

# CSIT 314 Lecture Notes

## Overview

### Topics Covered

- Introduction and Software Development Lifecycle
- Overview of software development process models (including evolutionary process models)
- Unified Software Development Process
- Unified Modelling Language
- Agile principles of software development
- Scrum methodology for software development
- Extreme Programming
- Dynamic Systems Development Method
- Test driven software development
- Capability Maturity Model Integration (CMMI) model
- Emerging trends in software development methodologies

## Introduction and Software Development Lifecycle

### What are the software Development Activities:

- **Planning**
  - Identify business value
    - Develop a System Request(SR) to initiate the project. This identifies the business value of a new system
    - An SR is comprised of the business need, functionality, expected value and any issues or constraints.
  - Analyse feasibility
    - Feasibility Study is conducted by a business analyst to determine what the consequences are if the business chooses to invest in the approved SR
    - Three areas of feasibility are covered, Technical - Can we build it, Economic - Financially beneficial, Organisational - Will it be used.
  - Develop work plan
    - A method of breaking down a projects workload into tasks and sub-tasks.
      - **Example 1:** A *Work Breakdown Structure(WBS)* to identify tasks, create a timeline, highlight deliverables and assign team members.

- **Example 2:** A *Gantt chart* shows tasks, sub-tasks and their start date/end dates, milestones and task dependencies can also aid in developing a work plan.

- Estimate

- Calculate the effort it takes for the planning phase using industry standard % (person months)
- COCOMO (Constructive Cost Model): Is a regression model based on LOC, i.e number of Lines of Code.
  - Constants (These are the characteristics of projects the method can be applied to):
    - **Organic:** A small team develops a small system with flexible requirements in a highly familiar in-house environment.
    - **Embedded:** The system has to operate within very tight constraints and changes to the system are very costly.
    - **Semi-detached:** This combines elements of the organic and the embedded types or has characteristics that came between the two.
  - Basic Mode Algorithm:
    - **Effort = c x size<sup>k</sup>**
      - Effort is measured in person-months

- Identify risk

- What is a risk?
  - Something that can impact the delivery of the project
- **STEPS:**
  - **STEP 1:** Identify and classify all risks
  - **STEP 2:** Create a prevention and contingency plan
  - **STEP 3:** Implement the strategy created to reduce the risk.
- What are the **three strategies** for reducing risk?
  - Avoid the risk
    - Change the requirements, design, performance or functionality
  - Transfer the risk
    - Transfer responsibility or purchase insurance
  - Accepting the risk
    - Accept, mitigate or control the implications of the risk

## ➤ Requirements analysis

- What is the requirements analysis phase?
  - The process of establishing what services are required
  - The constraints on the systems both in development and its operation.
- What are the two major activities in the requirements analysis phase?
  - Requirements Elicitation
    - Who are the stakeholders of this project?
    - What do the stakeholders require or expect from the system?
    - Interviews, questionnaires, meetings between clients and developers
  - Requirements Specification
    - Defining the requirements in detail
    - Use cases are the major part of a requirements specification

## ➤ Design

- Design vs Requirements
  - Design is **HOW** we build a system
  - Requirements Analysis is **WHAT** is to be built
- What are the 5 types of design
  - Architectural Design
    - The major structure of the software
  - Sub System Design
    - Describes interfaces and protocols of each subsystem
    - The structure of each subsystem divides into packages and components
  - Detailed Design
    - Describes each class, methods, attributes and data types.
  - Persistent Data Design
    - Describes choice of database, tables primary and foreign keys
  - GUI Design
    - Describes how the interface will look
    - What tools will be used to build it
    - Structure of the GUI
    - Design/Style Guidelines to be followed.

## ➤ Implementation

- What is the Implementation Phase?
  - The implementation plan characterises the creation of the actual product by the projects Daniel.

### ➤ **Software verification and validation**

- What is the Verification and Validation Phase?
  - Verification:
    - The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.
  - Validation:
    - The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
- What is the most common V&V activity?
  - Software Testing
- What is involved in System Testing?
  - Executing the system with test cases derived from the specification of real data to be processed by the system.

### ➤ **Cost of defects:**

- Cost of correcting an error in requirement specifications increases as we move through life cycle phases.

### **Software maintenance and evolution**

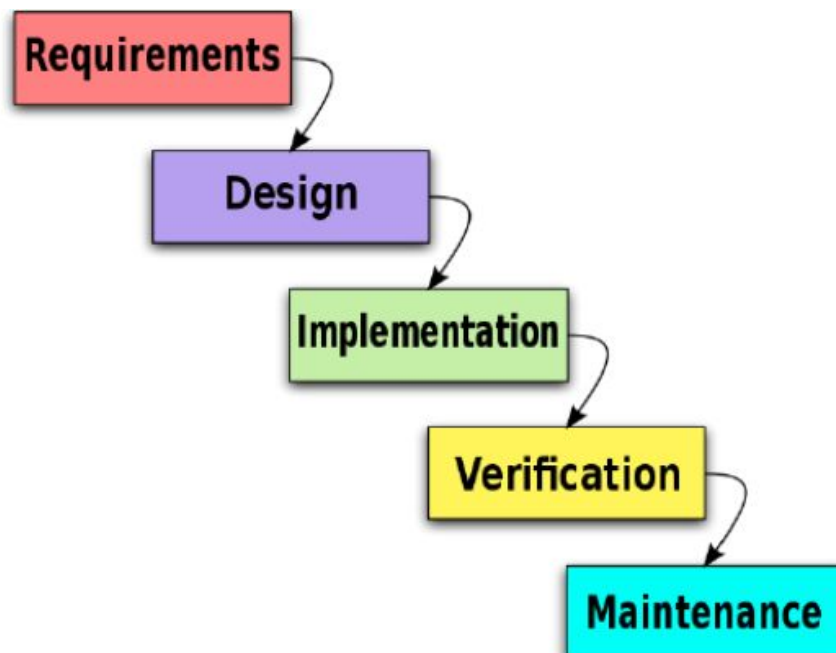
- The Maintenance and evolution Phase occurs once the system is operational. It includes implementation of changes that software might undergo over a period of time, or implementation of new requirements after the software is deployed at the customer location.

### **Types of software maintenance**

- Adaptive Maintenance
  - Changing the system in response to changes in its environment so it continues to function.
- Corrective Maintenance
  - Fixing errors and bugs.
- Perfective Maintenance
  - Changing the system's functionality to meet changing needs (normally based on customers feedback)

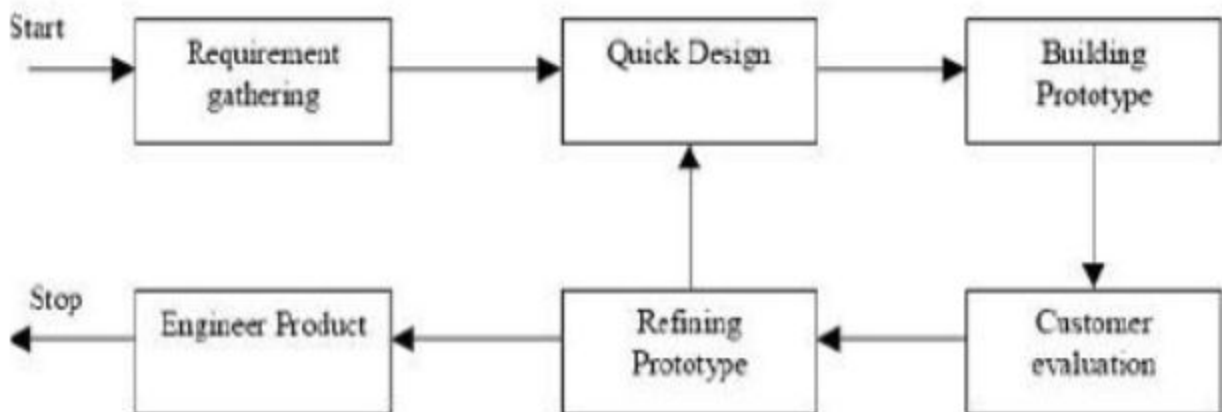
## Software Process Models

- What is a Software Process Model?
  - A structured set of activities required to develop a software system
- What are the 4 main processes all Models Contain?
  - Requirements specification
    - Defining what the system should do
  - Design & Implementation
    - Defining the organisation of the system and implementing the system
  - Verification and Validation (V&V)
    - Checking that the system conforms to the specification and does what the customer wants it to do
  - Maintenance and Evolution
    - Altering the system in response to changing customer needs
- What is the Waterfall Model
  - A sequential software development process, where **one phase must be completed and correct prior to moving onto the next.**
- What are some of the limitations of the Waterfall Model
  - Hard to respond to changing customer requests
  - Requirements must be fully understood and have limited changes during design phase
  - Designers must produce a fully correct design prior to implementation.
  - You are not allowed to go chasing them.
    - Stick to the rivers and the lakes like you are used to.

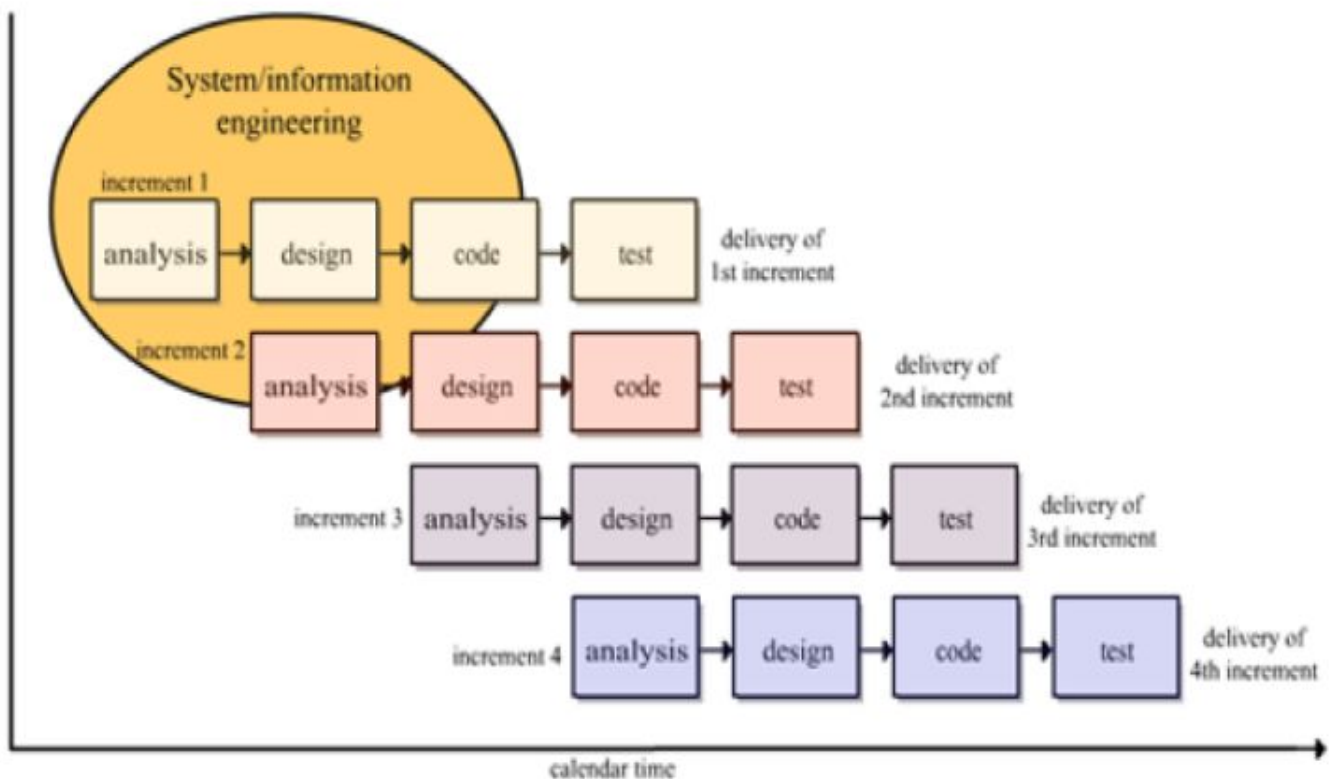


➤ What is the **prototyping model**?

- **STEP 1:** Developers gather requirements and quickly build a working prototype for customer evaluation.
- **STEP 2:** The prototype is refined based on customer feedback and then returned for further customer reviews.
- **STEP 3 (repeat 1 and 2):** This process is repeated until the customer is satisfied.
- **STEP 4 :** The prototype is then discarded and a new high quality system is built based on the final prototype design.
- Prototyping Pro's
  - Users actively involved in the development
  - Great for Human-Computer Interface projects
- Reduce time and potentially cost.
  - Prototyping Con's
  - Users can get confused between prototype and completed system
  - Developers can spend an excessive amount of time creating a prototype that is too complex

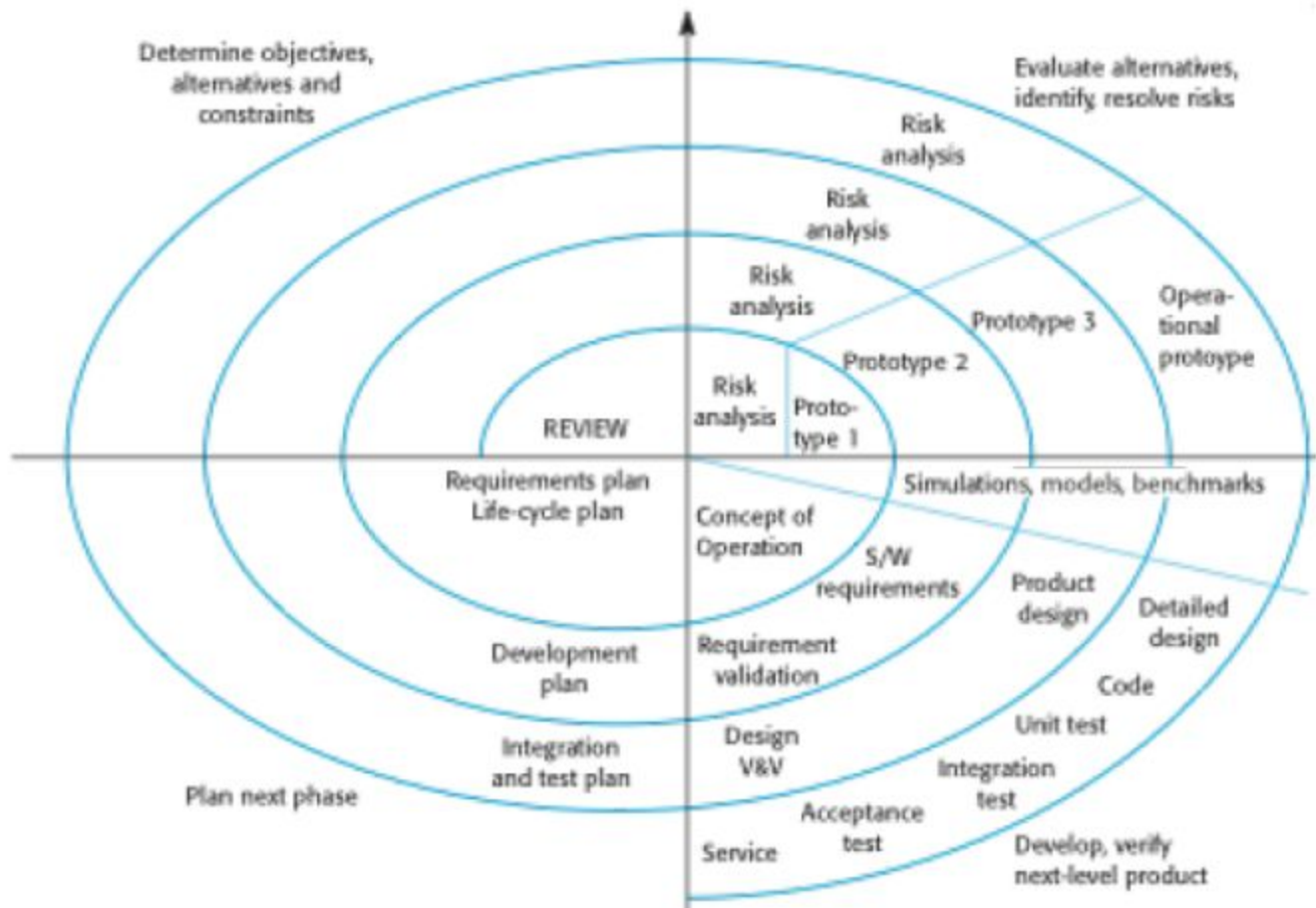


- What is the **Incremental and Iterative Model**?
  - The development of a system through repeated cycles
  - User requirements are prioritised as the highest and included in early increments
  - Each iteration has new features and capabilities
- Incremental Pro's
  - Reduced cost in accommodating customer changes
  - Early increments act as a prototype to help elicit future requirements
- Incremental Con's
  - System structure tends to degrade as new increments are added
  - Some costs will be repeated.



➤ What is the **Spiral Model**

- The process is represented as a spiral rather than a sequence of activities
- Each loop in the spiral represents a phase.
- *When are they used?*
  - Mostly used in large projects
    - Eg, Games and Military Projects










## Rational Unified Process(RUP)

### ➤ What is RUP?

- A flexible model for a software development process tailored to your projects needs

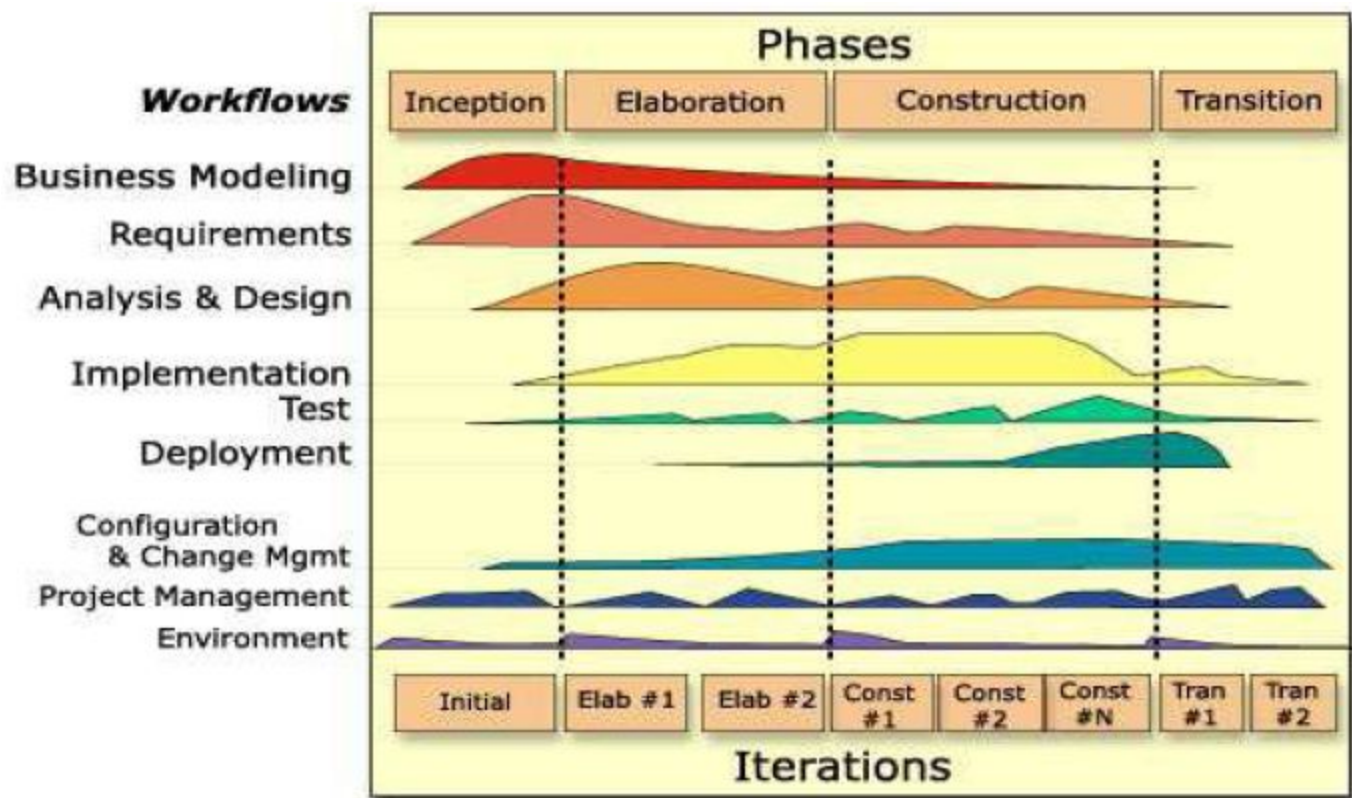
What are the six principles to follow when designing any software project to minimize faults and increase productivity?

1. Develop iteratively
2. Manage requirements  
 Always keep in mind the requirements set by customers and will be changed by users.
3. Use components  
 Test individual components and reusability.
4. Model visually  
 Use diagrams to represent all major components, users, and their interaction. UML.
5. Verify quality  
 Always make testing a major part of the project at any point of time.
6. Control changes  
 A project may be created by teams, sometimes in various locations, different platforms may be used, etc. As a result it is essential to make sure that changes made to a system are synchronized and verified constantly.

## RUP Process Structure

What are the two axis?

1. Horizontal axis represents time and shows the life cycle aspects of the process as it unfolds.
2. Vertical axis represents core process workflows, which group activities logically by nature.



## RUP building blocks

- Roles (who)

A Role defines a set of related skills, competencies and responsibilities.

- Work Products (what)

A Work Product represents something resulting from a task, including all the documents and models produced while working through the process.

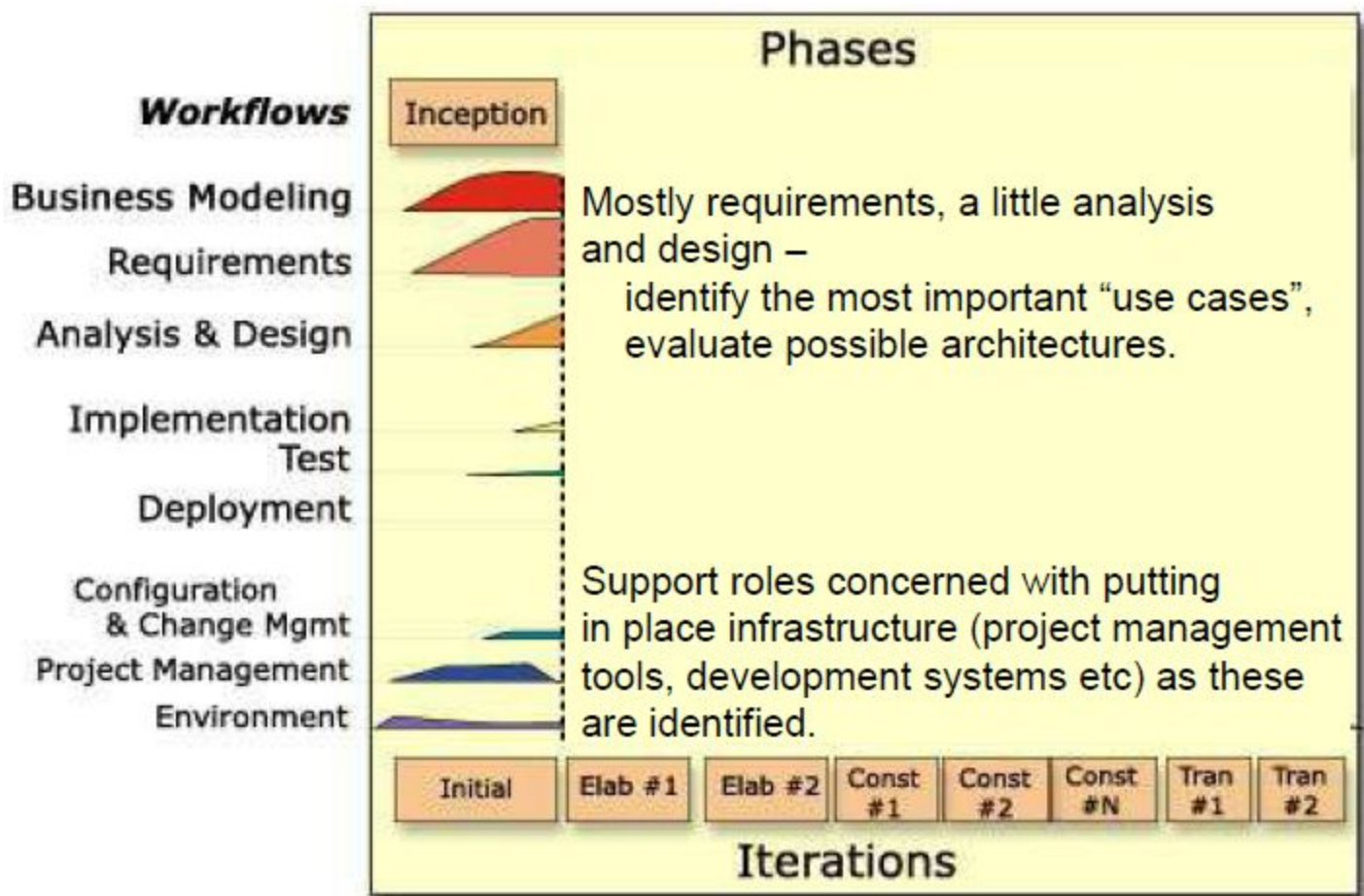
- Tasks (how)

A Task describes a unit of work assigned to a Role that provides a meaningful result.

## What are the RUP 4 Phases?

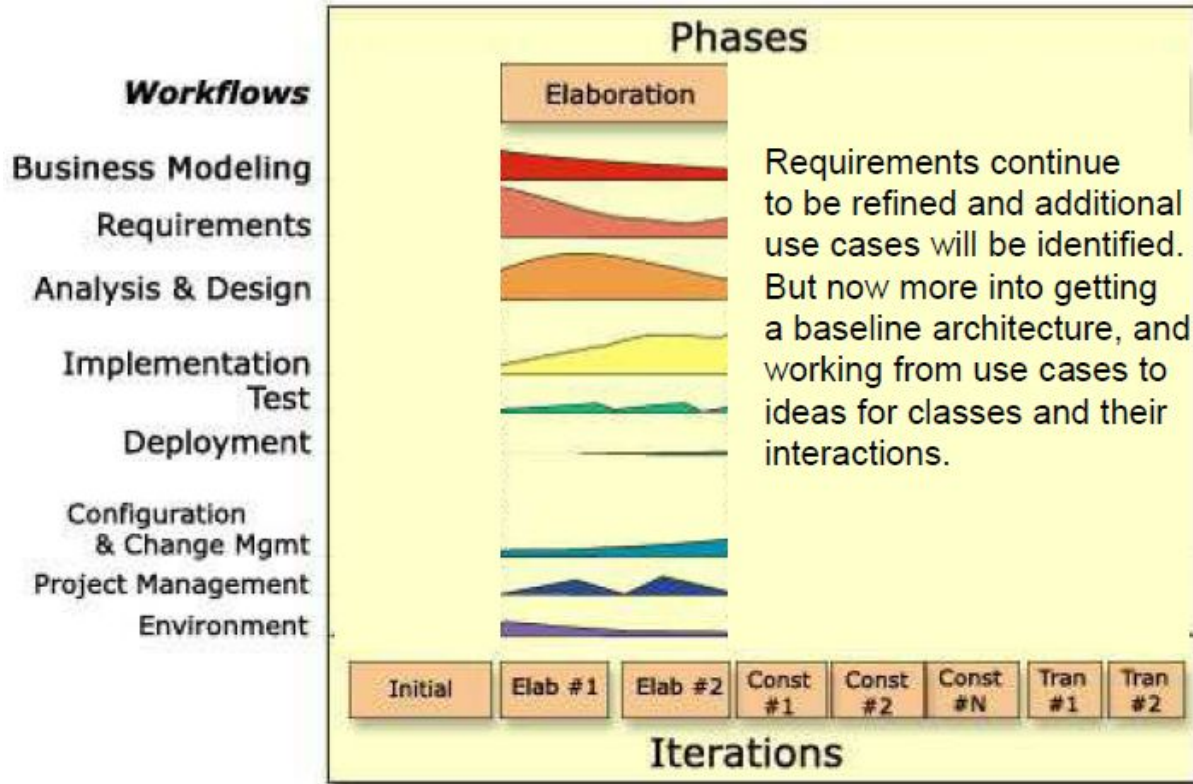
- Inception

- The idea for the project is stated. The development team determines if the project is worth pursuing and what resources will be needed.



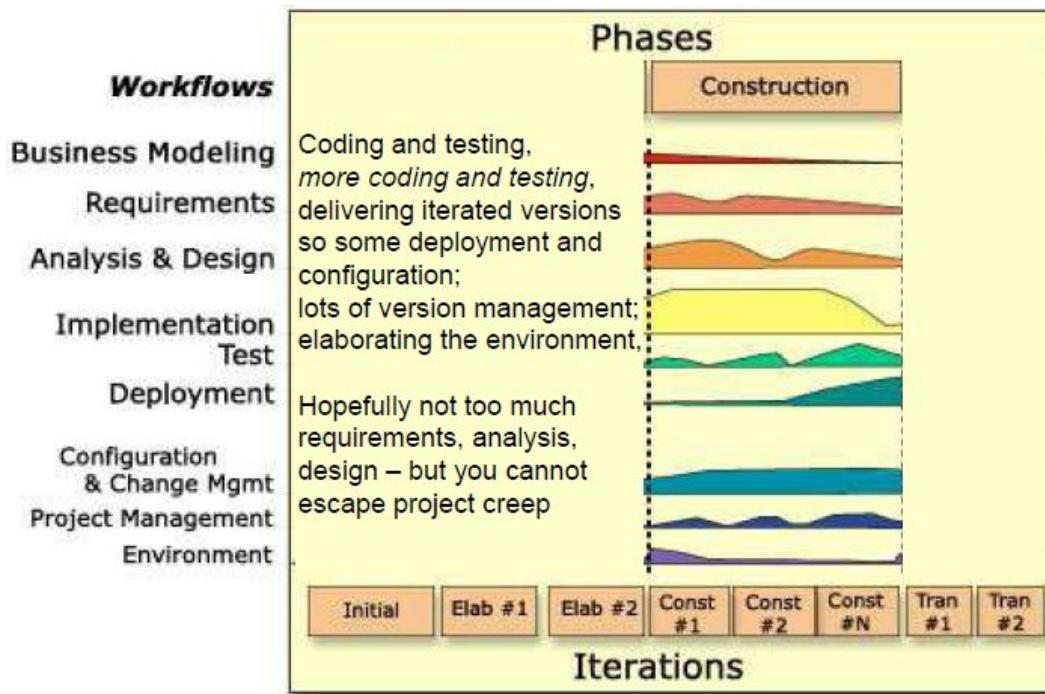
➤ Elaboration

- The project's architecture and required resources are further evaluated. Developers consider possible applications of the software and costs associated with the development.



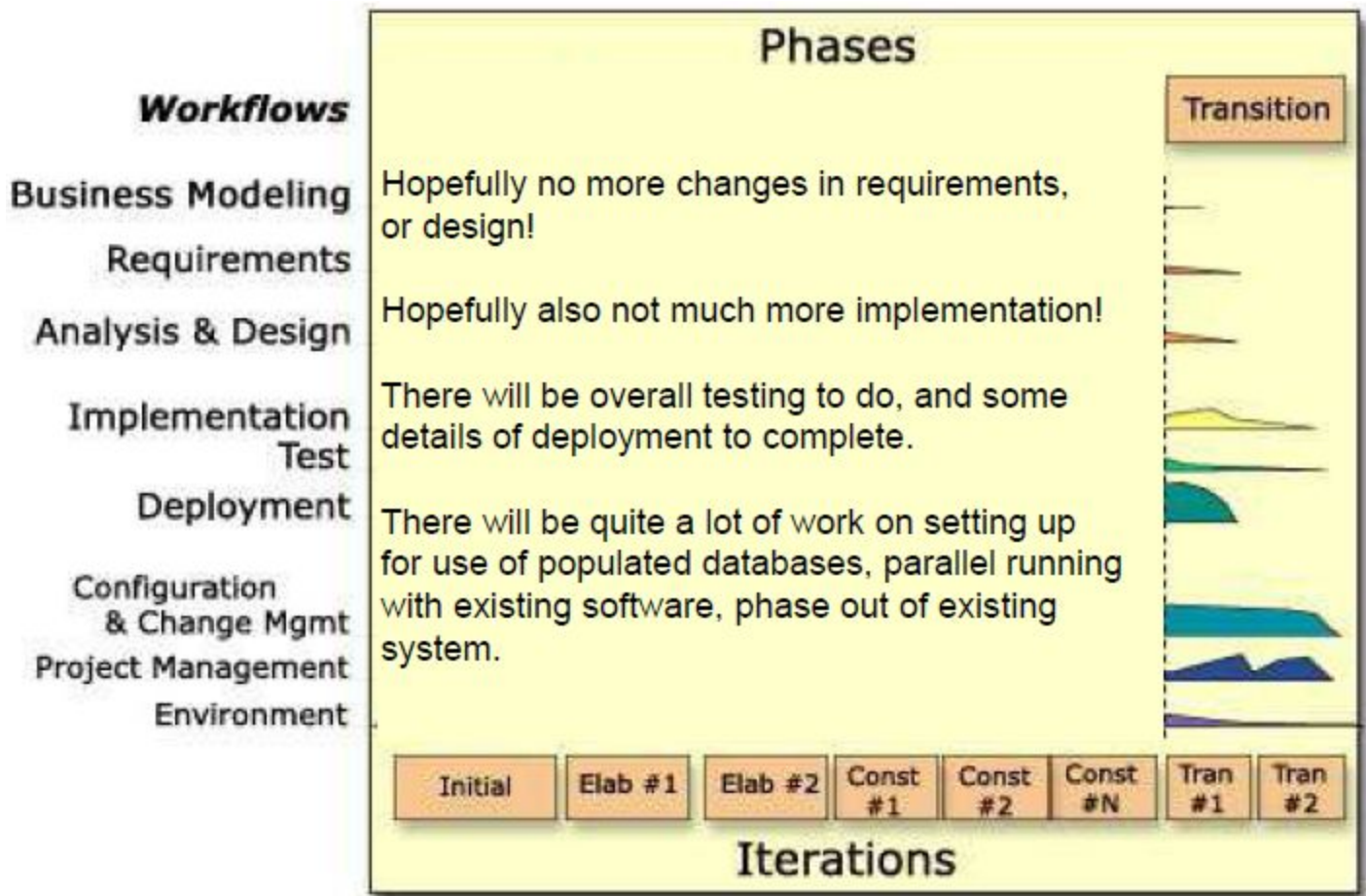
➤ Construction

- The project is developed and completed. The software is designed, written, and tested.

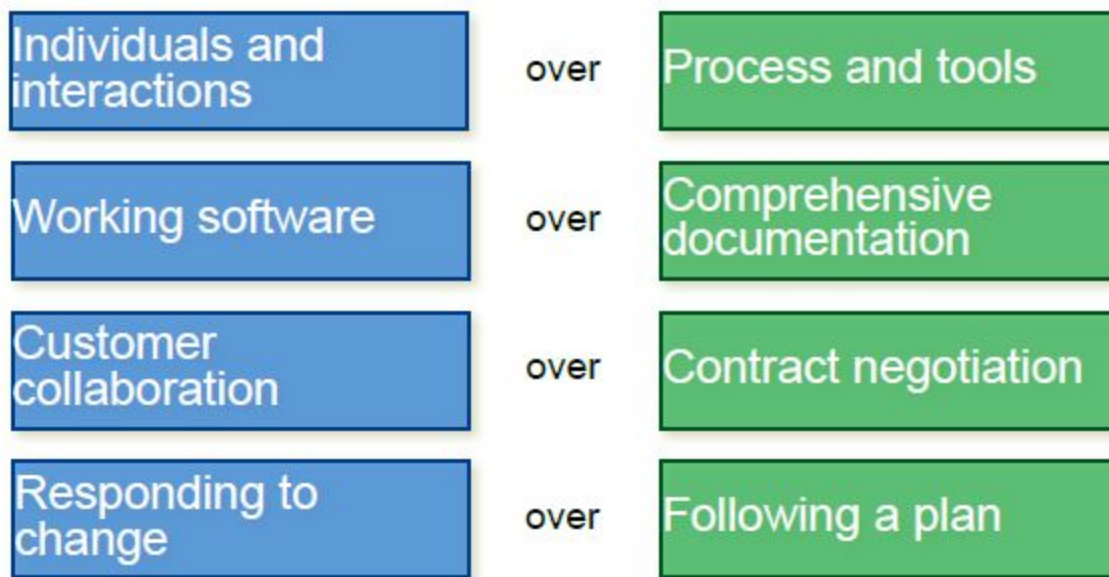




- Transition
  - The software is released to the public. Final adjustments or updates are made based on feedback from end users.



➤ The Agile Manifesto, a statement of Values



- Agile Methods:
  - **Focus on the code** rather than the design
  - Are based on an **iterative** approach to software development
  - Are intended to **deliver working software quickly** and evolve this quickly to meet changing requirements.
- What factors contribute to a high performance team:
  - Frequent communication.
    - These can be achieved by (inspect-and-adapt cycles):
      - Pair Programming (every minute)
      - Continuous integration (every few hours)
      - Stand up meeting (every day)
      - Review and Retrospective meeting (every iteration)
  - Productive interactions.
- Documentation is important but Working software is even more important
  - Delivering small pieces of working software to the customer at regular intervals is essential
- Customers should be engaged and be part of the development process
  - This ensures the product meets the customers needs
- Plans and processes need to be able to change to accommodate customer feedback

## Agile Principles

### Can A Foreign Child Solve Easy Work As Australians Seem Super Retarded

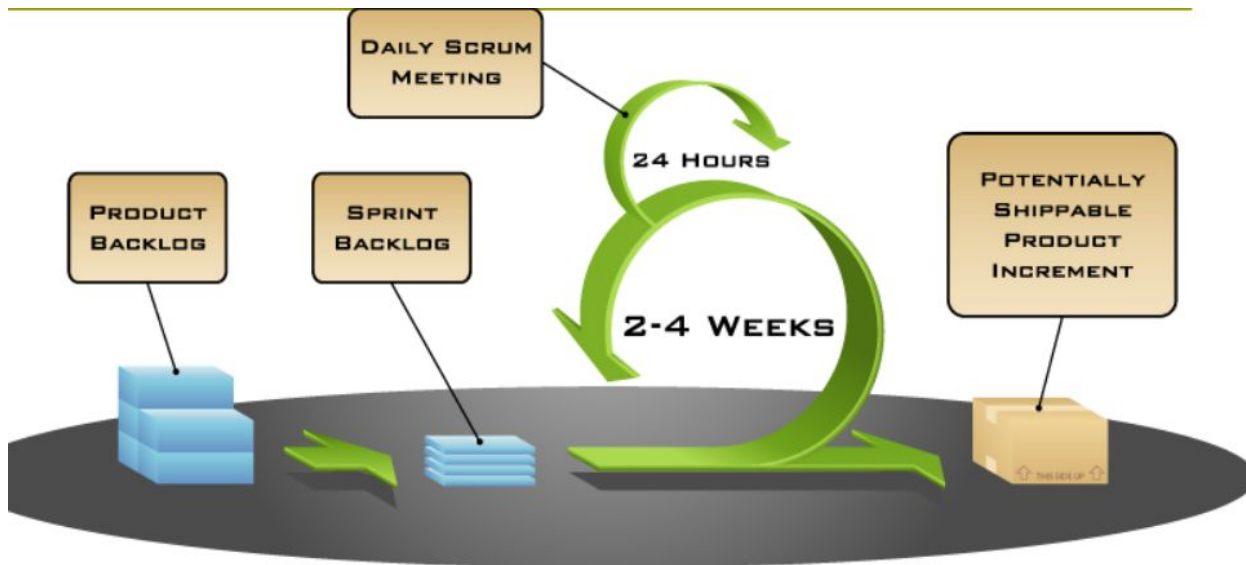
1. **Customer satisfaction through early and continuous software delivery** - The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. **Accommodate changing requirements throughout the development process** - Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Frequent delivery of working software** - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
4. **Collaboration between the business stakeholders and developers throughout the project** - Business people and developers must work together daily throughout the project.
5. **Support, trust, and motivate the people involved** - Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. **Enable face-to-face interactions** - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. **Working software is the primary measure of progress**
8. **Agile processes to support a consistent development pace** - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. **Attention to technical detail and design enhances agility** - Continuous attention to technical excellence and good design enhances agility.
10. **Simplicity** - Develop just enough to get the job done for right now.
11. **Self-organizing teams encourage great architectures, requirements, and designs** - The best architectures, requirements, and designs emerge from self-organising teams.
12. **Regular reflections on how to become more effective** - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

## Agile Software Development Methods:

- Scrum
- Extreme Programming
- Dynamic Systems Development Methods
- Kanban
- Lean Software Development

### Scrum Characteristics

- Self-organizing teams
- Product progress in a series of month-long “Sprints”
  - Typical duration is 2-4 weeks or a calendar month at most.
  - Product is designed, coded, and tested during the sprint.
  - Scrum teams (Gather requirements, design, code and test).
- Requirements are captured as items (user stories) in a list of “product backlog”
- No specific engineering practices prescribed
- One of the “agile processes”



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE



## Scrum Framework

### ➤ Roles

- Product Owner
  - The **Scrum product owner** is typically a project's key stakeholders.
  - What do they do:
    - Define the features of the product
    - Decide on the release date and content.
    - Prioritize features according to market value.
    - Adjust features / priorities every iteration, as needed.
    - Accept or reject work results
- Scrum Master
  - The **scrum master** manages the process for how information is exchanged.
  - What do they do:
    - Responsible for upholding Scrum Values and Practises
    - Represents management to the project
    - Ensure team is fully functional and productive
    - Resolves obstructions or blocks.
- Team
  - Self-organising, cross functional members of typically 5-9 people
  - Members should change only between sprints

### ➤ Ceremonies

- Sprint Planning Meeting
  - Sprint Prioritisation
    - Analyse and evaluate product backlog
    - Select the sprint goal
  - Sprint Planning
    - Decide how to achieve the sprint goal (design)
    - Create sprint backlog (tasks) from product backlog items (user stories / features)
    - Estimate sprint backlog in hours
- Sprint Review and Sprint Retrospective
  - Sprint Review Meeting
    - Team presents what has been accomplished during the sprint
    - Typically takes the form of a demo of new features or underlying architecture
  - Sprint Retrospective Meeting
    - Periodically take a look at what is and is not working
    - Whole team participates
    - Start the meeting by having all team members answer two questions;
      - 1) What went well during the sprint?
      - 2) What could be improved in the next sprint?
- Daily Scrum
  - Stand up daily meeting for approximately 15mins
  - **Three Questions everyone must answer**
    - What did you do yesterday?

- What will you do today?
- Is anything in your way?
- Helps avoid unnecessary meetings

## ➤ Artifacts

### ○ Product Backlog

- What is it:
  - The agile product backlog in Scrum is a prioritized features list, containing short descriptions of all functionality desired in the product.
- A typical scrum backlog comprises the following different types of items:
  - Features
  - Bugs
  - Technical work
  - Knowledge acquisition
- Typically, a Scrum team and its product owner begin by writing down everything they can think of for agile backlog prioritization.
- Scrum product backlog is able to grow and change as more is learned about the product and its customers.

### ○ Sprint Backlog

- What is it:
  - The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint.
- How does it work ?
  - The team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story.
    - User stories typically follow a simple template:
      - As a < type of user >, I want < some goal > so that < some reason >.
  - Most teams also estimate how many hours each task will take someone on the team to complete.

### ○ Burndown Charts

- What is it:
  - A burndown chart is a graphical representation of work left to do versus time.
- What does it do ?
  - Progress on a Scrum project can be tracked by means of a release burndown chart. The ScrumMaster should update the release burndown chart at the end of each sprint.
- How do they work
  - Horizontal axis of the sprint burndown chart shows the sprints.
  - Vertical axis shows the amount of work remaining at the start of each sprint

## **Lecture Slide 8 - Verification & Validation and Test driven development**

### **Verification:**

- What is it:
  - Verification is the process of evaluating work products, usually created within the development phase to determine whether they meet the requirements specified previously in the project.
- What does it do:
  - Ensures that the product is built according to the requirements and design specifications.

### **Validation:**

- What is it:
  - The process of evaluating the software to determine whether it actually meetings the customers / business requirements.
- What does it do:
  - This ensures that the product meets the clients needs, and not just ticks all the boxes from the specification.

### **The V and V Process**

- Verification and Validation must be applied during each phase of the software development process in order to:
  - Discover and rectify defects in a system.
  - To assess whether or not the system is usable in an operational situation (real life system).

### **Static Verification (Software Inspections)**

- What is it:
  - Is a set of processes that analyze code rto ensure defined coding practices are being followed, without executing the application itself.
- Software Inspections:
  - Involves examining the source code representation in order to discover anomalies and defects.
  - No code execution required
  - Can be applied to any part of the system. (Design, Configuration data, test data, requirements)
- Advantages:
  - You don't have to be concerned with errors hiding errors (interaction of errors) as the problem isn't being run (static testing).
  - Incomplete versions can be inspected (The program doesn't need to be operational yet)
  - Experienced reviewers are likely to have seen errors that commonly arrive (they can easily detect them)
  - Allow detection and correction in early development

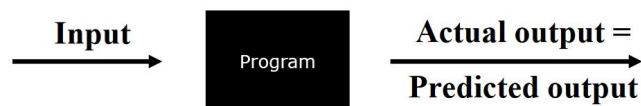
## Dynamic Verification (Software Testing)

- What is it:
  - Is performed during the execution of software, and dynamically checks its behaviour; it is commonly known as the Test phase.
- Software Testing:
  - The process of executing a program in order to:
    - Force a program to encounter an error / work incorrectly.
      - I.e. Has a high probability of finding an as yet undiscovered error.
    - To demonstrate the program works correctly.
      - E.g. Fault Testing: to show that common faults do not exist.
  - Testing can take about 30 - 40% of the development budget for the system.

## Development Testing

- What is it:
  - Development testing is a software development process that involves application of defect prevention and detection strategies in order to reduce software development risks, time, and costs.
- Testing Types
  - Unit Testing
    - What is it:
      - Unit Testing is a level of software testing where individual units / components of software are tested in isolation.
    - What does it do:
      - The purpose is to validate that each unit of the software works as designed.
      - A unit is the smallest testable part of any software.
        - E.g. Methods within an object, or objects that have several attributes and methods
  - Integration Testing
    - What is it:
      - Integration Testing is a level of software testing where individual units are combined and tested as a group.
    - What does it do:
      - The purpose is to expose faults in the interaction between integrated units.
  - System Testing
    - What is it:
      - System Testing is a level of software testing where a complete and integrated software is tested.
    - What are the characteristics:
      - System testing should focus on testing component interactions.
      - **Function Testing**
        - Tests functions of the system i.e. the functional requirements

- **Performance Testing**
  - Tests some of the non-functional requirements, e.g. reliability, availability etc
- **Acceptance Testing**
  - Performs validation testing on the system prior to handover to the customers. Most usual test phases here are Alpha Testing and Beta Testing.
- **White Box Testing**
  - White Box testing is a software testing method in which the internal structure / design / implementation of items being testing is known to the tester.
  - **Basic Forms**
    - Statement Coverage
      - What is it:
        - Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once.
    - Decision (branch) coverage
      - What is it:
        - Decision coverage / Branch coverage is a testing method, which aims to ensure:
          - ***Each of the possible branches from each decision point is executed at least once.***
          - All reachable code should be executed.
    - Condition coverage
      - What is it:
        - This technique requires ***every possible statement in the code*** to be tested at least once during the testing process of software engineering.
    - Path coverage
      - What is it:
        - Path coverage testing is a specific kind of methodical, sequential testing in which ***each individual line of code*** is assessed.
- **Black Box Testing**
  - Black Box testing is a software testing method in which the internal structure / design / implementation of the item being tested is not known to the tester.
  - Generally, set of results is derived to ensure that modules produce correct results. Aka. A predicted output should match the Actual output, else the system isn't working correctly.



- **Systematic Testing**
  - Systematic testing is a much more exhaustive means of debugging software.
  - The goal is to test the software against a variety inputs in order to find as many defects/errors as possible.
  - *All inputs are random and deemed equally valuable.*
- **Random Testing**
  - Programs are tested by generating random, independent inputs.
  - Results of the output are compared against software specifications to verify that the **test** output is pass or fail.
  - *Choose inputs that are valuable.*
- **Equivalence Partitioning**
  - What is it:
    - A testing method that divides the input domain of a program into **partitions** of **equivalent** data from which test cases can be derived.
  - What does it do:
    - In principle, test cases are designed to cover each **partition** at least once thereby reducing the total number of test cases that must be developed.
- **Boundary Value Analysis**
  - A test case design technique that complements equivalence partitioning.
  - What is it:
    - It selects test cases at the edges of each set, rather than any element of an equivalence set.
  - What does it do:
    - It leads to a selection of test cases that exercise boundary values.

## Test-driven Development

- What is it:
  - Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only.
- What are the characteristics:
  - Iterative testing and code development
  - Tests are written before code, 'passing' the tests is the critical driver of development. Aka. You don't move on to the next increment until the code that you have developed passes its test.
- Bonus Information
  - TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in other development processes.
- **Process:**
  - a. Start by identifying the new functionality required.
  - b. Write the test for this functionality and implement this as an automated test.
  - c. Run the test, along with all the other tests that have been implemented. As you have not implemented the functionality, the new test will fail.
  - d. Implement the functionality and re-run the test.
  - e. Once the tests run successfully, you move on to implementing the next chunk of functionality.
- **Benefits:**
  - Code Coverage
    - Every code segment that you write has at least one associated test so all code written has at least one test.
  - Regression Testing
    - Regression testing is checking that the testing has not 'broken' previously working code.
    - A regression test suite is developed incrementally as a program is developed.
  - Simplified Debugging
    - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
  - System Documentation
    - The tests themselves are a form of documentation that describe what the code should be doing.

## Lecture Slide 10 - Extreme Programming (XP) and Dynamic Systems Development Method (DSDM)

What is Extreme Programming?

- Extreme programming (XP) is an agile software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.

What are the Values of Extreme Programming?

- Simplicity
  - The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month
  - Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed
  - A simple design with very simple code could be easily understood by most programmers in the team.
- Communication
  - Whole team communicates face to face daily
  - Collaborate on everything from requirements to code
  - The goal is to give all developers a shared view of the system which matches the view held by the users of the system
- Feedback
  - Within extreme programming, feedback relates to different dimensions of the system development:
    - Feedback from the system:
      - By writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
    - Feedback from the customer:
      - The functional tests are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once every two or three weeks so the customer can easily steer the development.
    - Feedback from the team:
      - When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
- Respect
  - Respect for others as well as self-respect.
  - Developers respect the expertise of the customers and vice versa.
  - Management respects our right to accept responsibility and receive authority over our own work.
- Courage
  - We will tell the truth about progress and estimates.
  - We don't document excuses for failure because we plan to succeed.
  - We don't fear anything because no one ever works alone.
  - We will adapt to changes when ever they happen.



## What is the Extreme Programming Process

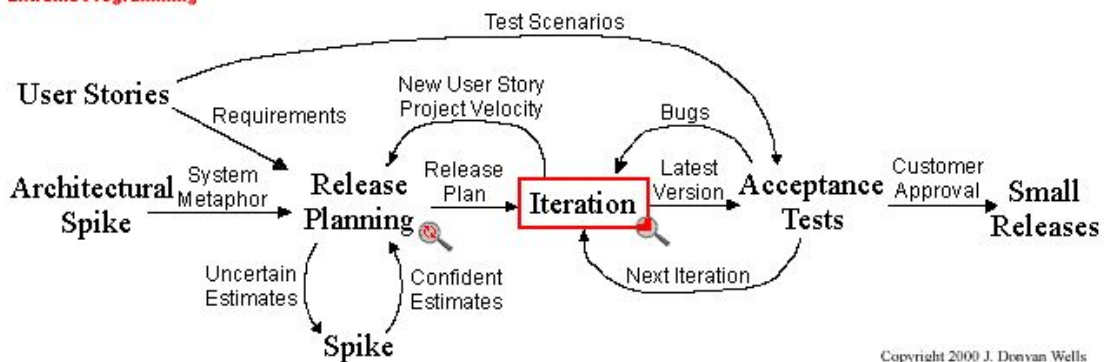
### ➤ Planning

#### ○ Release planning

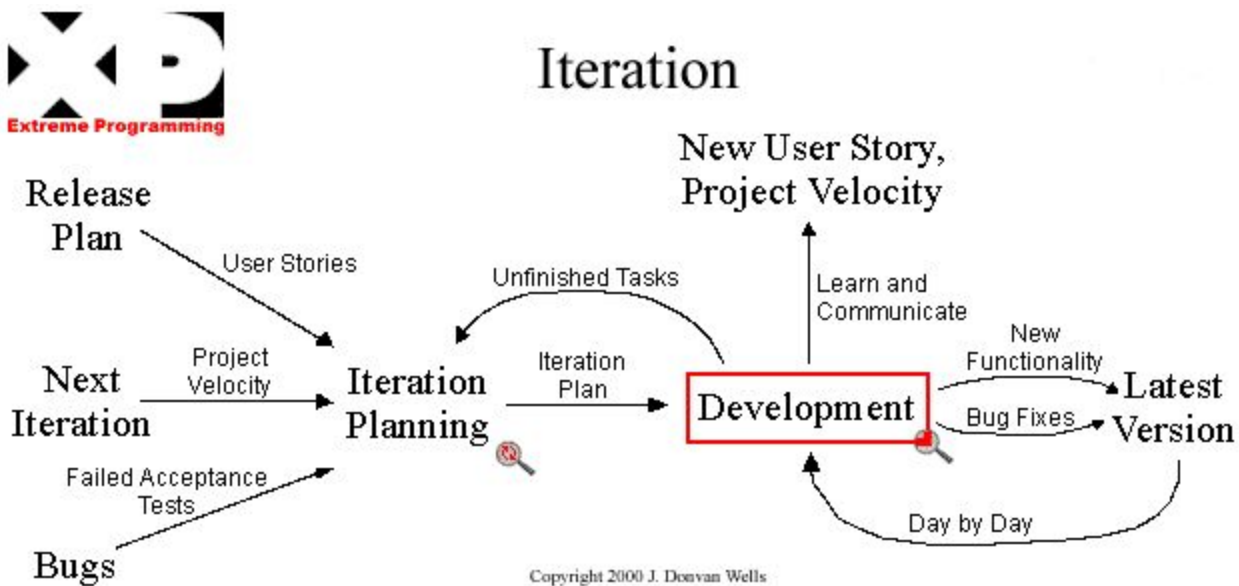
- User stories are written where each user story represents a desired feature.
- All stories represent the specification of the system.
- Customers prioritise user stories based on business value
- Estimate each user story in terms of ideal days.
- Create a set of stories to be implemented as the first/next release
- Make frequent small releases
  - Usable/Testable system
- Plan by time or scope
  - Time, how many user stories implemented by a specific date/time
  - Scope how long a set of user stories will take to finish.



## Extreme Programming Project



- Iteration Planning
  - Unimplemented user stories are selected from the iteration plan for the upcoming iteration.
  - User stories implemented in the previous iterations but did not pass acceptance tests are also selected.
  - The total user stories selected should have the total estimate up to the project velocity from the last iteration.
    - Project velocity is used to measure how much work is getting done on your project.



- Designing
  - Simplicity in Design
    - Testable, Understandable, Browsible and Explainable
  - Use Class Responsibilities and Collaboration (CRC) cards for design sessions.
  - Create spike solutions to reduce risk.
    - spike solution is a very simple program to explore potential solutions.
    - Build the spike to only addresses the problem under examination and ignore all other concerns.
    - Most spikes are not good enough to keep, so expect to throw it away.
  - Refactor whenever and wherever possible
- Coding
  - Code must be written to agreed coding standards
  - Code the unit test first (Test Driven Development)
  - All production code is pair programmed

- Review Code and Integrate often

➤ Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- Move people around.

How does Extreme Programming Differ from Scrum?

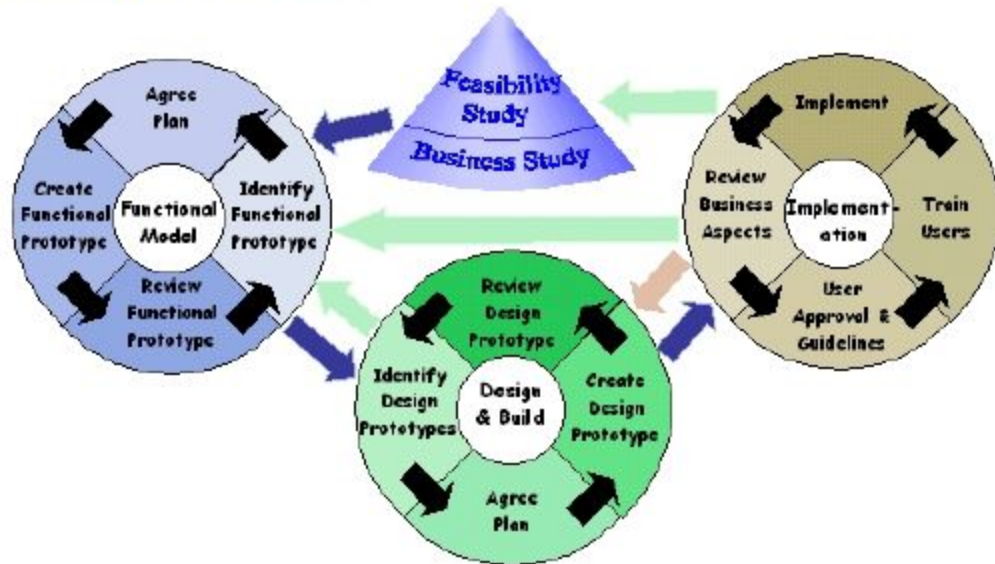
- Extreme Programming teams allows for user stories to be removed or added in an ongoing iteration.
  - Scrum teams do not allow changes into their sprints.
- Extreme Programming teams work in a strict priority order.
  - Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items.
- Extreme Programming implements engineering practises like test-driven development, pair programming, etc.
  - Scrum doesn't prescribe any engineering practices.

## Dynamic Systems Development Method (DSDM)

### What is Dynamic Systems Development

- DSDM is an agile project delivery framework, initially used as a software development method.

### The DSDM Development Process



### What are the DSDM Principles?

- Active user involvement
- Teams must be empowered to make decisions
- Focus on frequent delivery.
- Iterative and incremental development
- All changes during development must be reversible.
- Requirements are baselined at high level.
- Testing is integrated throughout the life cycle.
- Collaborative and cooperative approach.

### What are the Core Techniques used in DSDM?

- Timeboxing
  - an interval, usually no longer than 2,4 or 6 weeks, where a given set of tasks should be achieved.
  - At the end need to deliver a product.
  - Are subject to change since tasks are defined not what to be delivered.
  - Can change the tasks during time box iteration which allows for rapid response to business needs.
  - DSDM drops functionality in favour of delivering in time.
- Moscow Rules
  - Must Have
    - All features classified in this group must be implemented and if they are not delivered, the system would simply not work

- Should Have
  - Features of this priority is important to the system but can be omitted if time constraints endanger.
- Could Have
  - These features enhance the system with functional items which can easily be reassigned to a later timebox.
- Want to Have
  - These features only serve a limited group of users and are of little value.

➤ Prototyping

- Evolutionary prototyping in DSDM projects satisfy 2 principles:
  - Frequent Delivery
  - Incremental development

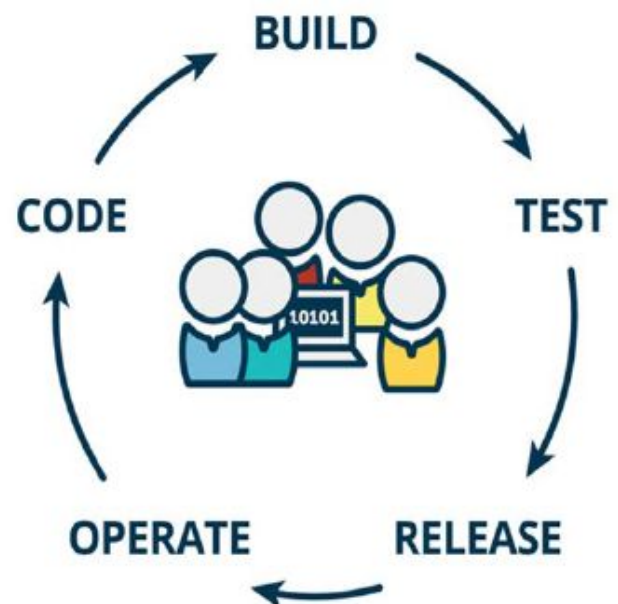
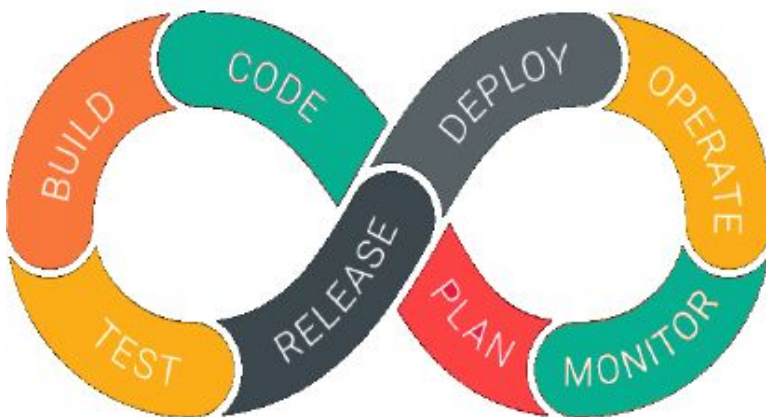
## Lecture Slide 11 - DevOps

### What is DevOps?

- DevOps is a software engineering methodology which aims to unify software development (Dev) and IT operations (Ops)
- Development and operations teams are no longer separated
  - these two teams are merged into a single team where the DevOps engineers work across the entire application lifecycle, from development and test to deployment to operations.
    - Increased collaboration between the roles of development and operations
    - shared responsibility
    - support autonomous teams
    - value feedback
    - Automation

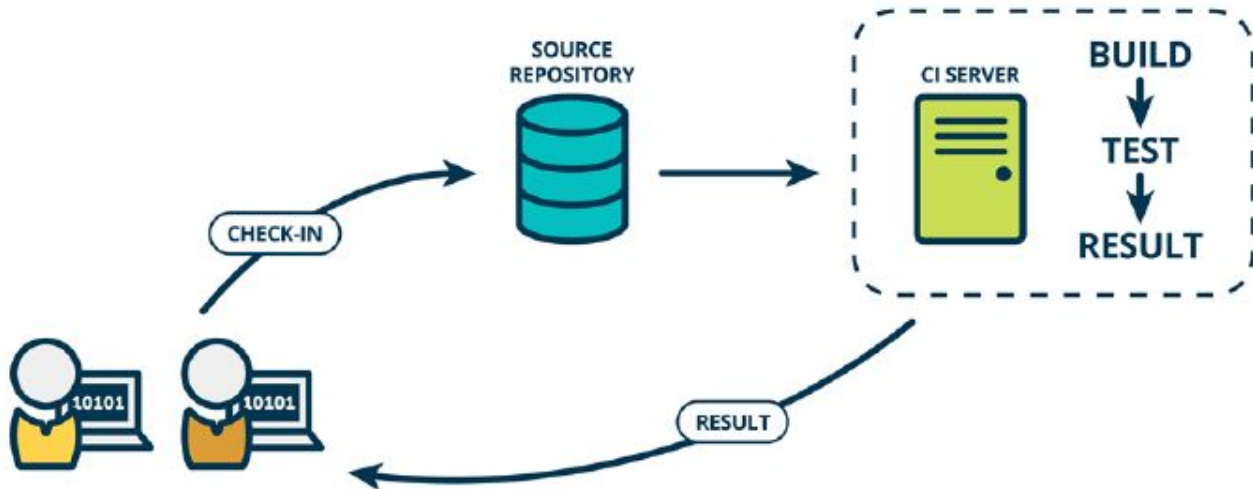
### What is the DevOps Lifestyle?

- Development
  - Agile, incremental and iterative development
- Testing
  - Test-driven development, acceptance testing
- Integration
  - New functionality is integrated with existing code, and testing takes place.
- Deployment
  - Deployment process takes place continuously.
- Monitoring
  - Bugs and performance reporting in real time.



### What is Continuous Integration?

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.
- Each integration is then verified by an automated build (including tests), allowing teams to detect problems early.



### Infrastructure as Code(IaC)

#### What is IT infrastructure?

- Physical machines, devices, OS, databases and any other systems that are used to run a software application.

#### What is Infrastructure as Code(IaC)?

- IaC Is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
- IaC is the prerequisite for common DevOps practices
- IaC model generates the same environment every time it is applied.
  - IaC enables DevOps teams to test applications in production-like environments early in the development cycle

#### What are the benefits of DevOps?

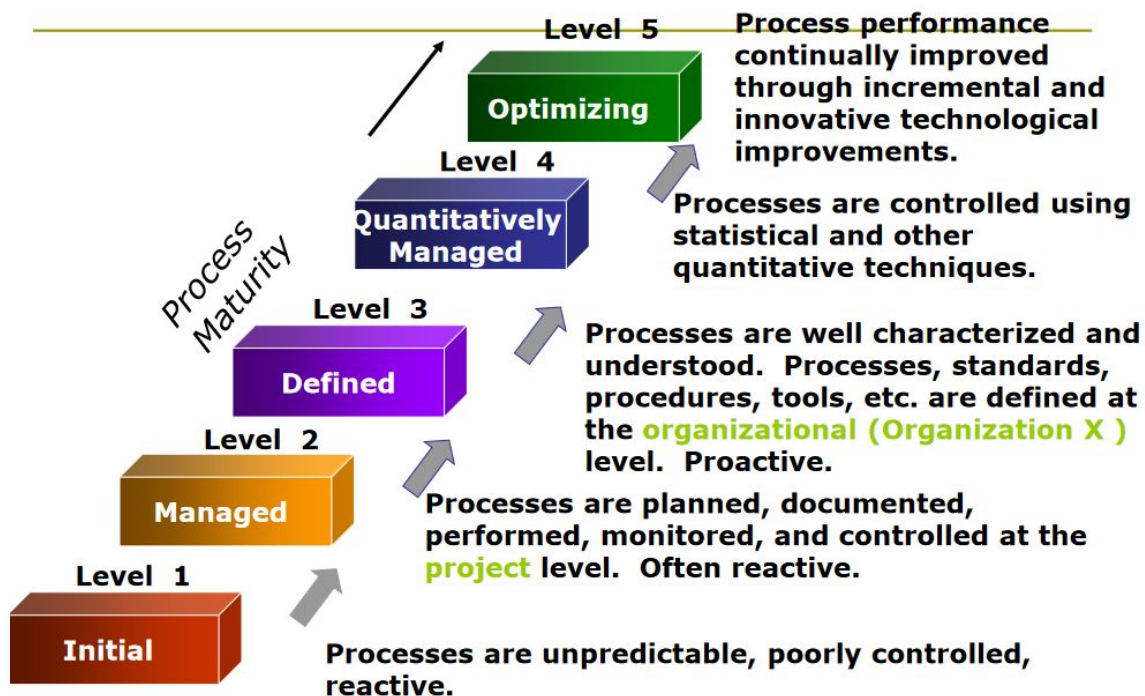
- Rapid Delivery
  - Innovate and improve your product faster (e.g. release new features and bug fixes quickly) to respond to your customers' needs and build competitive advantage.
- Reliability
  - Ensure the quality of application updates and infrastructure changes.
- Scale
  - Operate and manage your infrastructure and development processes at scale.

## Lecture Slide 12 - Introduction to Capability Maturity Model Integration (CMMI) model

### Capability Maturity Model Integration (CMMI) model

- What is it:
  - The **Capability Maturity Model Integration (CMMI)** is a process and behavioral model that helps organizations streamline process improvement and encourage productive, efficient behaviors that decrease risks in software, product and service development.
- What does it do:
  - Helps organizations:
    - Streamline process improvement
    - Encouraging a productive.
    - Efficient culture that decreases risks in software. product and Service development

### CMMI Staged Representation - 5 Maturity Levels





- **Maturity Level 1 - Initial**
  - Ad Hoc / Chaotic environment.
  - Organisation does not provide a stable environment.
  - Success is these organizations depends on the competence of the workers / less on proven processes.
  - Frequently exceed the budget and schedule of their projects.
  - Tendency to over commit, abandon processes in times of crisis, and not be able to repeat their past successes.
  
- **Maturity Level 2 - Managed**
  - The projects of the organization has ensured that requirements are managed and that processes are planned, performed, measured, and controlled.
  - Process discipline reflects ensures that existing practices are retained during times of stress.
  - Projects are performed and managed according to their documented plans.
  - Requirements, processes, work products, and services are managed. The products / delivery of these services are visible to management at defined points for review.
  - Work Products are reviewed with stakeholders and are controlled.
  - The work products and services satisfy their specified requirements, standards, and objectives.
  
- **Maturity Level 3 - Defined**
  - At maturity level 3, processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.
  
- **Maturity Level 4 - Quantitatively Managed**
  - At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.
  
- **Maturity Level 5 - Quantitatively Managed**
  - At maturity level 5, processes are concerned with addressing common causes of process variation and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical predictability) to achieve the established quantitative process-improvement objectives.

## Capability Maturity Model Integration (CMMI) model and Agile

- **Backlog Grooming:**
  - What is it:
    - A Common agile technique used by scrum teams to produce a prioritized backlog of epics and user stories before and during a sprint.
- **Continuous Integration:**
  - What is it:
    - An approach to continuous testing and product integration popular with agile teams that was first introduced in Extreme Programming (XP)
- **Daily stand/Daily Scrum:**
  - What is it:
    - Daily Standup Meeting is used as a way to identify issues and risks earlier than a traditional project (“fail fast”), and to increase collaboration between agile team members
- **Team Estimating / Planning poker:**
  - What is it:
    - An agile estimation technique that establishes relative sizing using story points and rough order of magnitude estimation.
- **Pair Programming:**
  - What is it:
    - Two people working on one machine, one drivers (coding) while the other navigates (assists the programmer).
  - What does it do:
    - Usually increases the initial cost of programming, but it more than pays for itself in increased code quality.
- **User Stores:**
  - What is it:
    - Allow easy understanding of customers / stakeholder requirements, these leads to clearer instructions for programmers when coding.
- **Test-driven development:**
  - What is it:
    - Powerful technique that can improve the quality of code and requirements, so therefore has a strong relationship to Validation, Verification, and Requirements Development.

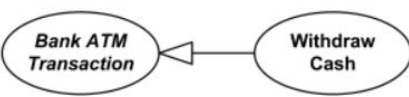
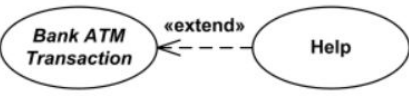
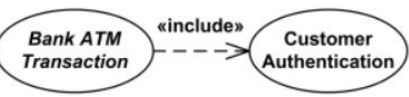
## UML Diagrams

- Structural model
  - **“Class”** diagrams.
    - the structure and substructure of the system using objects, attributes, operations, and relationships.
- Functional model
  - **“Use case”** diagrams.
    - the functionality of the system from the user's point of view.
- Behavioural model
  - **“Sequence”** diagrams, **“Activity”** diagrams and **“State”** diagrams.
    - the internal behaviour of the system.

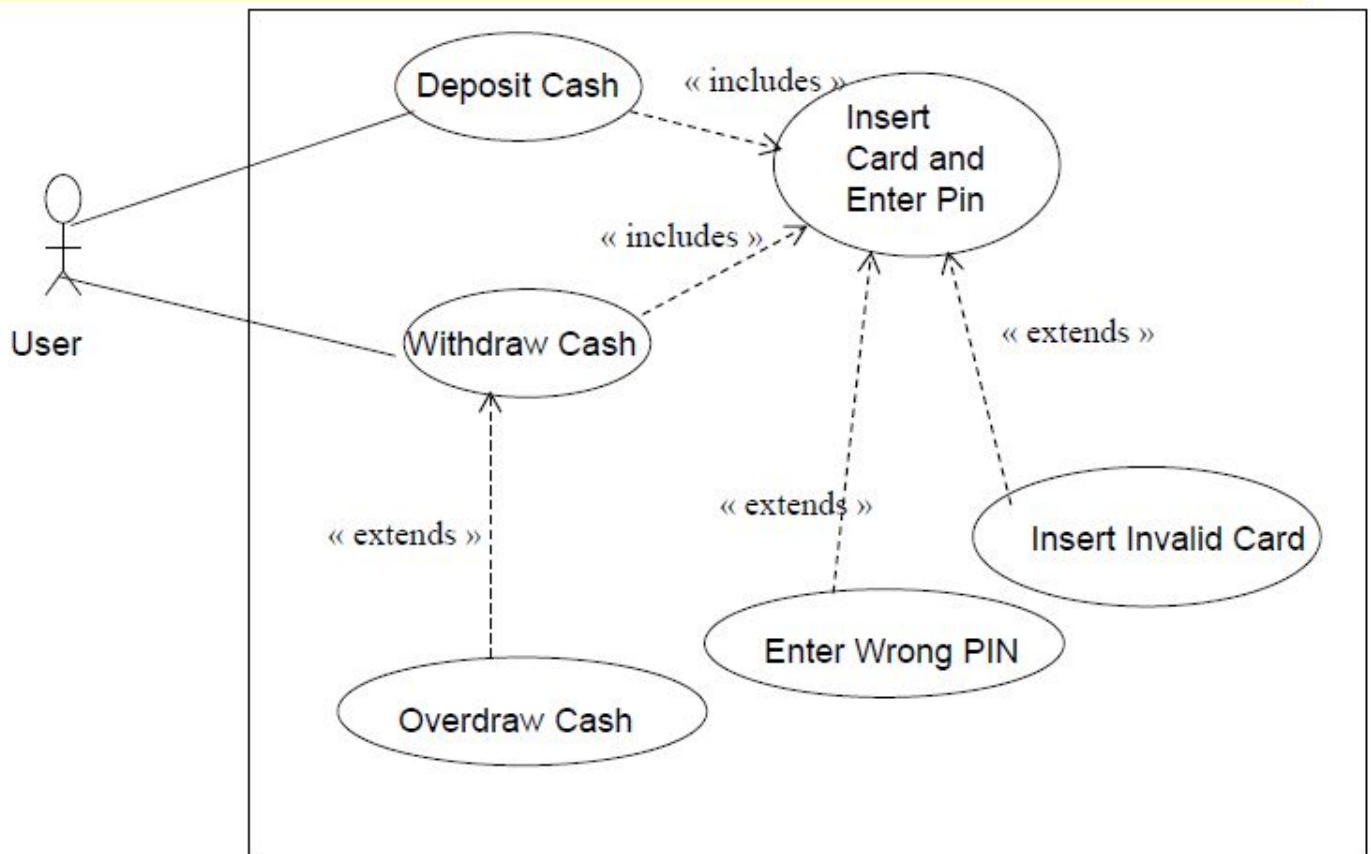
## Use Case Diagram

### Use Case Relationships Compared

This site received many requests related to which use case relationship should be used in which situation. I combined several key points from UML 2.4 Specification into the table below. Also, take a look at related discussion in the next paragraph.

Generalization	Extend	Include
		
Base use case could be <b>abstract use case</b> (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete ( <b>abstract use case</b> ).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

### Example



# Template

---

<b>Name: Name of Use Case (use verb-noun)</b>	<b>ID: unique</b>
<b>Stakeholders and goals:</b> Specify the Stakeholder(s) and goals	
<b>Description:</b> High level description of the use case (one sentence max.)	
<b>Actors:</b> Name the actor(s) who interact with this use case	
<b>Trigger:</b> Specify what actor or thing triggers this use case	
<b>Normal flow:</b> Use numbered simple sentences, each on a new line (it is best to use subject-verb-object sentences where possible)	
<b>Sub-flows:</b> Same as normal flow except prefix numbers with "S"	
<b>Alternative/Exceptional flows:</b> Same as normal flow except use numbers which refer to Normal/sub-flows statements e.g. 3b, S1a, 2a	

# Example

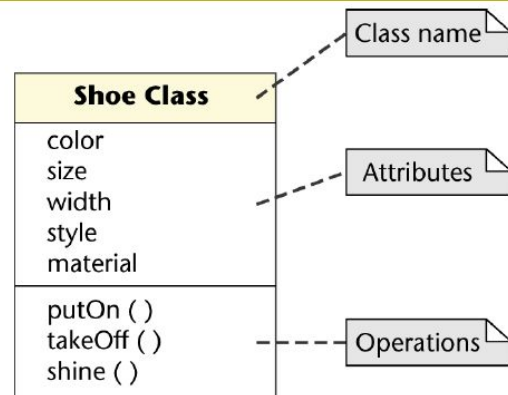
---

<b>Name: Borrow Book</b>	<b>ID: 2</b>
<b>Stakeholders and goals:</b> Student – wants to borrow book from university library	
<b>Description:</b> A borrower wants to borrow a book and must present his/her ID card to the system.	
<b>Actors:</b> Borrower	
<b>Trigger:</b> The borrower in the library brings a book(s) to the librarian's desk	
<b>Normal flow:</b> <ol style="list-style-type: none"><li>1. The borrower swipes his/her card through the card reading machine</li><li>2. The system checks the ID of the card in the database</li><li>3. The borrower passes the book through the bar-code scanner</li><li>4. The system displays the date for returning the book</li><li>5. Repeat steps 3-4 for each book to borrow</li><li>6. End</li></ol>	
<b>Sub-flows:</b> None	
<b>Alternative/Exceptional flows:</b> 2a Invalid card: The <i>Invalid Card</i> use case is performed 2b Unpaid fine: The <i>Pay Fine</i> use case is performed 2c Book overdue: 2c1 – The system displays a message displaying the book overdue, the days overdue, and whether a fine must be paid	

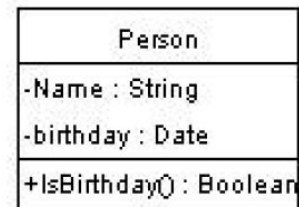
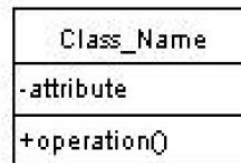
## Class Diagram

# UML notation for class

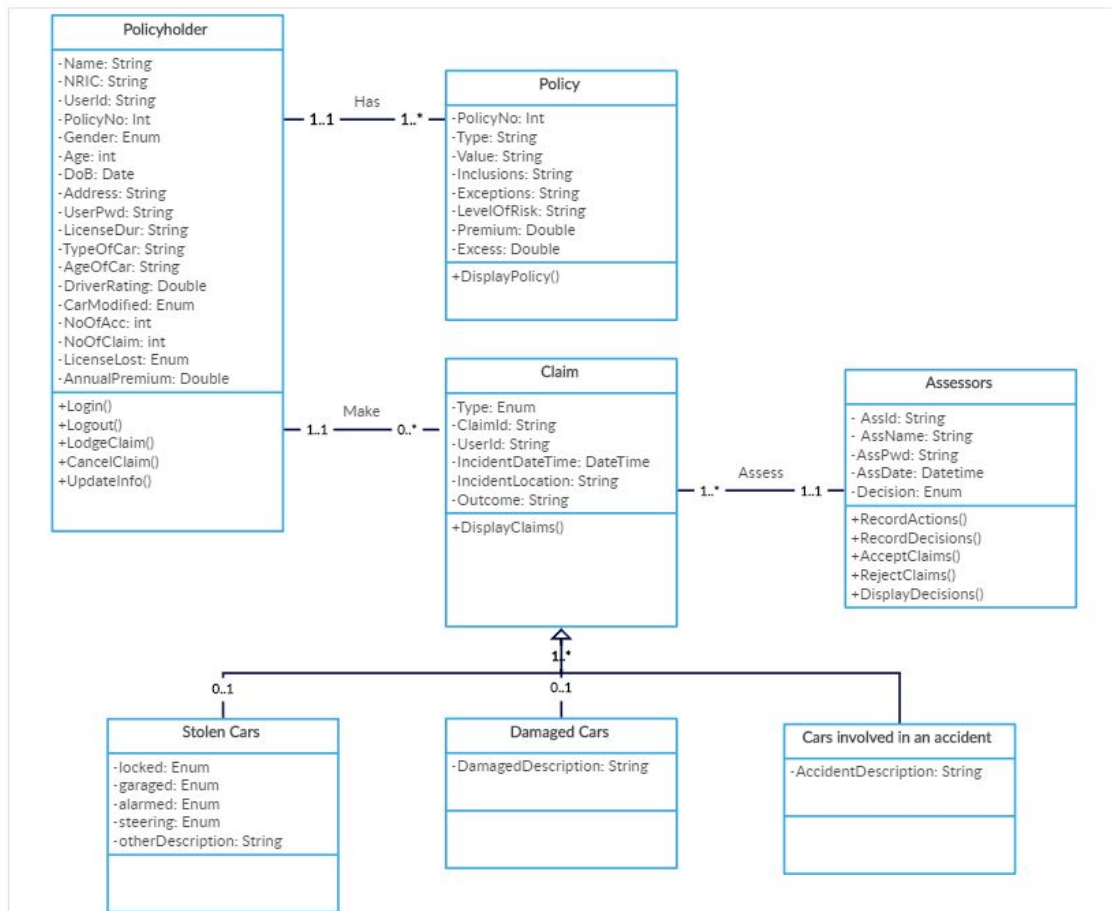
- There are three boxes
  - Top box: Class name
  - Middle box: Attributes
  - Lowest box: Operations



- Attributes/methods are prefixed with:
  - Private
  - + Public
  - # Protected



- All the boxes are optional



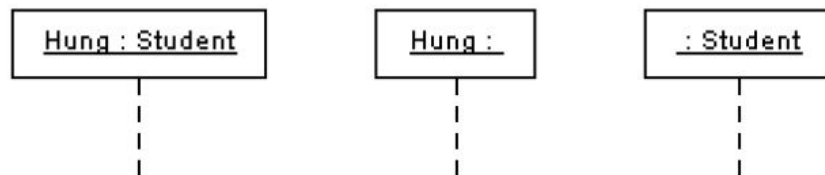


# Sequence Diagram

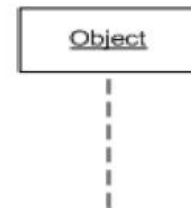
---

## Objects

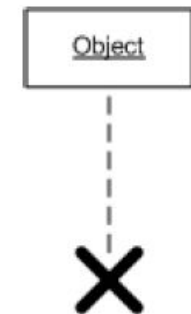
- Shows instance name and/or class name  
(instance\_name:class\_name)



A lifeline shows the life of an object

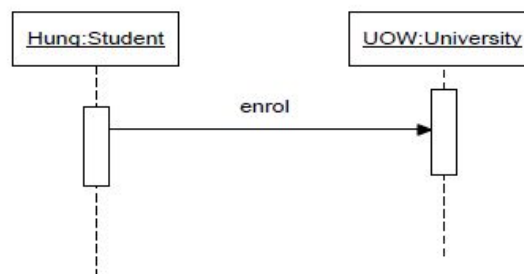


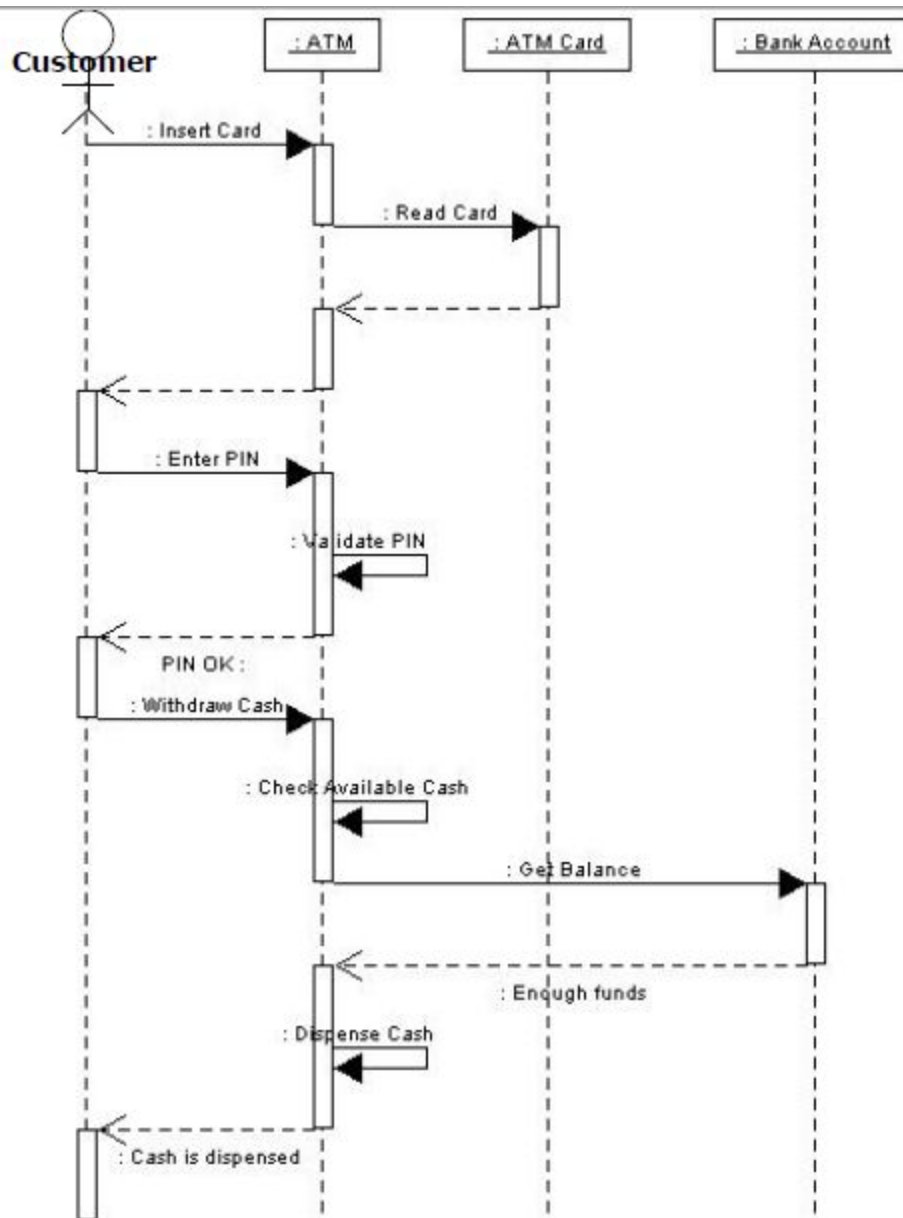
The X shows the point at which the object is deleted



## Simple message

- Shown by a line with a filled in arrow head





---

Sequence  
Diagram showing  
Withdraw Cash  
use case



## Elements in activity diagrams

### □ Initial node

- (Complex activity diagrams can have multiple initial nodes)



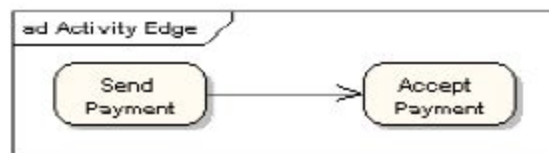
### □ Action

- Incoming activity edges
  - control flow and data flow from other nodes.
  - Action starts when all of its input conditions are satisfied.
- Outgoing activity edges
  - When action completes it sends its outputs via outgoing edges to successor nodes

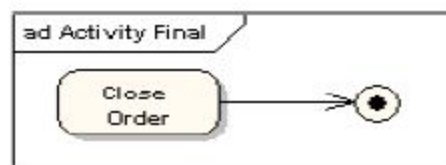
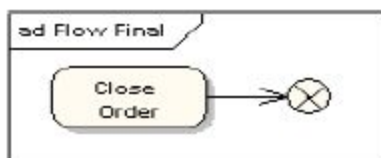


## Elements in activity diagrams -2

### □ Control flow



- Final nodes – two kinds “flow final” “activity final”  
(flow final terminates one parallel flow, activity final implicitly terminate all flows)



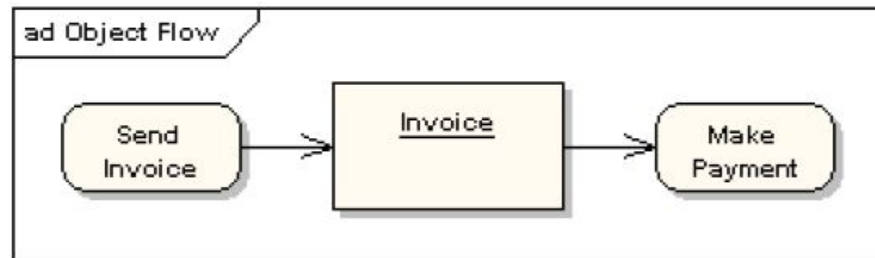
## Elements in activity diagrams -3

### ❑ ObjectNode

- Rectangle named with object type
- Indicates such an object becomes available at some point in activity

### ❑ ObjectFlow

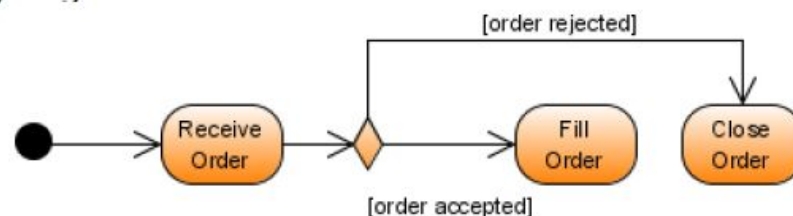
- An object flow is a path along which objects or data can pass.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.



## Elements in activity diagrams-4

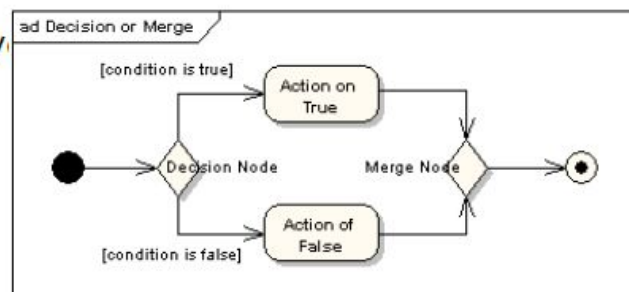
### ❑ DecisionNode

- A decision node is a control node that chooses between outgoing flows.



### ❑ MergeNode

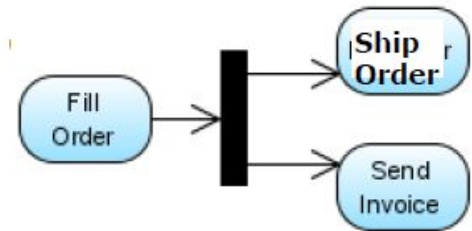
- Rejoin the alternative decision node



## Elements in activity diagrams-5

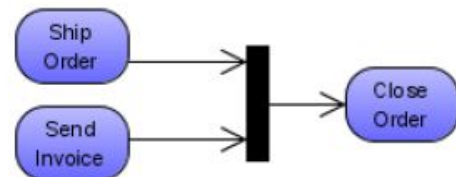
### ■ ForkNode

- A fork node has one incoming multiple outgoing edges.
- Implies parallel execution follows

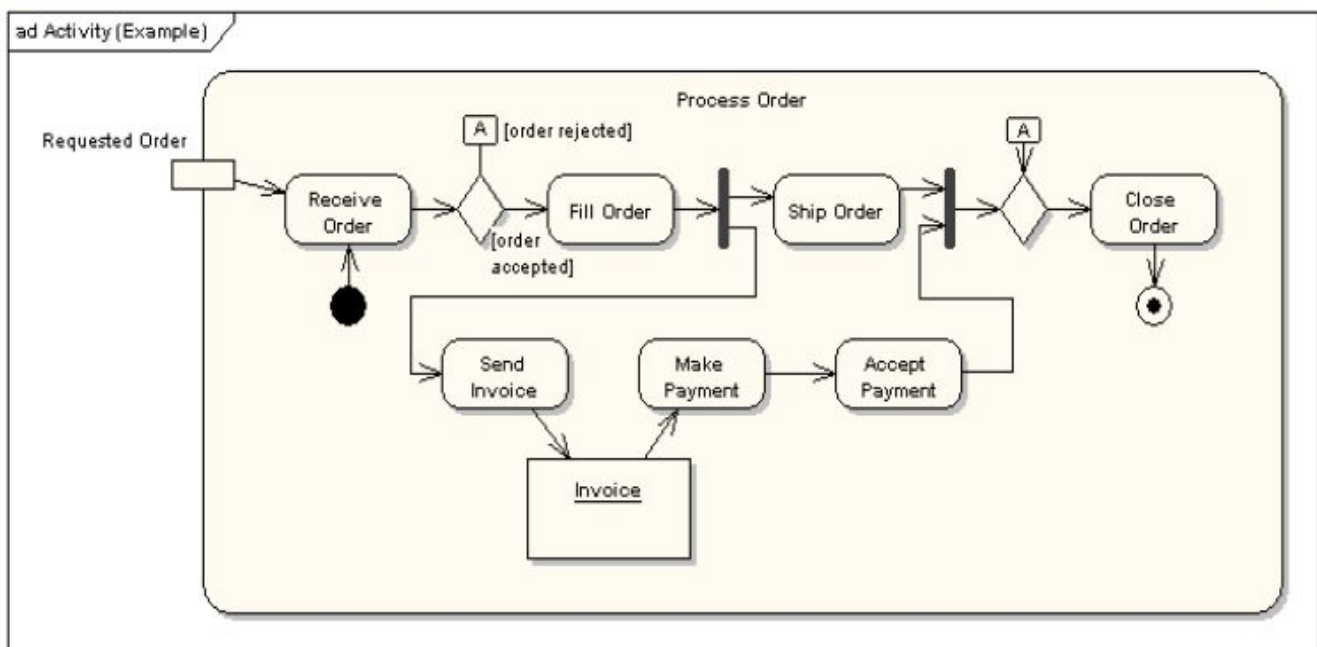


### ■ JoinNode

- A join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received.

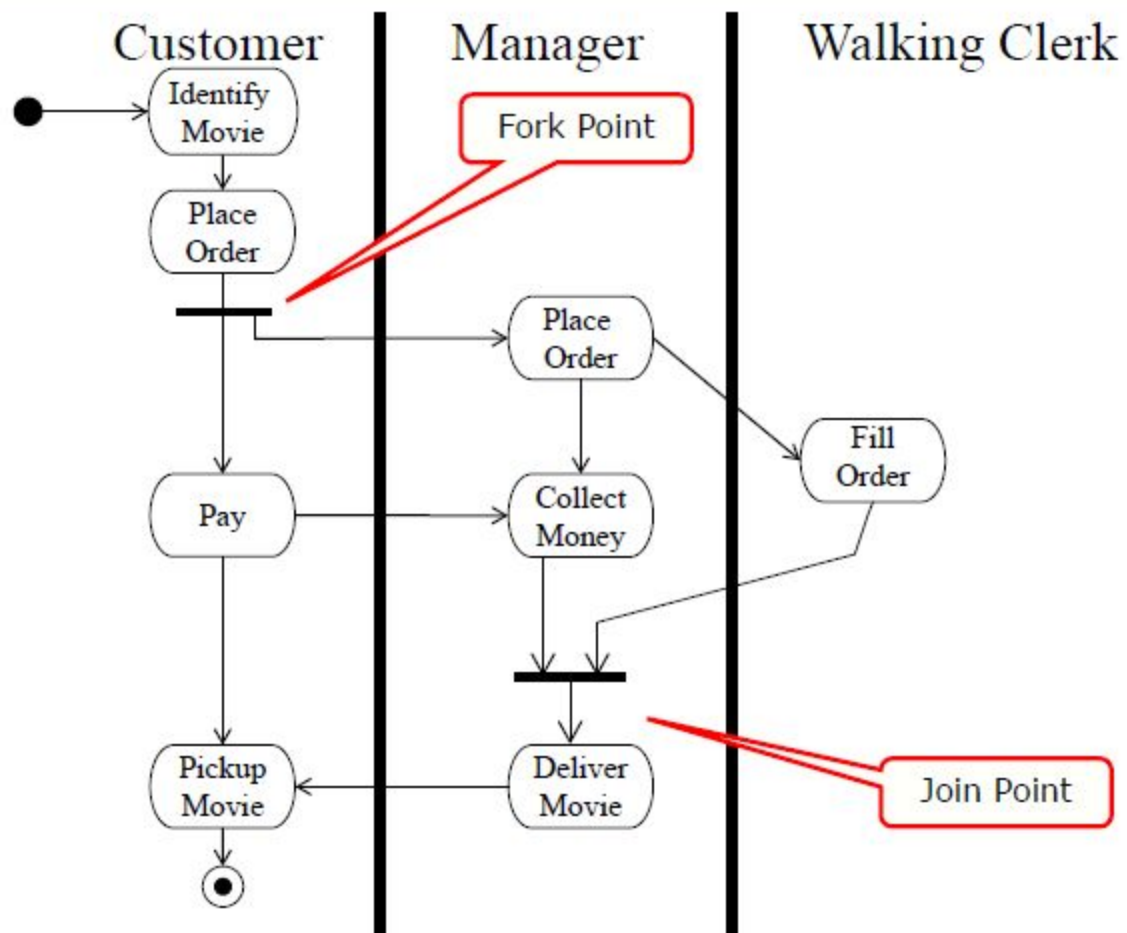


## Activity diagram – example







# Swimlanes and Fork/Join Points

Source: adapted from Easterbrook, 2006





# State Diagram

Term and Definition	Symbol
<b>A State</b> Is shown as a rectangle with rounded corners.	
<b>An Initial State</b> Is shown with a small filled in circle	
<b>A Final State</b> Is shown with as a circle surrounding a small solid filled-in circle. This represents the completion of activity.	
<b>An Event</b> Is an occurrence that triggers a change in state. It is used to label a transition.	<b>Event Name</b>
<b>A Transition</b> Indicates that an object in the first state will enter the second state. Is triggered by the occurrence of the event labeling the transition. Is shown as a solid line.	

Draw a state diagram for a tape recorder

