What is Project Evaluation?

Project evaluation is a systematic process of assessing a project's performance and outcomes against its original goals and criteria.

It's not just a simple review; it's a formal, objective analysis that happens at various stages of a project's life cycle. The main goal is to collect and analyse data to understand what worked, what didn't, and why.

Benefits and Importance

1. **Informed Decision-Making:**
   Evaluation provides **data-driven insights** that help managers and stakeholders make better, more timely decisions. This could involve adjusting the project scope, reallocating resources, or even deciding to terminate a project that isn't meeting its goals.

2. **Continuous Improvement:**
   By identifying both successes and failures, project evaluation helps an organisation learn from its experiences. It allows teams to refine their processes, implement best practices, and avoid repeating mistakes in the future, fostering a culture of continuous learning.

3. **Accountability and Transparency:**
   It establishes clear metrics and criteria for success, promoting accountability among team members and stakeholders. By openly reporting on a project's performance, evaluation builds trust and ensures that resources are being used effectively.

4. **Risk Reduction:**
   Regular evaluations throughout the project lifecycle help to identify and mitigate potential risks early on. This proactive approach prevents small issues from becoming major problems, increasing the likelihood of a successful outcome and preventing costly delays.

5. **Enhanced Stakeholder Satisfaction:**
   By involving stakeholders in the evaluation process, their needs and expectations are more clearly understood and addressed. This leads to a more tailored final product, increased stakeholder buy-in, and greater satisfaction with the project's outcomes.

Cost-Benefit Evaluation Techniques

Cost-benefit evaluation techniques are methods used to assess the economic viability of a project or decision by comparing its total costs to its total benefits. These techniques are essential in project management for making informed decisions and prioritising investments. The primary goal is to determine if the benefits outweigh the costs and to what extent.

In examples below, Cost = $10,000; Returns = $15,000; time(t) = 1 year

1. **Net Present Value (NPV)**
   - NPV is a core technique that accounts for the **time value of money**. It calculates the difference between the present value of benefits (*note that the benefit we typically calculate is the future value of benefits*) and the present value of costs over a specified period.
   - A project is considered economically viable if its NPV is **positive**, meaning the project's benefits are expected to exceed its costs in today's dollars. It helps compare multiple projects and select the one with the highest value.

Net Present Value (NPV)

The **NPV** formula calculates the total value of future cash flows in today's money. We use an assumed discount rate of 10% to account for the time value of money. A positive NPV suggests a profitable project.

- **Formula:** $NPV = (\frac{Return}{(1+DiscountRate)^t}) - Cost$
- **Calculation:** $NPV = (\frac{\$15{,}000}{(1+0.10)^1}) - \$10,000$
- **Result:** $NPV = \$13,636.36 - \$10,000 = \$3,636.36$

2. **Benefit-Cost Ratio (BCR)**
   - The BCR is a ratio that compares the present value of a project's benefits to the present value of its costs.
   - A project is considered worthwhile if the BCR is greater than **1.0**. A ratio of 1.5, for example, means the project is expected to generate $1.50 in benefits for every $1.00 spent. This technique is often used to quickly compare the returns of different projects

Benefit-Cost Ratio (BCR)

The **BCR** compares the present value of the benefits to the present value of the costs. A BCR greater than 1.0 indicates that the benefits outweigh the costs.

- **Formula:** $BCR = \frac{PresentValueofBenefits}{PresentValueofCosts}$
- **Calculation:** $BCR = \frac{(\$15{,}000/(1+0.10))}{\$10{,}000}$
- **Result:** $BCR = \frac{\$13{,}636.36}{\$10{,}000} = 1.36$

.

3. **Return on Investment (ROI) ⟶ Profit %**
   - ROI is a straightforward percentage that measures the gain or loss generated by an investment relative to its cost.
   - It's calculated by subtracting the cost of the investment from the gain and dividing the result by the cost. While it doesn't account for the time value of money, ROI is a simple and widely used metric for evaluating the profitability of a project.

Return on Investment (ROI)

ROI is a simple percentage that shows the profitability of an investment. It is calculated as the net profit divided by the cost of the investment.

- **Formula:** $ROI = (\frac{NetProfit}{Cost}) \times 100\%$
- **Calculation:** $ROI = (\frac{\$15{,}000 - \$10{,}000}{\$10{,}000}) \times 100\%$
- **Result:** $ROI = (\frac{\$5{,}000}{\$10{,}000}) \times 100\% = 50\%$

4. **Payback Period**
    - The payback period is the length of time it takes for a project's cumulative benefits to equal its initial investment cost.
    - This technique focuses on liquidity and risk. Shorter payback periods are generally preferred as they indicate a faster recovery of the initial investment, which can be seen as less risky.

Payback Period

The **payback period** is the time it takes for the initial investment to be recovered from the project's cash inflows.

- **Formula:** $PaybackPeriod = \frac{InitialInvestment}{AnnualCashInflow}$
- **Calculation:** $PaybackPeriod = \frac{\$10{,}000}{\$15{,}000}$
- **Result:** $PaybackPeriod = 0.67$ years (approximately 8 months)

The Process Improvement Cycle

The Process Improvement Cycle is a continuous, iterative approach to making processes more efficient and effective. It's often represented by the **Plan-Do-Check-Act (PDCA) cycle** or a similar model.

- **Plan:** Identify an opportunity for improvement and plan a change. This involves analyzing the current process, setting goals, and defining the steps needed to make the change.
- **Do:** Implement the planned changes on a small scale. This step is about testing the plan in a controlled environment to see its effects without disrupting the entire operation.
- **Check:** Measure the results of the change and compare them to the original goals. This is where you analyze the data collected during the "Do" phase to see if the change worked as intended.
- **Act:** If the change was successful, implement it on a broader scale. If it wasn't, go back to the "Plan" phase and refine the approach based on what you learned.

Process Change: The Process Change Process

The **Process Change Process** is the structured way to implement a new or modified process. It's a key part of change management and typically involves the following steps:

1. **Preparation:** Define the scope of the change, identify stakeholders, and create a communication plan. It's crucial to build a strong case for why the change is necessary.
2. **Implementation:** Roll out the new process, which may include training employees, updating documentation, and providing new tools. This can be done incrementally or all at once, depending on the scale of the change.
3. **Monitoring:** Track the new process using metrics defined in the GQM paradigm to ensure it's achieving the desired results. This step validates the success of the change and helps identify any new problems.
4. **Sustainment:** Integrate the new process into the organization's daily operations. This involves continuous review and refinement to ensure the improvements are maintained over the long term.

The GQM Paradigm (Process Measurement)

The **Goal-Question-Metric (GQM)** paradigm is a top-down, structured approach for defining and implementing a software measurement program. It helps ensure that you're not just collecting data, but that you're gathering **meaningful information** directly tied to your project or business goals.

The GQM paradigm works by creating a clear, hierarchical link between an organisation's high-level goals and the specific data that needs to be collected. This process has three distinct levels:

1. Goals (Conceptual Level)

This is the highest level of the paradigm. Goals are a clear, concise statement of what you want to achieve or improve. They define the purpose of the measurement program from the perspective of the people who will use the data.

Example Goal: "Evaluate the effectiveness of our new agile development process from the project manager's viewpoint to improve project delivery speed."

2. Questions (Operational Level)

At this level, you break down the high-level goal into a set of specific, actionable **questions**. These questions are designed to provide a deeper understanding of the goal and define what information is needed to achieve it. They bridge the gap between the abstract goal and the concrete data.

- **Questions are not metrics themselves.** They are inquiries that help you decide which metrics to use.
- **They should be specific and measurable,** even if they don't have a numerical answer yet.

Example Questions (for the goal above):

- How does the new process impact our feature delivery time?
- What is the rate of defects found per sprint?
- How satisfied are the developers with the new process?

3. Metrics (Quantitative Level)

This is the lowest and most concrete level. **Metrics** are the actual, quantifiable data points that are collected to answer the questions. They can be objective (e.g., lines of code, number of bugs) or subjective (e.g., a satisfaction rating on a scale of 1-5).

- Each question from the previous level should be tied to at least one metric.

Example Metrics (for the questions above):

- **For delivery time:** Lead time, Cycle time.
- **For defect rate:** Number of defects discovered, Defect density (defects per KLOC).
- **For developer satisfaction:** Results from a post-sprint survey (average satisfaction score).

*The GQM paradigm follows a **top-down** approach for definition (from goals to metrics) and a **bottom-up** approach for interpretation (using metrics to answer questions and evaluate goals). This ensures that every piece of data collected has a clear purpose and directly contributes to a strategic objective.*

Process Analysis

- Process analysis is a systematic examination of a business or software process to understand how it works, identify inefficiencies, and pinpoint opportunities for improvement.
- The goal is to optimise the process to make it more efficient, cost-effective, and aligned with organisational goals.
- This is a crucial step in any process improvement initiative, whether it's aimed at reducing waste, increasing productivity, or enhancing quality.

Flowcharting

Flowcharting is the visual diagramming of a process's steps in a sequential order. It helps quickly identify bottlenecks and redundant or unnecessary steps by providing a clear, top-down view of how a process actually works.

Root Cause Analysis

This is a systematic method for finding the underlying cause of a problem, rather than just treating the symptoms. Techniques like the "5 Whys" (repeatedly asking "why?") or a Fishbone diagram are used to dig deeper and prevent the problem from reoccurring.

Value Stream Mapping

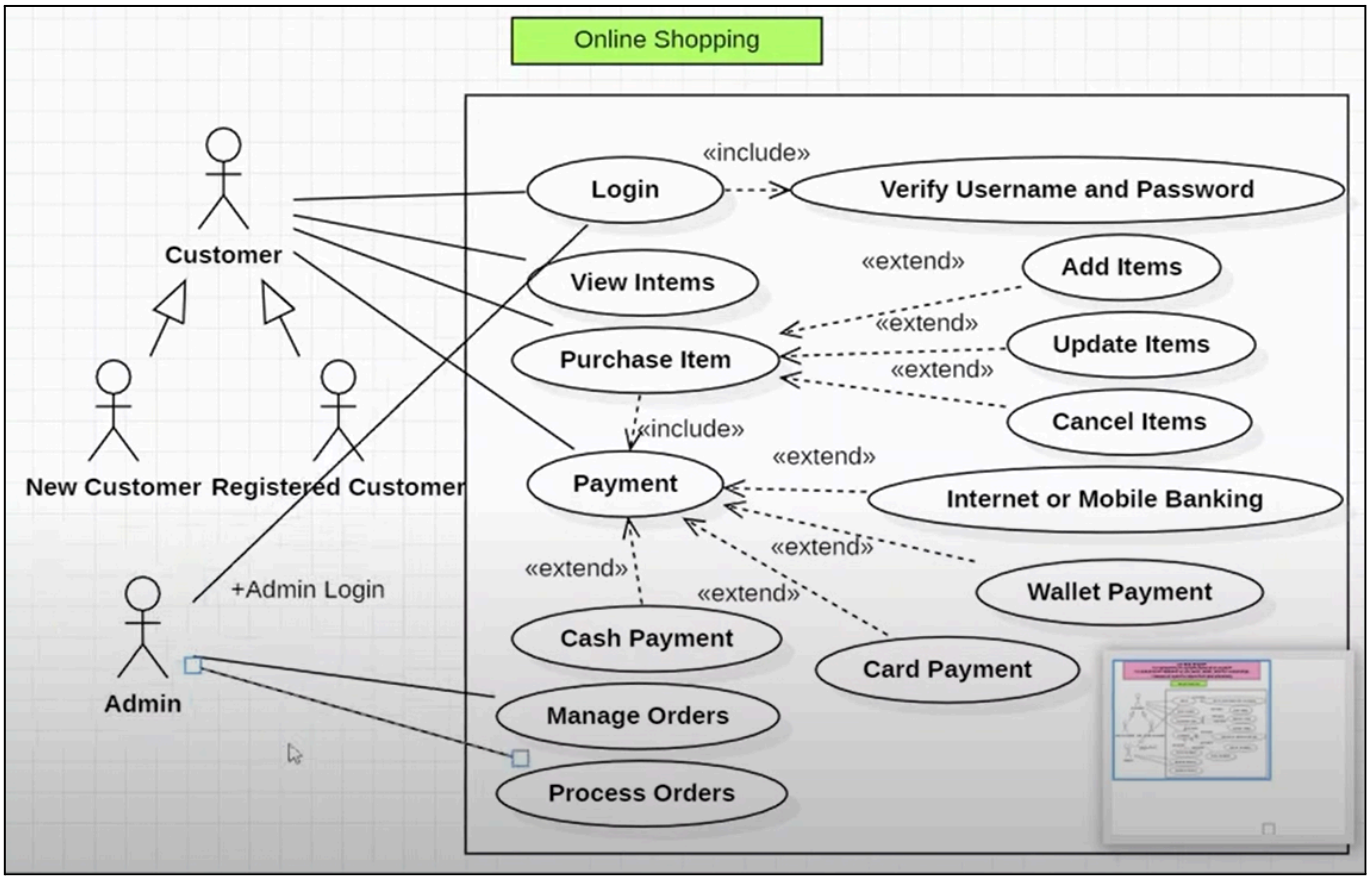
Value Stream Mapping (VSM) is a lean technique that visualizes the flow of materials and information to deliver a product or service. Its main purpose is to identify and eliminate "waste" in the process, such as delays and non-value-adding steps, to improve efficiency.

Components of a Use Case Diagram

A use case diagram provides a high-level view of a system's functionality and how users interact with it. Here are its main components:

- ***Actor:*** *An actor is a user, external system, or a person who interacts with the system to perform a function.*
- ***Use Case:*** *A use case represents a specific function or a set of actions that a system performs to produce an observable result of value to an actor.*
- ***System Boundary:*** *This is a box or rectangle that defines the scope of the system and separates the use cases from the actors.*
- ***Association:*** *A simple line connecting an actor to a use case, showing that the actor interacts with that use case.*
- ***Include Relationship:*** *A dashed arrow from a base use case to an included one, indicating that the base use case's behavior always includes the behavior of the other.*
- ***Extend Relationship:*** *A dashed arrow from an extending use case to a base one, showing that the extending use case's behavior may optionally be added to the base use case's behavior under certain conditions.*
- ***Generalization:*** *A solid line with a hollow arrowhead, indicating that one actor or use case is a more specific version of a general one.*

ONLINE SHOPPING



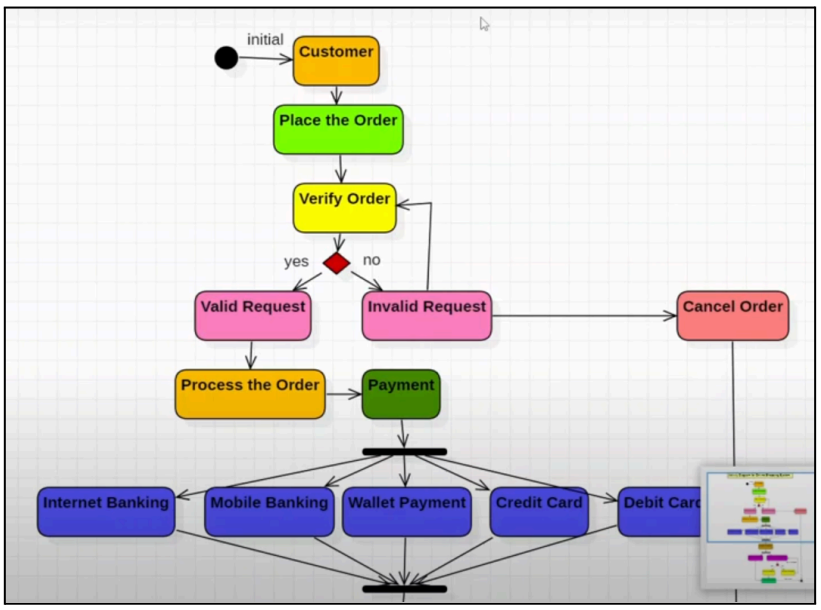
Components of an Activity Diagram

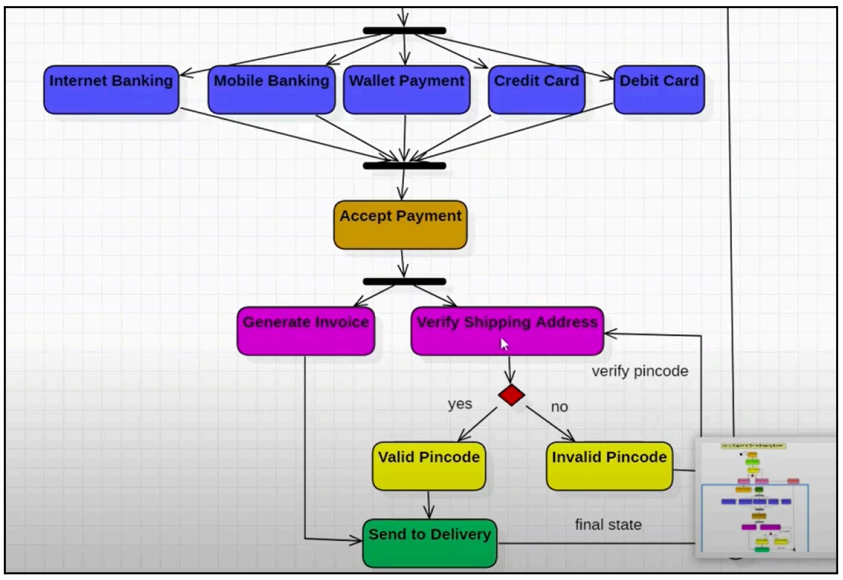
An activity diagram is a behavioral UML diagram that visualizes the flow of control and data in a system. Here are its key components:

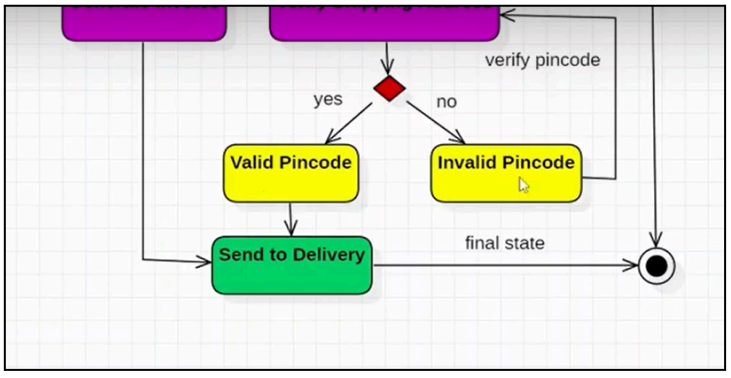
- ***Initial Node:*** *The starting point of the activity flow.*
- ***Activity/Action:*** *A single step or task performed by the system or user.*
- ***Final Node:*** *The ending point of the activity flow.*
- ***Control Flow (Edge):*** *An arrow connecting two nodes, representing the flow from one action to the next.*
- ***Decision Node:*** *A diamond shape that represents a point where a choice is made, leading to different paths.*
- ***Merge Node:*** *A diamond shape that joins multiple paths back into a single flow after a decision.*

- ***Fork Node:*** *A thick horizontal or vertical bar that represents the split of a single flow into multiple, concurrent flows.*
- ***Join Node:*** *A thick horizontal or vertical bar that synchronizes multiple concurrent flows before continuing as one.*

ONLINE SHOPPING



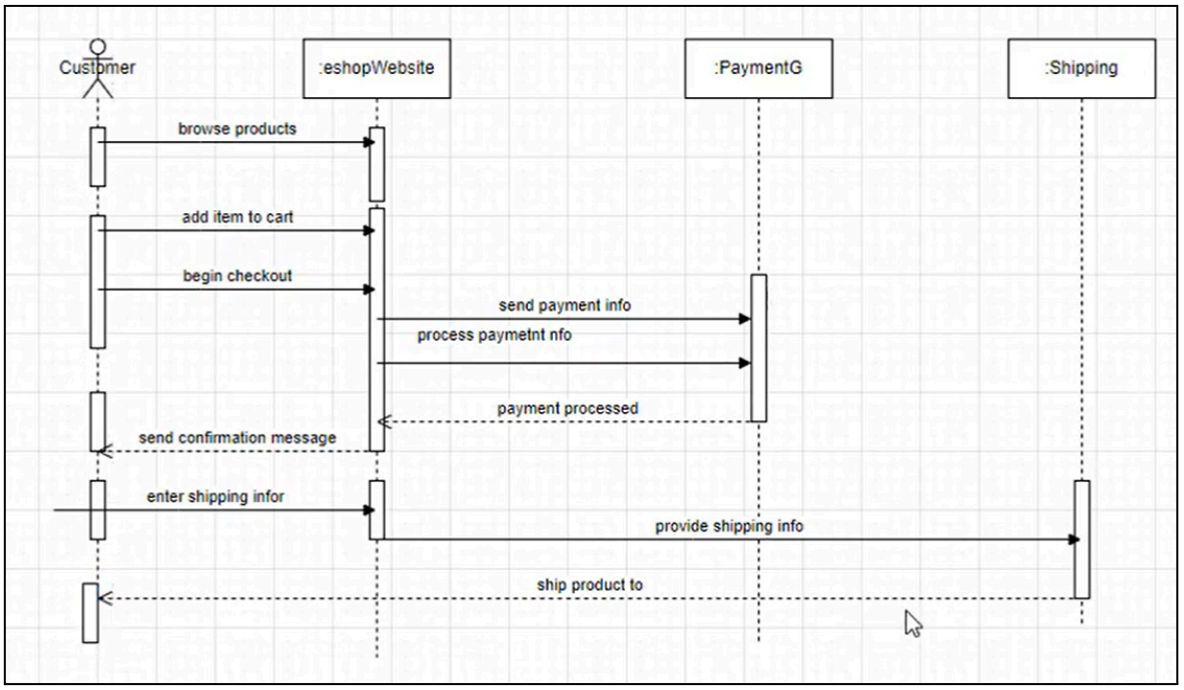




Components of a Sequence Diagram

A sequence diagram is a behavioral UML diagram that shows how objects interact with each other in a sequential manner over time. Here are its key components:

- ***Lifeline:** A vertical dashed line representing an object's existence and its participation in the interaction over time.*
- ***Actor:** A person or external system that initiates the interaction with the objects.*
- ***Object/Class:** A named rectangle at the top of a lifeline representing an instance of a class or a specific object.*
- ***Activation Bar:** A thin, vertical rectangle on a lifeline showing the period during which an object is performing an action or is active.*
- ***Message:** An arrow between two lifelines, representing communication or a call from one object to another.*
- ***Return Message:** A dashed arrow showing the return of a value from a called object to the caller.*



Relations in a Class Diagram

*A class diagram is a **structural UML diagram** that shows the static structure of a system. It represents the classes, interfaces, and their relationships, such as association, inheritance, and composition. Think of it as a blueprint of the system's architecture, revealing the classes that make up the system and how they interact with each other without showing their behavior.*

1. Association

*An **association** is a general relationship between two classes, indicating that they are connected or interact in some way. It can be a simple uses relationship.*

***Representation:** A solid line connecting the two classes. An arrowhead can be added to show the direction of the relationship (unidirectional). The lack of an arrowhead implies a bidirectional relationship.*

2. Aggregation

***Aggregation** is a specific type of association that represents a "has-a" relationship, where one class is composed of one or more instances of another. The key is that the component parts can exist independently of the whole.*

Representation: *A solid line with a* ***hollow diamond*** *on the side of the "whole" or container class.*

3. Composition

Composition *is a stronger form of aggregation that also represents a "has-a" relationship, but with a crucial difference: the component parts cannot exist without the whole. If the whole is destroyed, its parts are also destroyed.*

Representation: *A solid line with a* ***solid (filled) diamond*** *on the side of the "whole" or container class.*

4. Generalization (Inheritance)

Generalization *represents an "is-a" relationship, where one class (the subclass) is a more specific version of another (the superclass) and inherits its attributes and operations.*

Representation: *A solid line with a* ***hollow arrowhead*** *pointing from the subclass to the superclass.*

5. Dependency

A ***dependency*** *is a very weak relationship where one class depends on another, meaning a change in one class might affect the other, but not necessarily vice versa. This often occurs when a class uses another class's object as a parameter in a method.*

Representation: *A dashed line with an* ***open arrowhead*** *pointing from the dependent class to the class it depends on.*