entering email & password | 1 | High | |
| | Dashboard | | | | | |
| | | | | | | |
| | | | | | | |

**Project Tracker, Velocity & Burndown Chart: (4 Marks)**

| Sprint | Total Story Points | Duration | Sprint Start Date | Sprint End Date (Planned) | Story Points Completed (as on Planned End Date) | Sprint Release Date (Actual) |
|---|---|---|---|---|---|---|
| Sprint-1 | 20 | 6 Days | 24 Oct 2022 | 29 Oct 2022 | 20 | 29 Oct 2022 |
| Sprint-2 | 20 | 6 Days | 31 Oct 2022 | 05 Nov 2022 | | |
| Sprint-3 | 20 | 6 Days | 07 Nov 2022 | 12 Nov 2022 | | |
| Sprint-4 | 20 | 6 Days | 14 Nov 2022 | 19 Nov 2022 | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**Velocity:**

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

**Burndown Chart:**

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

**https://www.visual-paradigm.com/scrum/scrum-burndown-chart/**
**https://www.atlassian.com/agile/tutorials/burndown-charts**

**Reference:**
**https://www.atlassian.com/agile/project-management**
**https://www.atlassian.com/agile/tutorials/how-to-do-scrum-with-jira-software**
**https://www.atlassian.com/agile/tutorials/epics**
**https://www.atlassian.com/agile/tutorials/sprints**
**https://www.atlassian.com/agile/project-management/estimation**
**https://www.atlassian.com/agile/tutorials/burndown-charts**

In conclusion, the Product Backlog, Sprint Schedule, and Estimation are essential components of Agile project management. The Product Backlog provides a clear understanding of the project's requirements, while the Sprint Schedule enables the team to deliver working software in regular increments. Estimation allows the team to plan and commit to delivering specific work items during a sprint. By integrating these components, Agile teams can ensure a structured and efficient development process, ultimately delivering high-quality products that meet customer needs.

- Burndown charts to track progress and identify trends
- Velocity metrics to measure team productivity
- Retrospectives to identify areas for improvement
- Continuous Integration and Continuous Deployment (CI/CD) pipelines to automate testing and deployment
- Collaboration tools, such as Slack or Trello, to facilitate communication and task management

Additionally, Agile teams can also benefit from adopting various best practices, such as:

- Test-Driven Development (TDD) to ensure high-quality code
- Pair programming to promote knowledge sharing and code review
- Code refactoring to maintain a clean and efficient codebase
- Continuous learning and professional development to stay up-to-date with industry trends and best practices

By incorporating these tools, techniques, and best practices into their Agile development process, teams can further enhance their productivity, efficiency, and overall quality of deliverables.

- Scrum: A framework for managing and completing complex projects using iterative and incremental practices.

- Kanban: A visual system for managing work, emphasizing continuous flow and limiting work in progress.

- Lean: A methodology focused on eliminating waste and maximizing value for customers.

- Extreme Programming (XP): A software development methodology emphasizing technical practices such as pair programming, continuous integration, and refactoring.

Additionally, Agile teams can also leverage various metrics and benchmarks to measure their performance and progress, such as:

- Cycle time: The time it takes for a feature or user story to go from concept to delivery.

- Lead time: The time it takes for a feature or user story to go from concept to delivery, including any delays or wait times.

- Deployment frequency: The frequency at which the team deploys new code or features to production