Software Engineering

Unit 1 - Systems, Emergent Properties, Modelling, UML component diagrams

- Software Engineering
- Systems
- Emergent Properties
- Models
- Unified Modelling Language (UML) - *From Year 2 DSA*
- UML Component Diagrams

Unit 2 - Software lifecycle, development processes, agile and UML activity diagrams

- Software Development Lifecycle
- Iterative vs Incremental Development
- Waterfall
- Cyclic
- Unified Process
- Agile
- Scrum
- UML Activity Diagrams

File Software Engineering.md

> Software Engineering

Software Engineering

- The systematic procedures used for the analysis, design, implementation, testing and maintenance of software
- Term created by Margaret Hamilton
- Solves **software crisis** (failures due to increasing demands and low expectations)
- Standardised, structured, thorough approach to writing code
- Formal
- Increases quality of development

File Systems.md

Systems

Systems

- **A collection of components, modules and services that work together to perform a specific task or function**

Systems Thinking

- The approach to solving problems by thinking of how a systems components work together, how they relate to each other and how they change over time.
- A bicycle is a system
    - Multiple Components

    - Work together to make the user travel in a direction
- A pile of papers is not a system
    - Has multiple components
    - Does not have a common goal
- A tree is a system
    - Multiple components
    - Works together to live

Characteristics of Systems

These are fundamental parts of systems

- **Central Objective**
    - All of the components work together to achieve a common goal
- **Integration**
    - The components are linked together to complete the system
- **Interaction**
    - The components interact with each other to achieve the common goal
- **Interdependence**
    - The components are dependant on one another
- **Organisation**
    - There usually is a hierarchy among components

Components

These are the different components found in a system with examples

1. **Input, Processes, Output**
    1. **Input** - Data entry
    2. **Processes** - Check if data valid
    3. **Output** - Display result to user
2. **Environment**
    1. **Internal** - exists within the system
    2. **External** - exists outside of the system
3. **Boundary**
    1. Separates components from external entities
4. **Interface**
    1. A way to interact with the system, like a keyboard or APIs
5. **Feedback and Control**
    1. Feedback is the measured outputs of the system
    2. Control is the adjustment made to the system based on the feedback. Usually used to maintain a set amount of conditions

Hierarchy

- Complex systems are composed of many **subsystems**
- Subsystems are arranged into hierarchies and are integrated so they can achieve the common goal
- Subsystems also have their own boundaries, their own boundaries and their own environment

File Emergent Properties.md

Emergent Properties

Emergent Properties

Some properties only *emerge* when the system components are all integrated. These properties are called **emerging properties**.
Emergent properties come in two flavours:

- **Functional**
  - Emerges when all the individual parts of a system work together to achieve some objective
- **Non Functional**
  - Emerges as a result of the behaviour of the system in its environment

Examples

```
The motorbike weighs 180kg
```

This is a non-emergent property, as it is simply a sum of all the parts used to create a bike.

```
The motorbike can be ridden
```

This is an emergent property and is also functional. The ability of the motorbike to be ridden is not a direct sum of its individual components. It emerges when these

components (engine, wheels, brakes, etc.) work together in a specific way, allowing a person to ride the motorbike.

```
The lights of the bike can be turned on
```

This is also a functional emergent property. The capability of the bike's lights to be turned on is not inherent in any single component. It arises as a new property when the electrical system, switch, and bulbs are integrated, allowing the lights to function.

```
The ATM provides a secure service
```

This is a non functional emergent property. The security of the ATM service is not a direct sum of its individual components (keyboard, card reader, display, etc.). Instead, it emerges from the system's overall design, protocols, and software, providing a secure environment for transactions.

File Models.md

Models

System Models

- Models are used to propose, test or improve a system
- Contain components that make up a system
- Defines the types of relations between the components
- Defines how components relate into the whole system

- Shows us how the system interacts with its environment
- It's important to use the right level of detail

Model uses

- **Abstraction** - Allows focus on the important features
- **Representation** - Turn ideas into an actual thing
- **Communication** - Helps sharing different ideas
- **Early Evaluation** - Check if requirements are met

vs Modelled Systems

- Models are quicker and cheaper to build
- Can choose details to include or drop in models
- Models can sometimes be used in simulated environments
- Models can evolve

File Unified Modelling Lnaguages(UML).md

Unified Modelling Language (UML)

Unified Modelling Language (UML)

- UML is a notation commonly used for software design
- Very flexible and can represent a wide variety of things using diagrams

Class Diagrams

- A class diagram created with UML describes the **overall** design of a system for **all** situation that might occur.
- The diagrams can describe the different relations between the different classes

Things that might be included in a UML Class diagram;

- Classes used
- Relationships between classes
- Attributes and methods of classes
- Constraints

Visibility

There are different symbols that can represent the visibility of attributes and methods

Symbol	Meaning
+	Public
-	Private
#	Protected
-	Package

File UML Component Diagrams.md

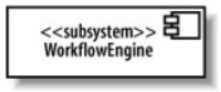
UML Component Diagrams

UML Component Diagrams

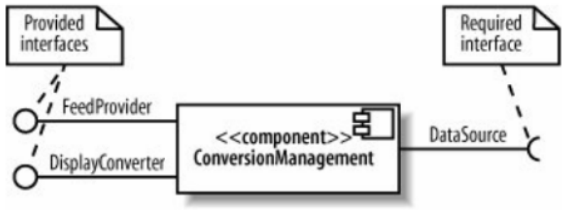
Component Diagrams are used to plan out a system and to easily explain the architecture of a system. It can also help manage the complexity and dependencies amongst components.

Component:

Subsystem:



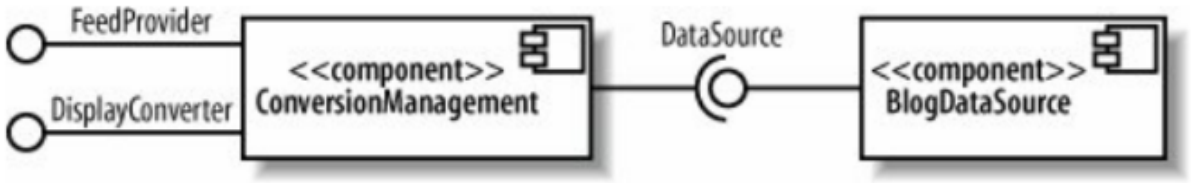
Component providers:



Dependency

An arrow between components denotes dependency

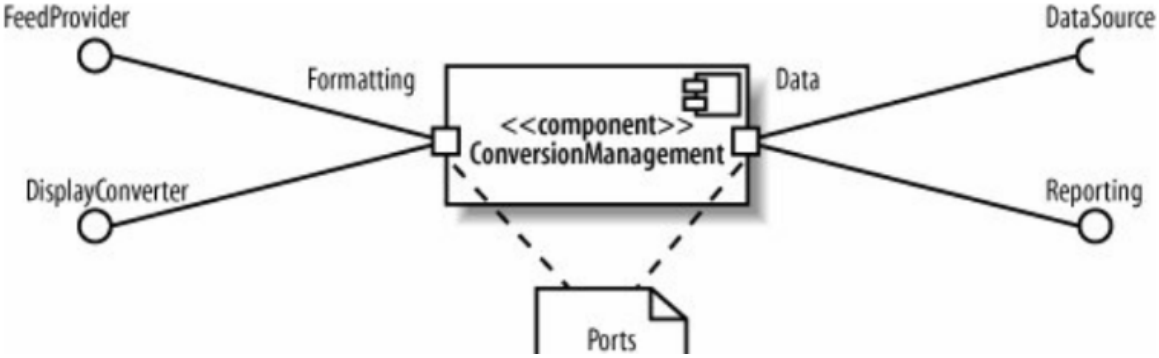
Interface



A ball socket connection denotes an interface
The half circle represents the interface being provided
The full circle represents the interface being required

Ports

A port is used to depict points of interaction between a component and the outside world



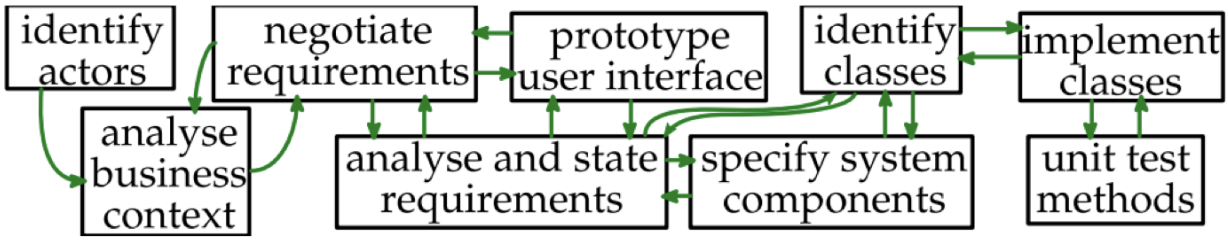
A square is used to represent a port to the outside world

File Software Development Lifecycle.md

Software Development Lifecycle

Software Development Lifecycle

- Defines phases we follow to make software
- *activities required to develop software*



- We use software development lifecycles because we need a logical progression of development activities
- By splitting parts into logical progressions, it's easier to keep track of what what to do now and what is required for the next stage

-

Specific stages of SDLC

Domain Analysis

- Research what is already taking place / what already exists for this category of products
- Literature review
- Speaking to experts

Specifications

- Also known as **Requirements Engineering**
- Check if solutions are feasible
- Get input from stakeholders
- Analyse systems that may be used or already in use

Design

- Model the structures to be used in the product
- Creating prototypes of software
- Create specifications for interfaces that are going to be implemented
- Create a plan for deploying software

Implementation

- Write out source code
- Database creation
- Implement deployment plan

Validation

- Testing individual components
- Testing sub systems
- Testing whole system
- User testing

Deployment

- Deploy system completely
- Create manuals for users
- Train staff and create training material

File Iterative vs Incremental Development.md

Iterative vs Incremental Development

Iterative vs Incremental

Iterative

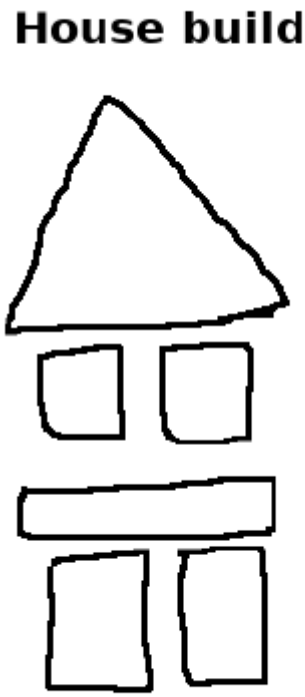
- Iterative development is when improvements are made with each cycle

Incremental

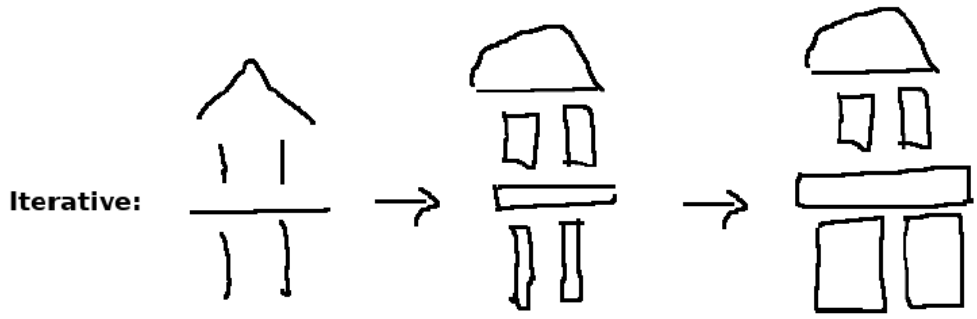
- Adding new features for each cycle

Example - House Building

We can imagine a company wanting to create a house.

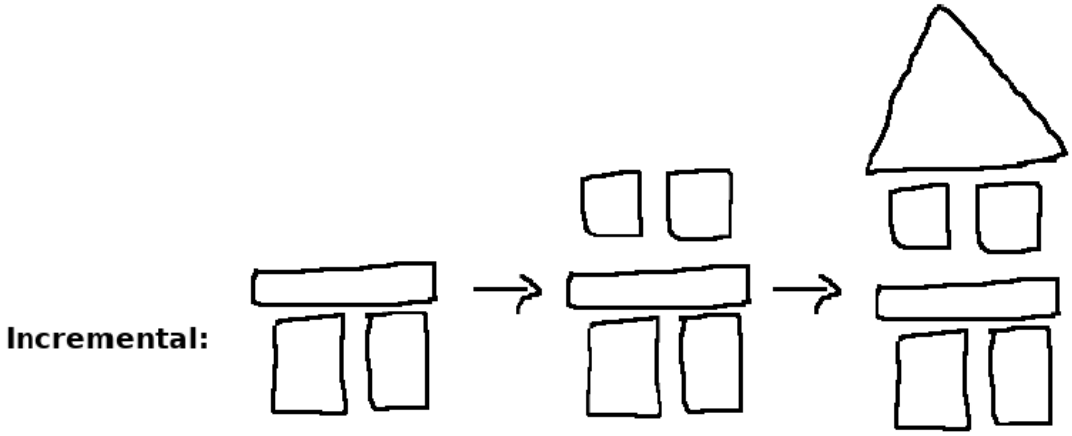


- **Iterative**
  - Would build a basic structure of whole house first
  - Then improve that structure next cycle
  - Then improve that structure further next cyle



- **Incremental**

  - Would start off with foundation features (e.g ground floor walls and roof)
  - Add new feature next cycle (1st floor walls)
  - Add another feature next cycle (1st floor roof)



File Waterfall.md

Waterfall

Waterfall Process

- Simplest way of organising software development
- No turning back usually

```
[Inception]
    -> [Requirements Analysis]
            -> [Systems Analysis]
                    -> [Design]
                            -> [Implementation]
                                    -> [Testing]
                                            -> [Deployment]
```

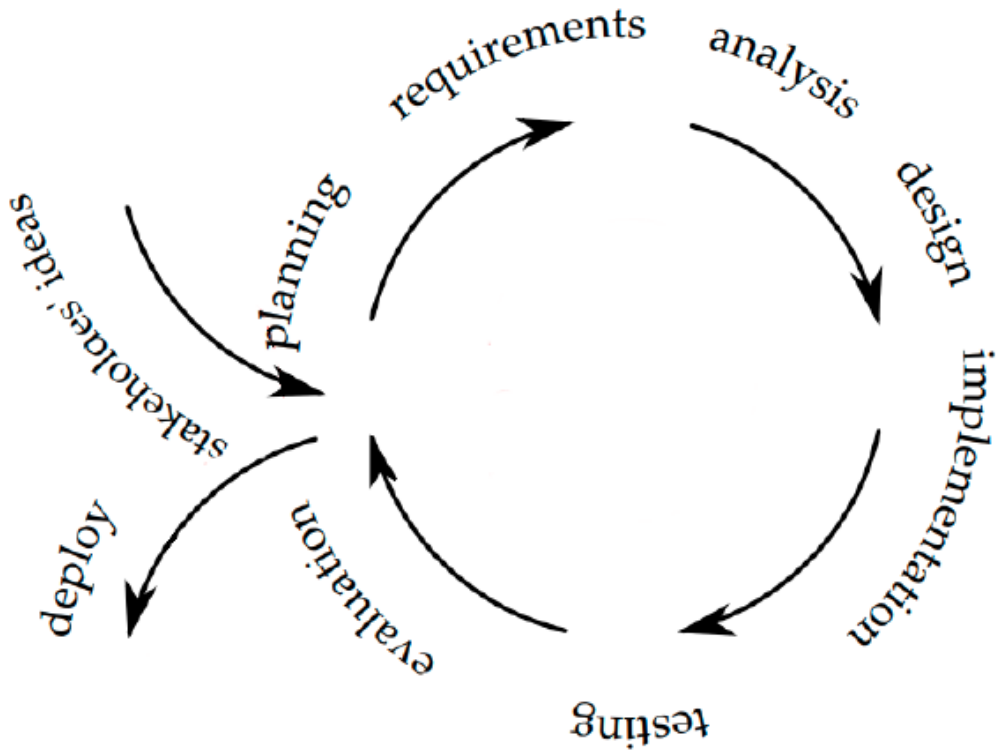
```
                                                                  ->
[Maintenance]
```

File Cyclic.md

Cyclic

Cyclic Process

- Cycles through the waterfall activities over and over
- Each cycle adds a new feature or makes improvements
- Allows for changes between cycles so can deploy stakeholders ideas



File Unified Process.md

Unified Process

Unified Process

The Unified Process is an iterative and incremental process framework that guides the development of software systems. It is based on the Unified Modeling Language (UML) and is designed to be adaptable to various project types and sizes.

Inception Phase

- **Objective:** Define the project scope and establish a solid foundation for the project.
- **Activities:**
  - Develop the project scope.
  - Engage stakeholders to determine goals and requirements.
  - Plan the product, considering costs and resources.
  - Produce a vision document and use case models to capture key features.

Elaboration Phase

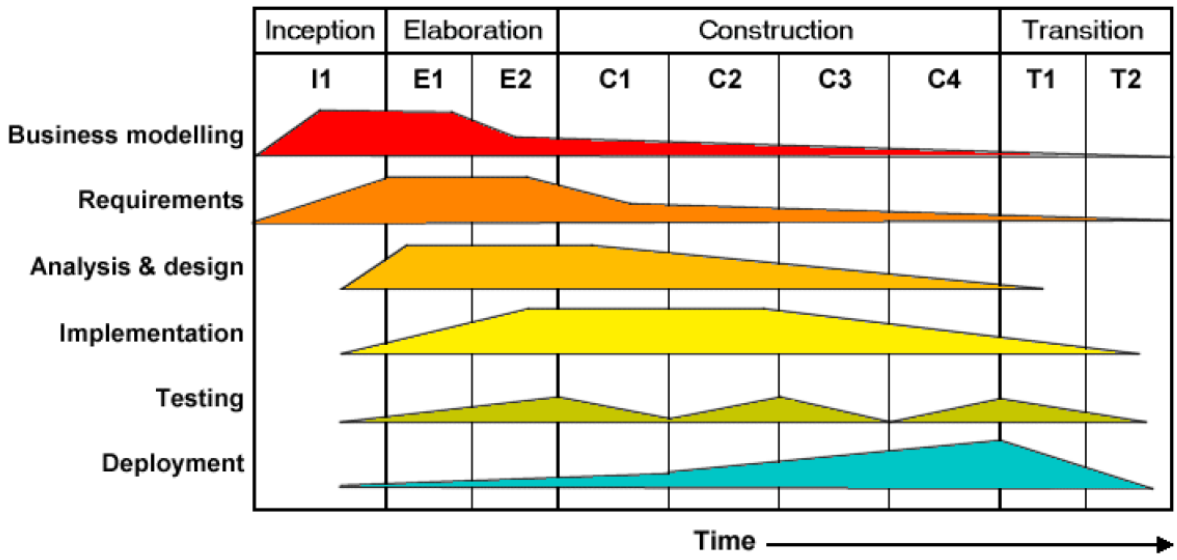
- **Objective:** Address potential issues and refine the project plan in preparation for development.
- **Activities:**
  - Analyse potential problems and risks.
  - Mitigate major risks identified.
  - Further refine the project plan based on insights gained.

Construction Phase

- **Objective:** Develop a robust and functional version of the software incrementally.
- **Activities:**
  - Create a beta version of the software.
  - Continuously meet stakeholders' needs through iterative development.
  - Conduct thorough testing and address issues identified.
  - Repeat the development-testing cycle until the product is ready for deployment.
  - Produce comprehensive documentation.

Transition Phase

- **Objective:** Deploy the software, ensure smooth operation, and address post-deployment concerns.
- **Activities:**
  - Deploy the software for use.
  - Conduct ongoing maintenance and address bug reports.
  - Provide training to end-users.
  - Monitor and manage the system's performance.



File Agile.md

Agile

Agile Methodology

Agile is an iterative and incremental approach to software development that prioritizes flexibility, collaboration, and customer satisfaction. It values responding to change over following a rigid plan and emphasizes delivering small, functional increments of software at regular intervals.

Agile Principles

Agile is guided by the following key principles:

1. **Individuals and Interactions over Processes and Tools:**
   - Emphasis on effective communication and collaboration within the development team.

2. **Working Software over Comprehensive Documentation:**
    - Prioritizes delivering functional software over extensive documentation.
3. **Customer Collaboration over Contract Negotiation:**
    - Encourages continuous collaboration with customers to adapt to changing needs.
4. **Responding to Change over Following a Plan:**
    - Agile embraces change and adjusts plans to accommodate evolving requirements.

Workflow

1. **Backlog Refinement:**
    - Continuously update and refine the backlog of features.
2. **Sprint Planning:**
    - Select features for the upcoming Sprint.
3. **Daily Standup:**
    - Brief, daily meetings for status updates and issue resolution.
4. **Sprint Review:**
    - Demonstrate the completed Increment to stakeholders.
5. **Sprint Retrospective:**
    - Reflect on the Sprint process for improvement.
6. **Repeat:**

- Iterate through the cycle, continuously adapting to changing requirements.

File Scrum.md

Scrum

Scrum Framework

Scrum is an Agile framework that facilitates iterative and incremental development. It provides a structured yet flexible approach to deliver high-value software, emphasizing collaboration, adaptability, and customer satisfaction.

Roles in Scrum

1. Product Owner:

- Represents the voice of the customer.
- Defines and prioritizes features in the product backlog.
- Works closely with the development team to ensure the product meets business goals.

2. Scrum Master:

- Acts as a servant-leader for the Scrum Team.
- Facilitates the Scrum process, removes impediments, and ensures the team adheres to Scrum principles.
- Encourages continuous improvement within the team.

3. Development Team:

- Cross-functional, self-organizing team responsible for delivering increments of product.
- Collaborates to determine how to best accomplish the work within a Sprint.
- Typically consists of developers, testers, and other necessary skills.

Scrum Artifacts

1. Product Backlog:

- An ordered list of features, enhancements, and bug fixes.
- Prioritized by the Product Owner based on business value.
- Evolves over time as requirements change.

2. Sprint Backlog:

- A subset of the Product Backlog selected for a specific Sprint.
- Determines the work to be done in the Sprint.
- Owned and managed by the Development Team.

3. Increment:

- The sum of all the completed and tested product backlog items at the end of a Sprint.
- Represents a potentially releasable product.

Scrum Events (Ceremonies)

1. Sprint Planning:

- A meeting at the beginning of each Sprint to decide which features will be worked on.
- Involves the Product Owner and the Development Team.

2. Daily Scrum (Standup):

- A short, daily meeting for the Development Team to discuss progress and plan for the day.
- Facilitated by the Scrum Master.

3. Sprint Review:

- A meeting at the end of each Sprint where the team presents the Increment to stakeholders.
- Stakeholders provide feedback, and the Product Owner updates priorities.

4. Sprint Retrospective:

- A meeting at the end of each Sprint for the team to reflect on the process and identify improvements.
- Led by the Scrum Master.

Scrum Workflow

1. **Product Backlog Refinement:**
    - Ongoing process to clarify and update the Product Backlog.

2. **Sprint Planning:**
    - Decide which items from the Product Backlog will be worked on during the Sprint.
3. **Daily Scrum:**
    - Brief daily meetings for status updates and issue resolution.
4. **Sprint Review:**
    - Demonstrate the completed Increment to stakeholders.
5. **Sprint Retrospective:**
    - Reflect on the Sprint process and identify improvements.
6. **Repeat:**
    - Iteratively go through the Scrum events for each Sprint.

Scrum's iterative approach, clear roles, and defined ceremonies make it a powerful framework for delivering high-quality software in a collaborative and adaptive manner.

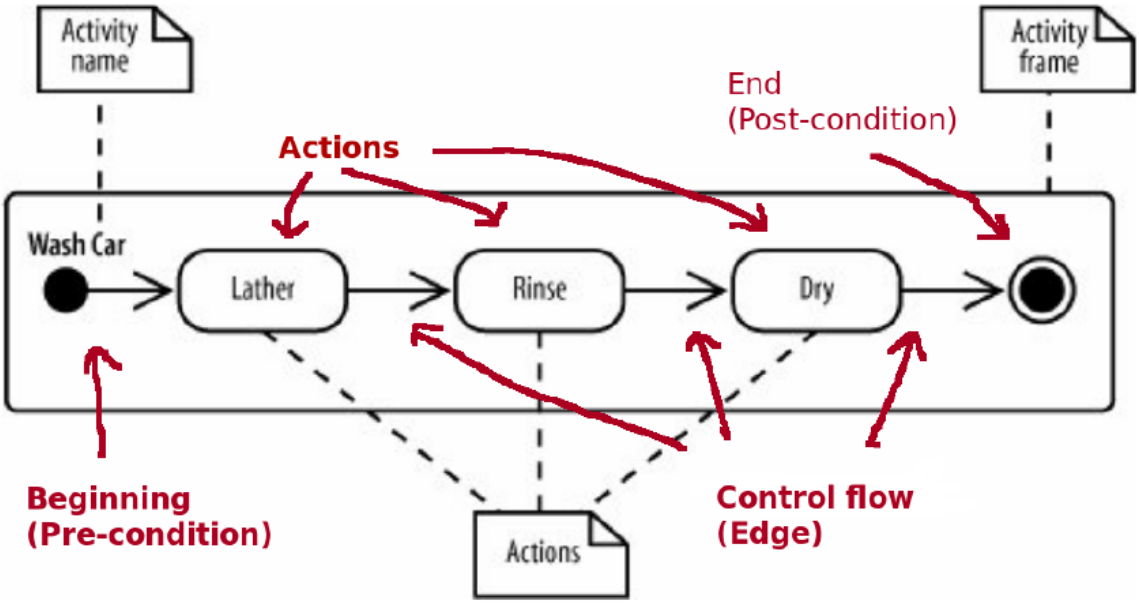
File UML Activity Diagrams.md

UML Activity Diagrams

UML Activity Diagrams

UML Activity diagrams show high level actions chained together to represent a **business process** happening in our system

Basic UML Activity Diagram



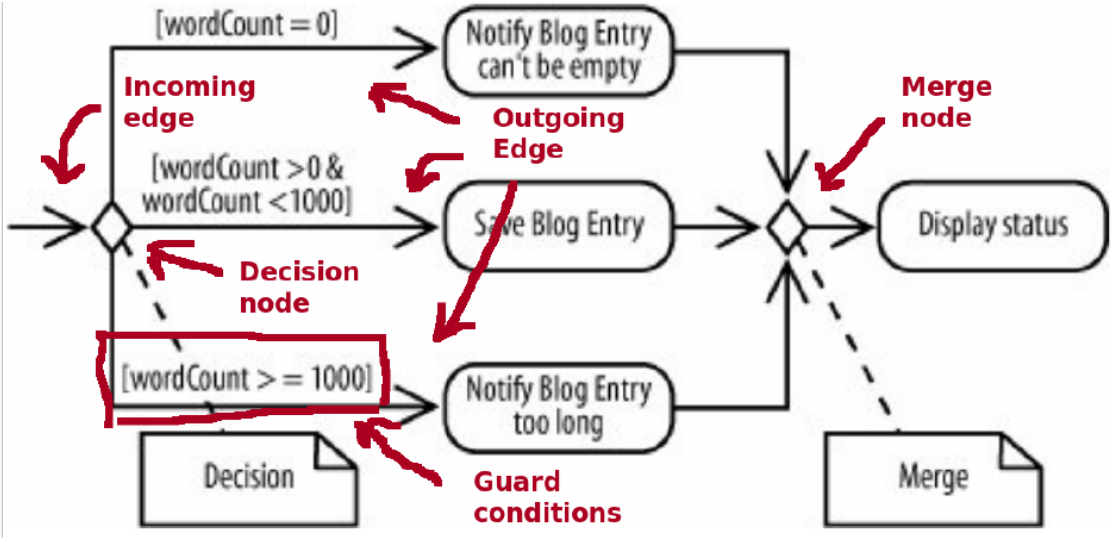
This is a **UML Activity Diagram**

- The **black circle** at the start represents the **pre condition**
- The **arrows** represent an **edge** each and show the **direction of process flow**
- The **squircles** with text in them represent **actions**
- The **black circle at the end with a no fill circle around it** represents the **post condition** (the end)

Here's what this UML Diagram shows in natural language

- **Wash car** is the **name** of this **activity**
- **First action** - Lather
- **Second action** - Rinse
- **Third action** - Dry
- After the third action this **activity is complete**

Control



This is another **UML Activity diagram.**

- **Diamonds** are used to denote **branching** or **converging**
- **Arrows** coming off a **diamond** show that a **decision** has been made there - hence its a **decision node**
- **Arrows branching** from a decision node will have a **guard condition** that denotes the requirements for the control token to take that path
- **Arrows converging** into a **diamond** show that it's a **merge node**

This is what the UML diagram shows but written in Java to make it easier to read

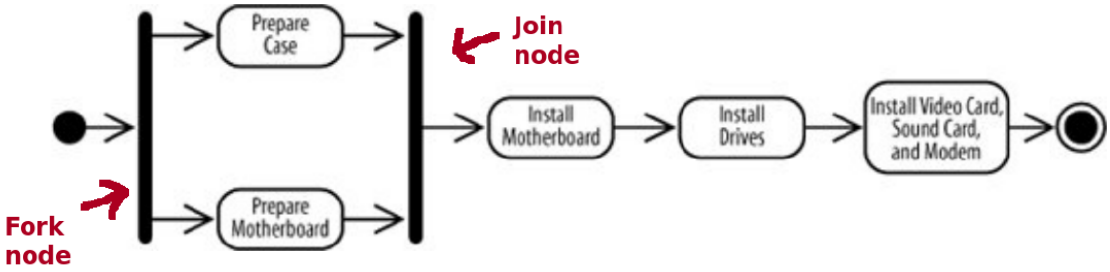
```
public void UMLDiagram2(int wordCount){

        // start decision node
        // top guard condition
```

```
        if (wordCount == 0) {
                status = "Blog entry can't be empty";
        }
        // middle guard condition
        else if (wordCount < 1000) {
                blogEntry.save();
                status = "Saved";
        }
        // bottom guard condition
        else if (wordCount >= 1000) {
                status = "Blog entry too long";
        }

        // all edges will merge to here automatically
        System.out.println(status);
}
```

Simultaneous Flows



This UML diagram shows an example of a **fork** and **join** node

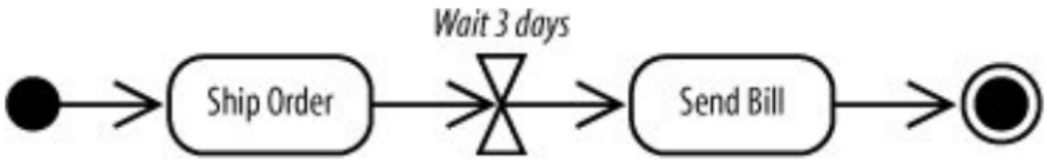
- The **black bar** with the edge going into it is a **fork node**
- The **black bar** with multiple edges leading into it and a single edge leaving it is a **join node**

- This denotes that the **multiple edges** leading off from the fork node can be run in **parallel**
- **All actions** before the join node must **finish** before the flow moves past the join node

The example activity seems to represent computer building

- The start node leads to the fork node directly
- The case and motherboard can be prepared at the same time
- **Before** installing the motherboard, both the case and motherboard **must** be prepared
- **After** both have completed, the motherboard can be installed
- The drives can then be installed
- Finally the Video Card, Sound Card and Modem can be installed before the activity is completed.

Time Events

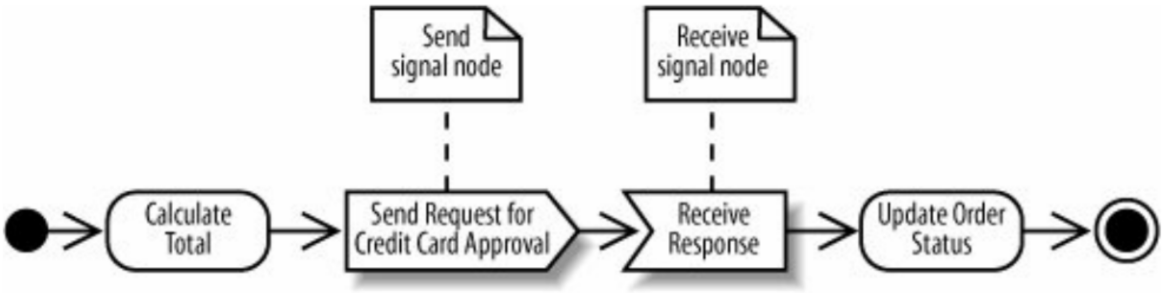


This UML diagram depicts a waiting period

- This is depicted by the two triangles facing towards each other like a ⏳
- This denotes that some time has to pass before the flow is transferred

The example shows a wait period of 3 days between shipping an order and sending a bill

External Actors



Sometimes external actors may act on the system

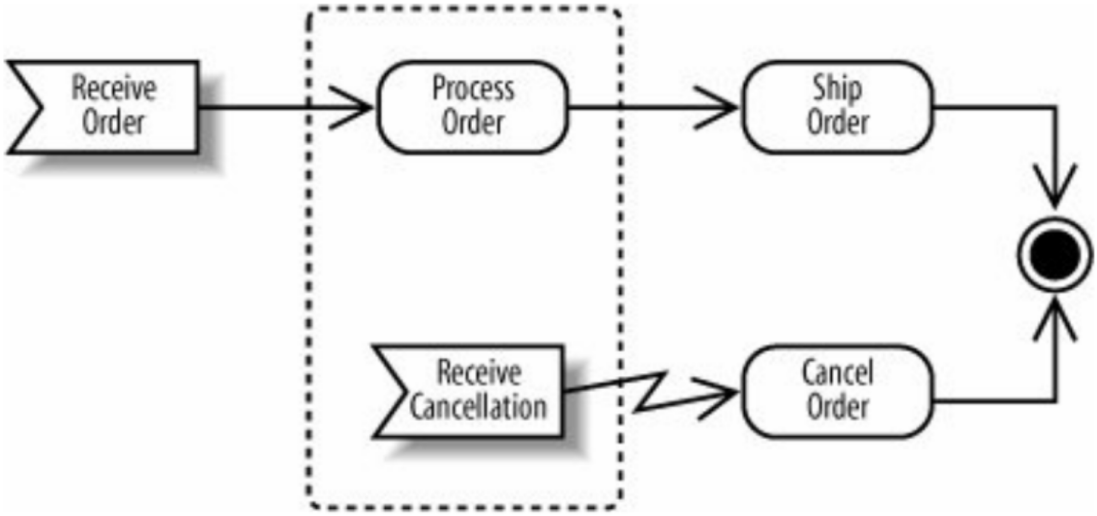
- When an activity interacts with external actor, signals are sent and received
- A **send** signal (represented by the elongated pentagon) sends a message to a external actor
- A **receive** signal (represented by the inverse) receives a message from an external actor

In the example:

- The calculate total action is completed then
- A send signal is created by sending a request for credit card approval
- Once a response signal has been received then
- The order status is updated

When the send and receive signals are combined a **synchronised** flow is depicted
If theres only a send signal, the flow is asynchronous.

Interruptions



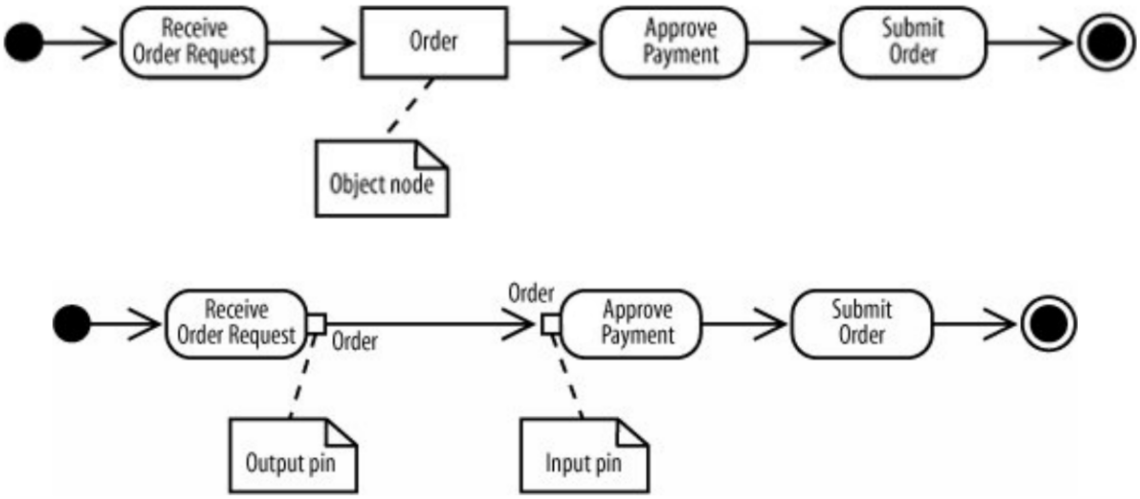
Interruptions are used to interrupt the flow of an activity

- The zone in the dotted line represents zones in which activities can be interrupted
- The **ziz-zag** line represents an interruption

The example uses an order system

- The process begins when an order signal is received
- The order can then be processed
    - In this time however, if a cancellation signal is received, the order is cancelled and this activity reaches the end
- If processed completely, ship order and this activity reaches the end

Objects and I/O



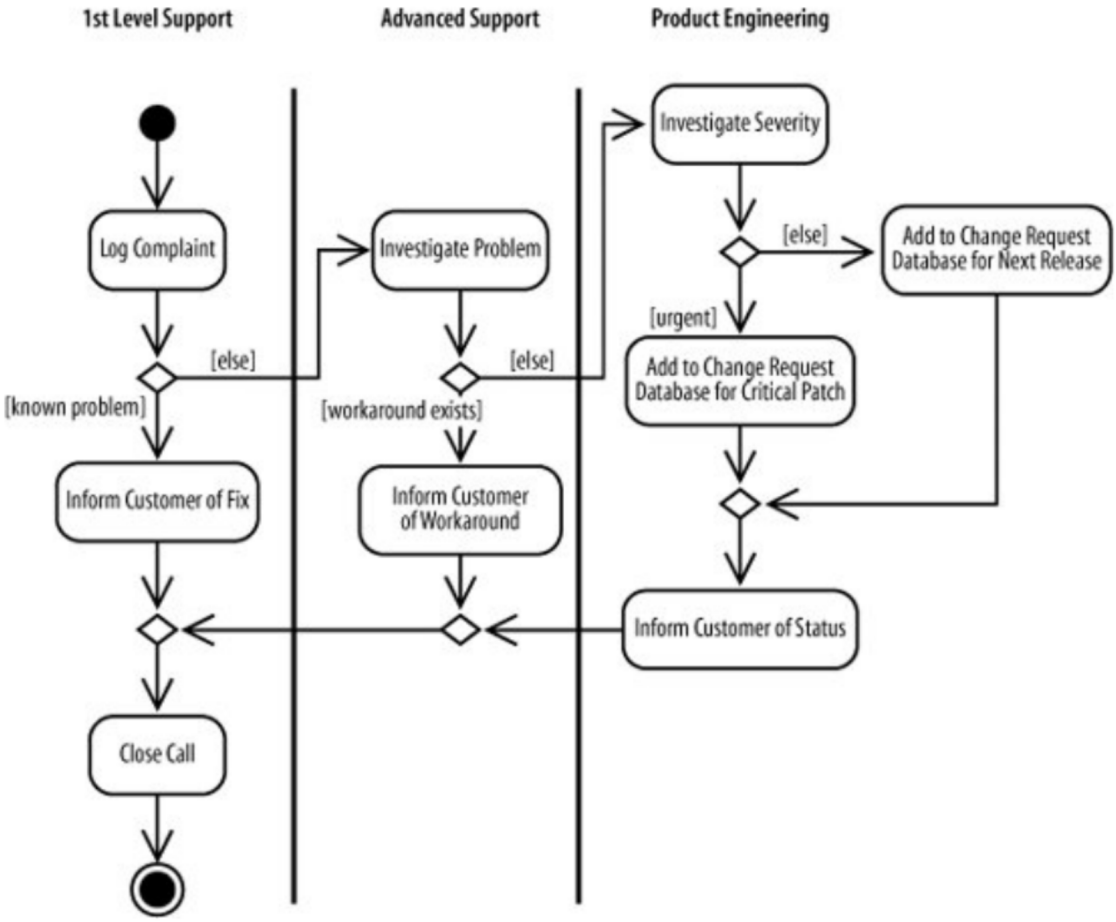
We can depict objects and inputs and outputs of our UML Activity Diagram

- Object nodes are represented using rectangles
- Input and Output pins are represented using small squares at the start and end of an activity
- Placing a pin at the end of an activity with an arrow leaving it denotes that the contents of the pin is the output of the activity it is joined to
- Placing a pin at the start of an activity with an arrow going into it denotes that the contents of the pin is the input of the activity it is joined to

The example continues with the ordering schema but represents it in 2 different ways
First:

- Receive Order Request sends an Order object to Approve Payment
  Second:
- Receive Order Request will output an Order object
- Approve Payment requires an Order object input

Swimlanes



Swimlanes are used to partition components into different sections

- Representing swimlanes is as simple as creating lines partitioning it from other swimlanes and labelling them on the top

In the example

- 3 swim lanes exist
  - 1st Level Support
  - Advanced Support
  - Product Engineering
- The activity starts in 1st level support

- This can then be escalated to Advanced Support
- And then further escalated to Product Engineering