Introduction to Software Engineering

SE-110

Software and Software Engineering

Lecture # 1, 2, 3
21,22, 23 Jan

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- Administrative Stuff
- Overview of 110
- Introduction to Software Engineering
- Why study Software Engineering?

About me

- Graduated from FJWU
- Completed Masters from NUST
- Research interests:
  - Software engineering
  - Database systems
  - Algorithm analysis

Office hours and meeting rules

- Office Room 6 (CS building)
- Office hours: 3:00 to 4:00 pm
  - (Monday, Wednesday, Thursday)

About course

- Study and application of the principles and techniques of computer science, engineering, and mathematical analysis to the design, development, testing, and evaluation of the software and the systems that enable computers to perform their many applications.
- This is perhaps the most important course in your CS curriculum in CAREER PATH FORECAST.

Career Forecast

- Software engineers are in demand in not only at software development companies but also in all other organizations that are involved in the development of significant information systems – including
- governments,
- telecommunications companies,
- the chemical industry,
- the bio-medical industry,
- financial institutions,
- pharmaceuticals,
- healthcare sector corporations,
- engineering and manufacturing firms,
- e-commerce,
- In fact in the words of noted software engineer, David Parnas, "career opportunities for software engineers are essentially unlimited."

Knowledge assumed

- Programming language concepts

Skills assumed

- We assume you have the skills to code in programming language therefore you
  - can design, implement, test, debug, read, and understand the programs.

Tentative Mark Distribution and Grading Policy

- Assignments: 10%
- Quizzes: 10 %
- Mid Exam 1: 15 %
- Mid Exam 2: 15 %
- Class Participation: 3 – 5 %
- Final: 40%
- Presentations / Projects: 10%
- ***Absolute Grading Policy***

Project

- An advance project
  - Provides interesting problem, realistic pressures, unclear/changing
- Small 3-4 member teams
- Emphasis on good SE practice/methodology
  - Homework's and deliverables tied to the project
  - Grading: practices more important than the end product

Course Ethics

Projects/Assignments

- Deadlines are always final
- No credit for late submissions
- One student per assignment at maximum
- Project Proposal Submission : 3rd Week
- Project Presentation : Last week

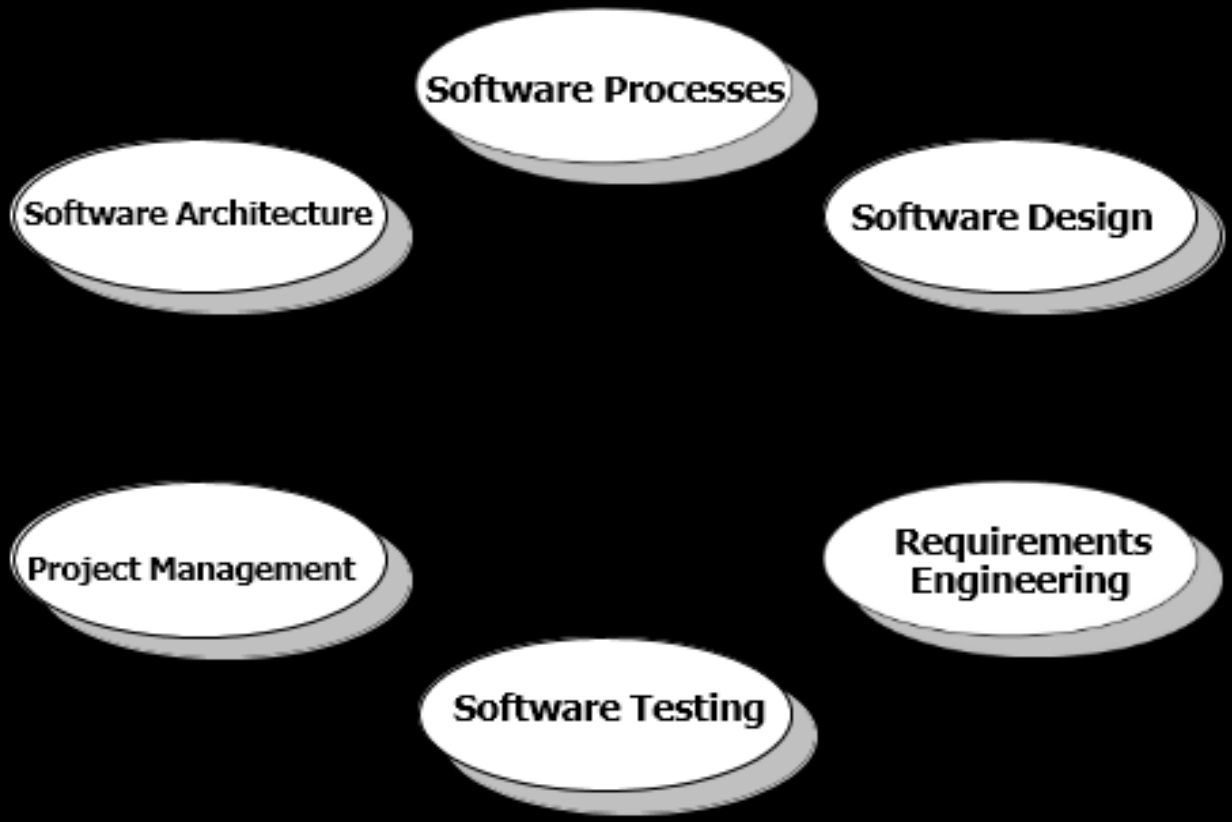
Quizzes

- Announced
- Unannounced

Honesty

- All parties involved in any kind of cheating in any exam will get zero in that exam.

Major Topics of the Course



Course Outline

- Overview of Software Engineering SE
- Process framework and models
- Requirement Engineering
- Requirement Elicitation
- Requirement Modeling
- Software Architecture
- Risk Management
- Software Quality
- Testing
- Project Management
- Work Breakdown structure

Course Material

- You will have Presentations of each topic and ***reference books*** in PDF format will be available at slate.
- Text Books
  - Ian Sommerville, Software Engineering 10th Edition
  - Pressman, R S Software Engineering: A Practitioners Approach (7th Edition, European Adaptation), McGraw Hill, 1994
- Reference Books
  - (Undergraduate Topics in Computer Science) Pankaj Jalote - A Concise Introduction to Software Engineering-Springer (2008)

Course Goals

- Have a sound understanding of the fundamental concepts of the software engineering paradigm
- Understand and apply different practices used in software industry for software development
- Recognize the risks of software failure
- Verify through review practice accuracy, ambiguity and completeness of a software
- Develop a software engineering mindset

Why Study Software Engineering ?

What is Software?

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information and

(3) documentation that describes the operation and use of the programs.

What is Software?

- *Software is developed or engineered, it is not manufactured in the classical sense.*
- *Software doesn't "wear out."*
- *Although the industry is moving toward component-based construction, most software continues to be custom-built.*

Wear vs. Deterioration

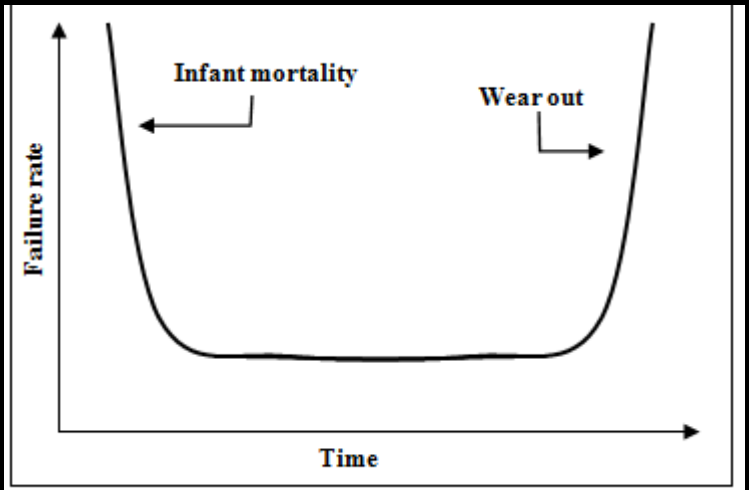


Figure 01: Failure curve for hardware

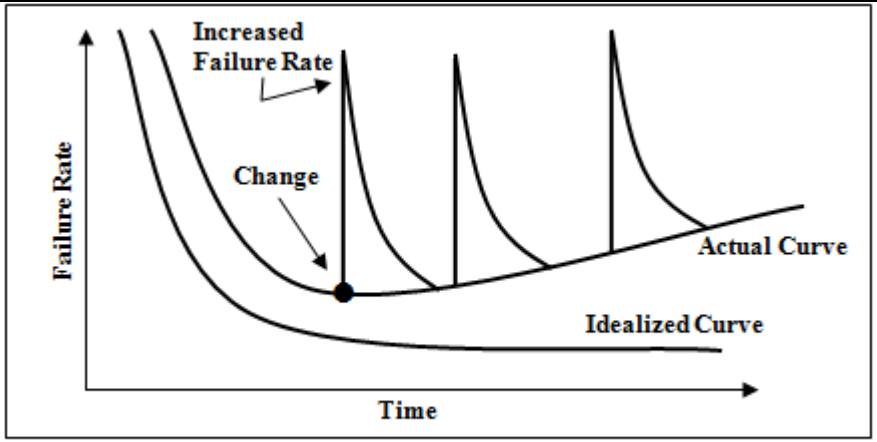


Figure 02: Failure Curves for software

Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

Software Application Domains

1. System software:

   collection of programs written to service other programs. such as compilers, editors, file management utilities

2. Application software:

   stand-alone programs for specific needs.

3. Engineering/scientific software:

   Characterized by “number crunching”algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc

Software Applications

4. Embedded software:

resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)

5. Product-line software:

focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)

6. WebApps (Web applications)

network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

7. AI software

uses non-numerical algorithm to solve complex problem. E.g Robotics, expert system, pattern recognition game playing

Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Netsourcing—the Web as a computing engine
- Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also …
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Why study Software Engineering ?

- We will try to understand . .
  - Typical ***Engineering approach***
- How software development ***is different*** . .
  - from ***other engineering*** disciplines ?

Engineering Approach

Before discussing this we need to understand difference between **science** and **Engineering**

Difference b/w Science & Engg

- **What is Science ?**
  - Scientist invented a ***wheel***,
  - ***Engineer*** will use that wheel for bi-cycle, motor-car, for anything.

  - Scientist invented an ***electric motor***,
  - ***Engineer*** will use that motor to construct things like pumps, electric-lifts, toys, etc.

- **What is Engineering ?**
  - Process of ***productive use*** of scientific knowledge is called engineering.

Difference b/w CS & SW-Engg

- ***Computer science*** is knowledge contains theories and fundamentals.
- Somebody has invented followings
  - Database
  - Design methodology
  - Algorithms
  - Testing methods
  - Methods of analysis and Verification.

- ***Software Engineering*** is a approach of applying engineering principles in order to develop software.
- SW-Engg applies ***various phases*** to implement all of the components needed to satisfy the user requirements.
- SW-Engg follows engineering which is to ***plan,*** with ***suitable skills*** and craft for moving into Mass-production.

Engineering Approach

- Day to day life we see successful projects in other fields of engineering
  - Such as in civil, or mechanical engineering.
  - Building bridges & roads are also large projects and the carryout large efforts in construction.

- It is our common observation that usually large software projects are NOT very successful.
  - In the middle of software projects we realize that **more time** is required or
  - **more resources** are required to complete the project, than we initially anticipated.

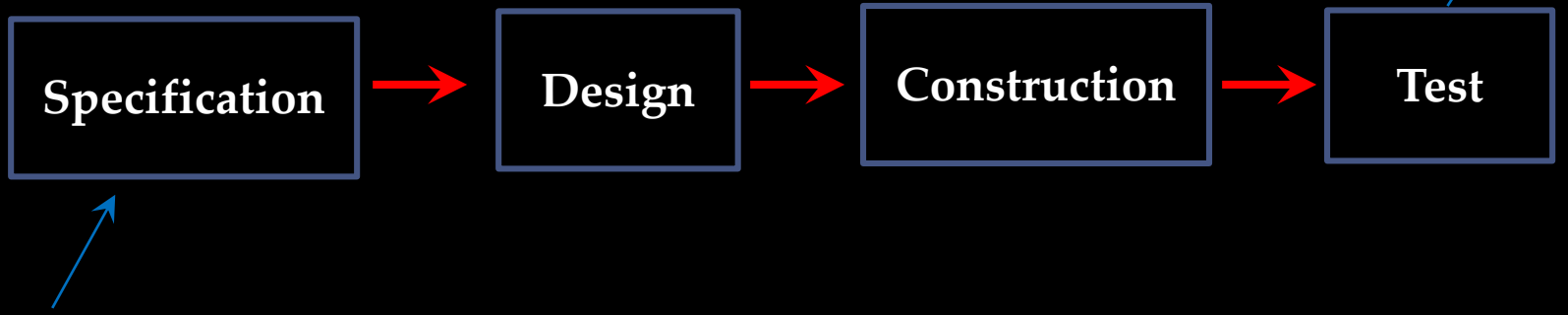
Engineering Approach

- Following are ***Large*** engineering projects ***in other discipline*** and are normally successful
  - Building bridges & roads
  - Power Plants
  - 60 story luxury apartments
  - Aircraft missiles
- Why such kinds of project are generally successful?
- What are the ***practices being followed*** in these kinds of project that make them successful?
- Is there any ***common attributes*** or these are different than developing a software?

Engineering Approach

Using this approach →

- Construct a house
- Construct a bridge
- Or design a CAR



This ***engineering approach*** helps engineers developing what they intend to and complete projects successfully

Example – Construct a house

Following are logical steps to develop a house

- Stable and *detail requirements* from customer.
- ***Record*** requirement so that we ***do not miss*** any requirement.
- After recording it must be *validated* by user again.
- Any ***Change*** means additional cost and time.

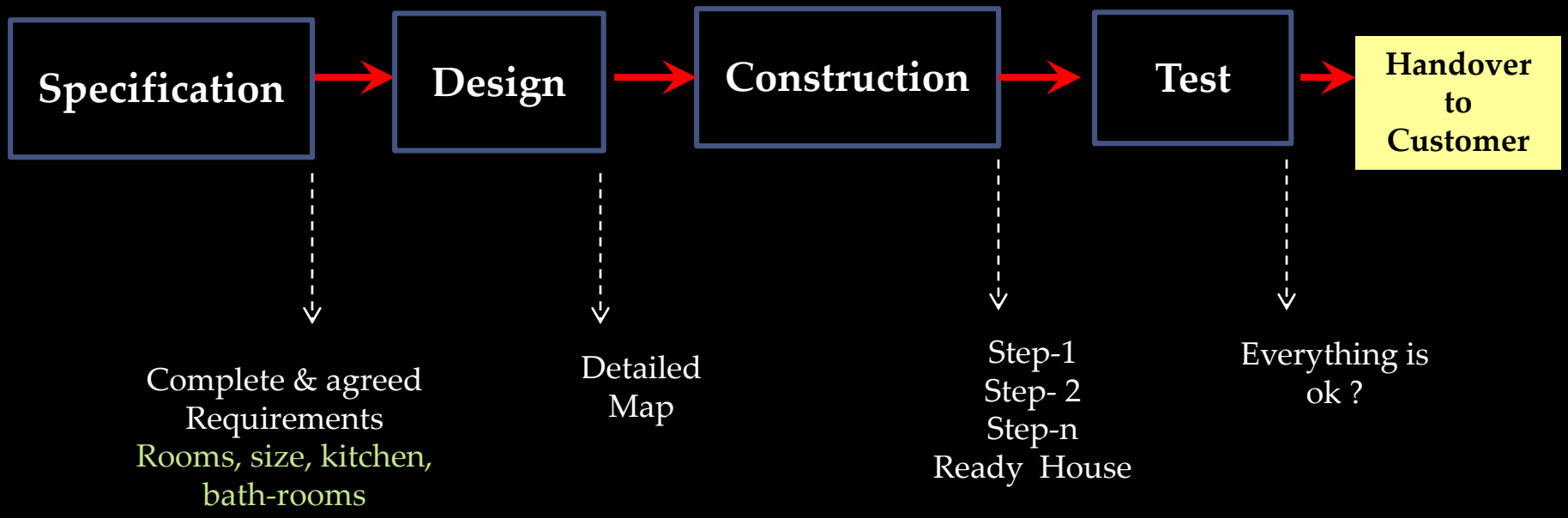
- Once the *requirements are agreed* (baseline) next step we develop ***a map*** (design)
  - Size of rooms
  - Doors and windows placement
  - Size of kitchen, bathrooms etc
  - Design should have electrical cabling layout
  - plumbing layouts etc

Example – Construct a house

- Once the map is *approved* by the customer you ***start*** construction of house.
- ***What if*** you developed the structure of the house and customer request you to ***change*** dimensions of rooms?
  - Can you accommodate that change at this stage?
- How do you manage agreed (scope) map?

Example – Construct a house

How Engineering approach helped **construction** of **House** ?



Software Engg is different

Software Engineering is different

- Requirements are complex and ***multi-dimensional***.
- Difficult to asses *progress* of the project.
- Delivering an ***intangible*** product.
- Software ***get obsolete*** quickly.
- Can be developed using ***different approach***
  - Approach = process, templates, techniques etc
  - Plan-driven (Predictive) & change-driven (adaptive)

Requirements are complex

Components of any automation Solution

- Features and functions
- Process/Practices (need to be automated)
- People (their experience, knowledge, and ability to adopt change)
- Usability (Desktop, Internet, Smart-phone)
- Environment (Temp, Dust-free etc)

Guess my requirements

I need a equipment

- That is equipped with internal combustion engine (such as Car, truck)
- That is available in yellow & black colour
- It has three wheels
- Has 3 speed forward and 3 speed reverse
- Levels uneven surface nicely and quickly

Difficult to asses progress

- You can asses progress in civil engineering
  - **50%** work is completed (in 60 story building)
  - You may verify that by visiting the project site and see 30 stories are built or not
- When Project Manager of software project says 90% work complete.
  - How can you verify that statement **?**
  - Is it really 90% or much less . . .

Delivering an intangible thing

- In civil engineering you are delivering house or a road or a bridge.
  - Can understand the time taken and cost
  - Buy a Car you look for rooms, their size, design etc

- In software project what do you deliver ?
- A CD only cannot asses ***how much efforts*** it had taken and ***what is its cost***.

Can differentiate - 2 kilo potato or 20 Kilos
Cannot differentiate - Small SW or large SW

Software get obsolete quickly

- Software is developed to ***automate business process***, as business grows its process changes.
- Need some changes approximately after very 8-9 months in developed software.
- This is ***not that case*** when you purchased a ***brand-new car*** or new house.

Can be developed using different approach

- Software house use different approaches to develop same functionality
  - Plan-Driven (Sequential or Predictive)
  - Change-Driven (Agile or Adaptive)

- Building a 60 story cannot be build with ***steel***, ***cement*** and ***concrete***. Approach is not very different in other engineering discipline.

Software Engineering is different

- It is difficult to estimates efforts and plan these activities for software development.
- There are so many failures in software development projects.
- It is NOT difficult to estimate efforts in construction of 50-story building.

Software Engineering is different

- We must have, separate specialized team for . . .
  - Requirements elicitation
  - Dedicated team for software design
  - Team of developers
  - separate team for testing etc.
- All these team needs supervisions and monitoring for the work they do.
- Typical engineering approach is necessary for the successful software project.

Software Engineering –

Definition

Software Engineering as defined by **IEEE**

Application of a systematic,
disciplined, quantifiable approach
to the development, operation,
and maintenance of software;
that is, the application of engineering to software."

Why study Software Engineering ?

Why study Software Engineering ?

- What are ***challenges*** in software development ?
  - that will be addressed by using Engineering approach

- What is the need/required ?
  - Unable to handle large software projects - Why
- Why are we studying software Engineering?

Why study Software Engineering ?

- Historically 2-3 people used to develop ***small*** and ***simple*** software that is ok. No problems in small & medium size projects
  - Example: your FYP is small and simple project

- When a ***large*** software with ***complex*** requirements are developed through a large team (70-80 people) we had serious problems (called *Software crisis*).
  - Example: Developing a core banking software is large project, will require large team and few years.

Software Crisis

- In early ***60s*** software techniques that were used to develop ***small*** software were ***not applicable*** for ***large*** software applications.
  - In most of the cases that software which was tried to be build using those old tools and techniques were either ***not complete***.
  - Most of the times it was delivered ***too late***.
  - Most of the projects were ***over-budgeted***.
  - A few projects caused loss of life
  - Some projects caused property damage.

Cost of Failure

- ***Allstate Insurance*** – (www.allstate.com) in 1982
  - **$8M** were allocated to automate business
  - EDS providing software
  - Initial *5-year* project continued for *10-years*, until 1993
  - Cost approached $100M

- ***Bank of America*** – (www.bankofamerica.com)
  - Spent **$23M** on an initial **5 year** accounting & reporting system
  - Spent $600M trying to make it work
  - Project was *cancelled*
  - Lost customer accounts - $Billons

Cost of Failure

- ***Blue Cross and Blue Shield of Wisconsin*** – 1983
  - EDS was hired to build ***$200M*** computer system
  - Delivered on time in 18 months
  - System didn't work – issued $60M in overpayments and duplicate checks
  - Blue Cross lost 35,000 policy holders by 1987

NATO Conference

- In the fall of 1968 and again the in fall of 1969, NATO hosted a conference devoted to the subject of software engineering.

- The motivation for these conferences was that the computer industry at large was having a great deal of trouble in producing large and complex software systems i.e. the software crisis.

- Participants were solicited from computer manufacturers, computer users, software houses, and academia. Over the years, the conference reports have gained a certain amount of classical aura.

NATO Conference

- In late 1968 the Conference recommended the term Software Engineering. The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.

Software Engineering

- Writing software has evolved into a profession concerned with
  - How best to maximize the quality of software and of how to create it.
  - How best to create high quality software is a separate and controversial problem covering software design principles
  - How best to deliver software on time and as quickly as possible, work-place "culture", hiring practices, and so forth.

All this falls under the broad rubric of software engineering

That is all

Introduction to Software Engineering

SE-110

SDLC

Lecture # 4, 5, 6
28,29, 30 Jan

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- Process and Product
- Software Engineering Framework
- What is system?
- SDLC
- SDLC Phases

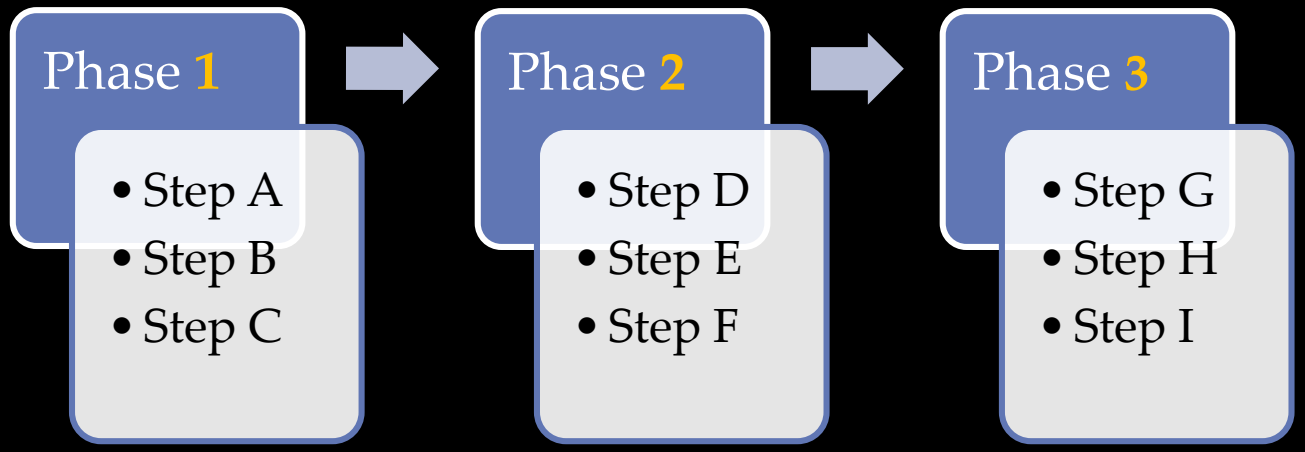
Process

Process: A particular method, generally involving a number of steps.

Process for making ***potato chips***

Process

- Process is generally a set of ***phases***.
- Each phase performs a well ***defined task*** and generally produces an output.



Software Process

- ***Software Process***: A set of steps, along with ordering constraints on execution, to produce software with desired outcome.
  - Many types of ***activities*** performed by ***different teams***
  - Software process is comprising of many component processes

Process

What we should do, to develop a *good quality* software ?

What are the attributes of a *good process* ?
Visible, Repeatable, Measureable

Can you give examples of *Technical* **and** *Managerial* problems in software development process ?

Project fails due to **Managerial** problems

Product

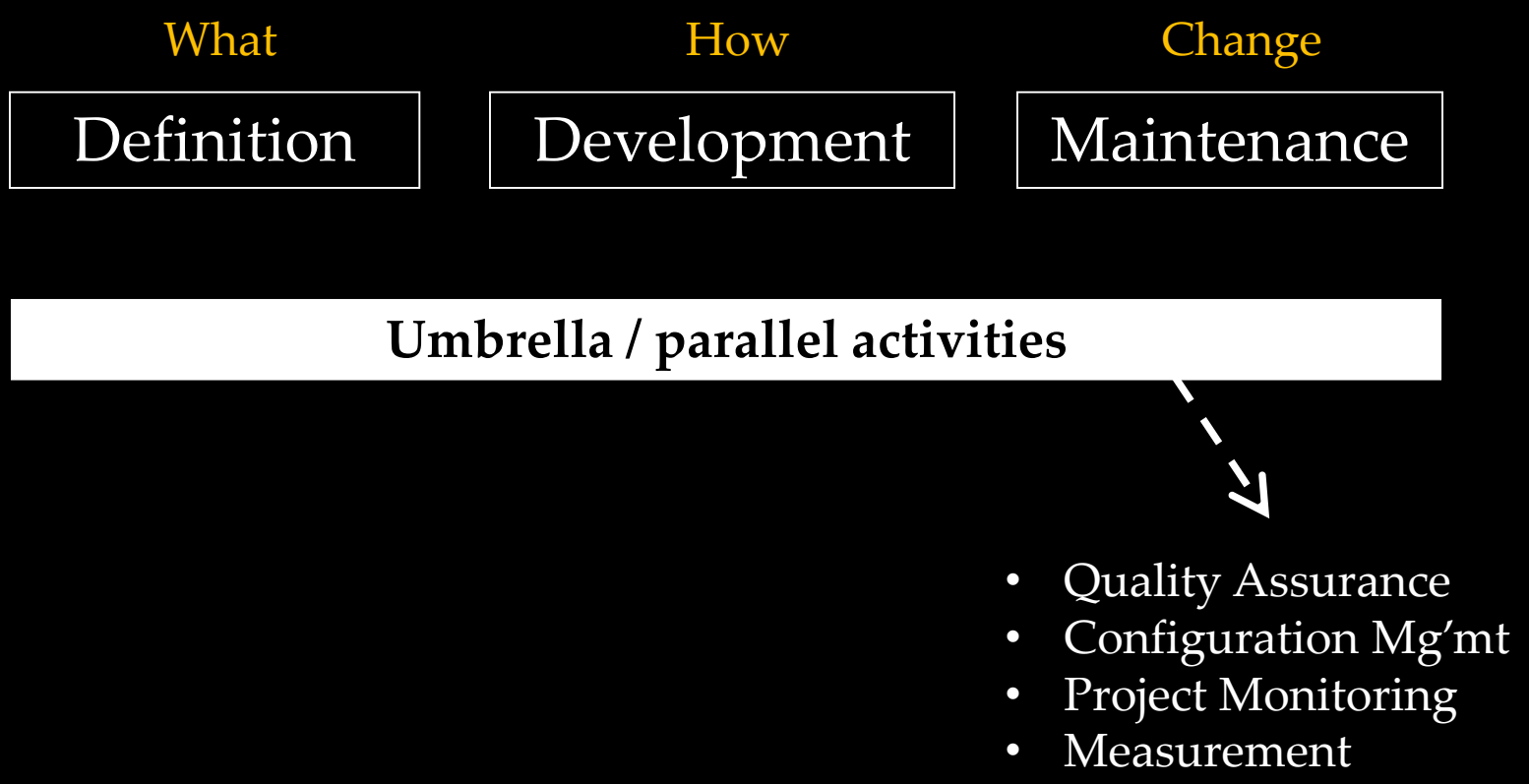
- ***Products*** are outcomes of executing a ***process*** for a project.
- Does Process quality and Product quality has any relation ?
- Software development life cycle (SDLC), is a structure imposed on the development of a software product.
- Software Engineering says if you follow the process the output is predicable and repeatable no matter who does that.

Software Engineering Framework

Software Engineering Framework

- What is ***framework*** and why we need framework?
  - Framework means; set of rules to be followed.
- What are ***those rules***? Those rules have been ***adopted*** by organizations that produce good results.
- Experts convert those rules into a framework to be used by every organization with respect to their needs.
- **Example:** Framework for Requirements Development

Software Engineering Framework



Definition Phase

- *Definition* phase focuses on *what* (is required).

- During definition, SW-development-Team and user attempts to identify the following questions:
  - What is ***need*** (or problem)?
  - What *features* are required?
  - What interfaces are to be established?
  - Any budget or technical ***constraints*** ?
  - What is *success criteria ?*.

Development Phase

- *Development* phase focuses on the **how**.

- During development, the SW-development-Team attempts to define how:
  - How database would be designed
  - How software *architectures* would be designed
  - How the design will translate into programming language
  - How *testing* will be performed

Development Phase

- Methods applied during development phase, *will vary* (depending on the SDLC) but the three steps will occur in some form:

- *Design:* Design translate the requirements into some graphical or tabular representations.

- *Coding:* Design is then translated into programming language.

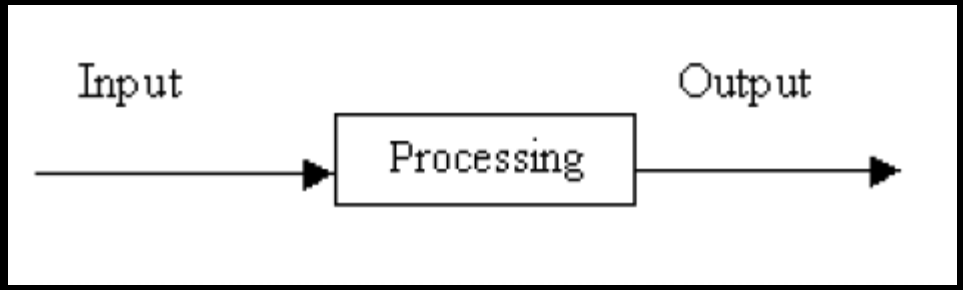
- *Testing:* The executable code must be tested to uncover errors.

Maintenance Phase

- Maintenance phase focuses on changes that associated with
  - Error Correction (Corrective)
  - Platform Adaptations required (Adaptive)
  - Enhancement due to change (Perfective)
  - The work carried out order to avoid any breakdown or malfunction (Preventive)

System

- A collection of components that work together to realize some objective forms a system.
- Basically there are three major components in every system namely input, processing and output.



System

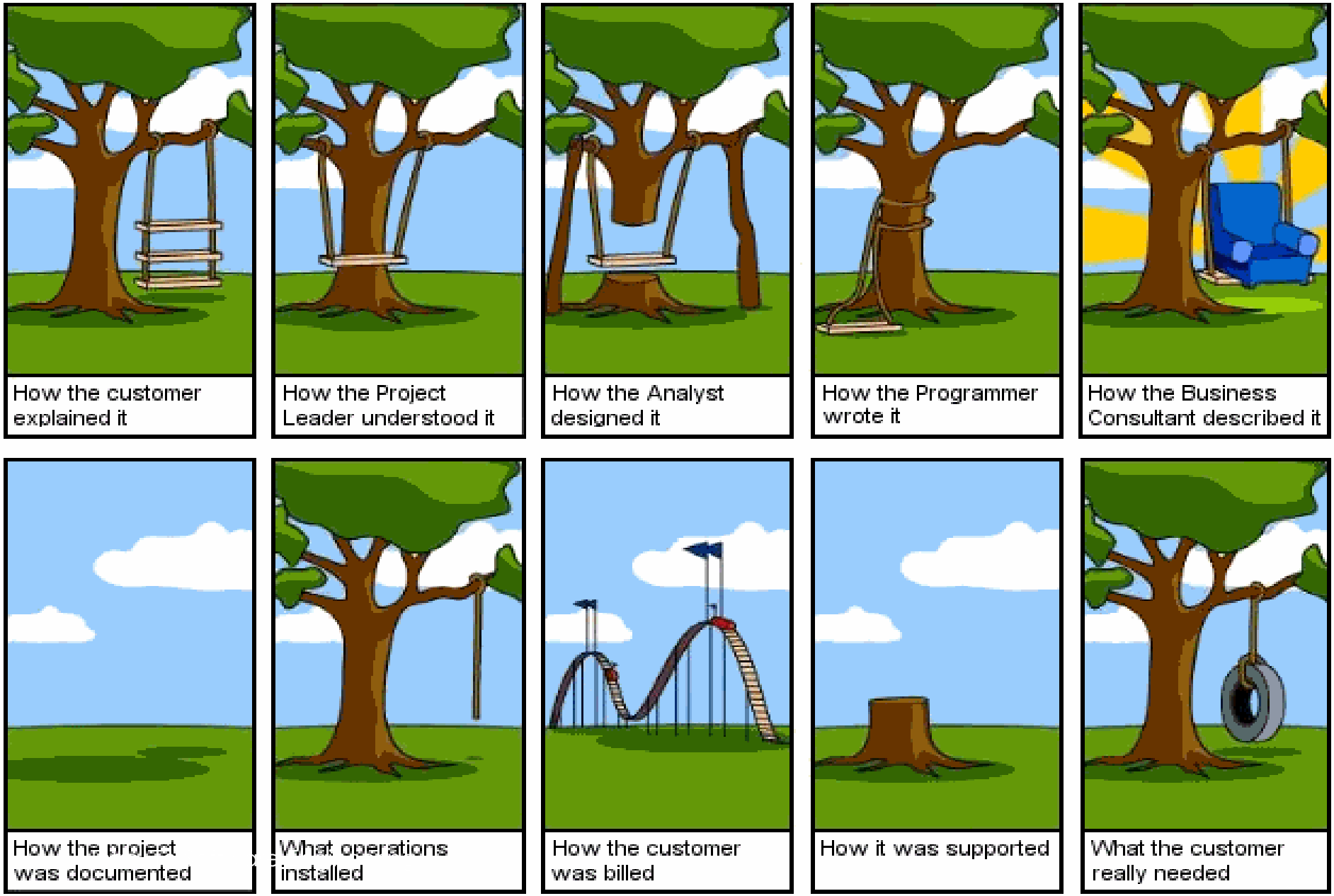
A System can be a Application program or it can be an Information System.

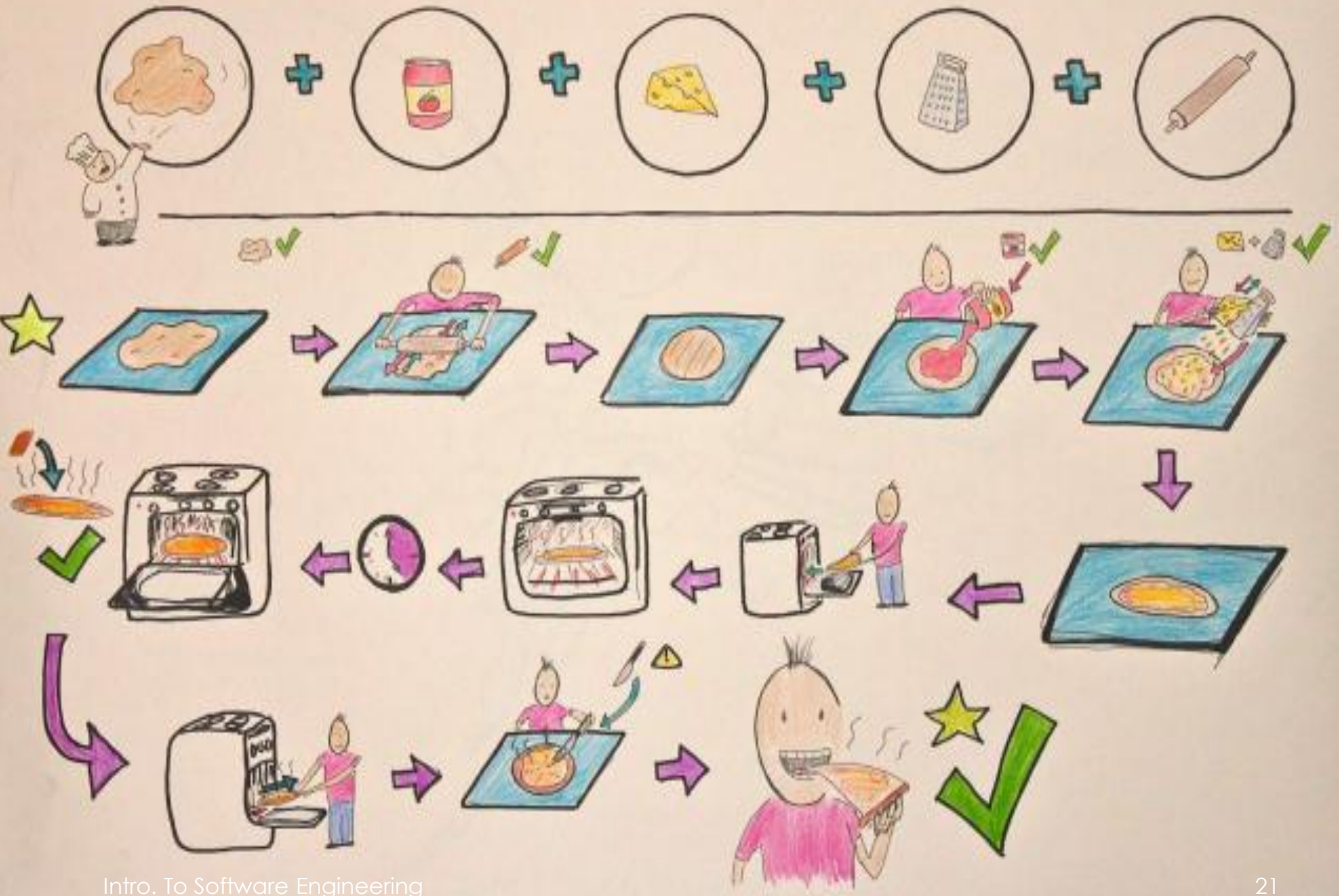
- Computer App: is an application program (app for short) is a computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user.
- Information System: is software that helps you organize and analyze data. This makes it possible to answer questions and solve problems relevant to the mission of an organization.

Software Crisis

Defining the *problem* is the PROBLEM

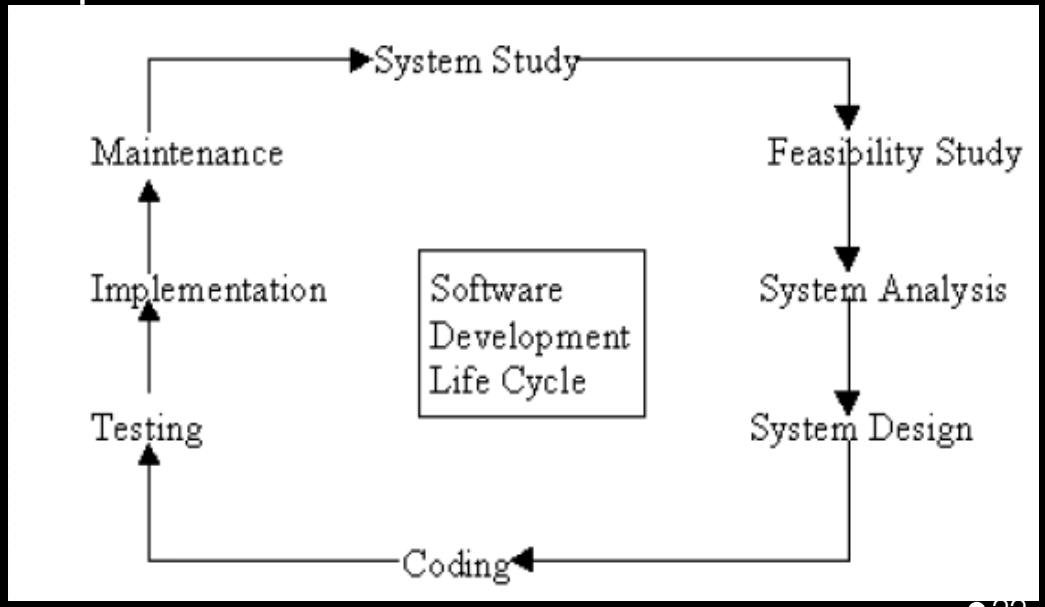
Why Object-Oriented?



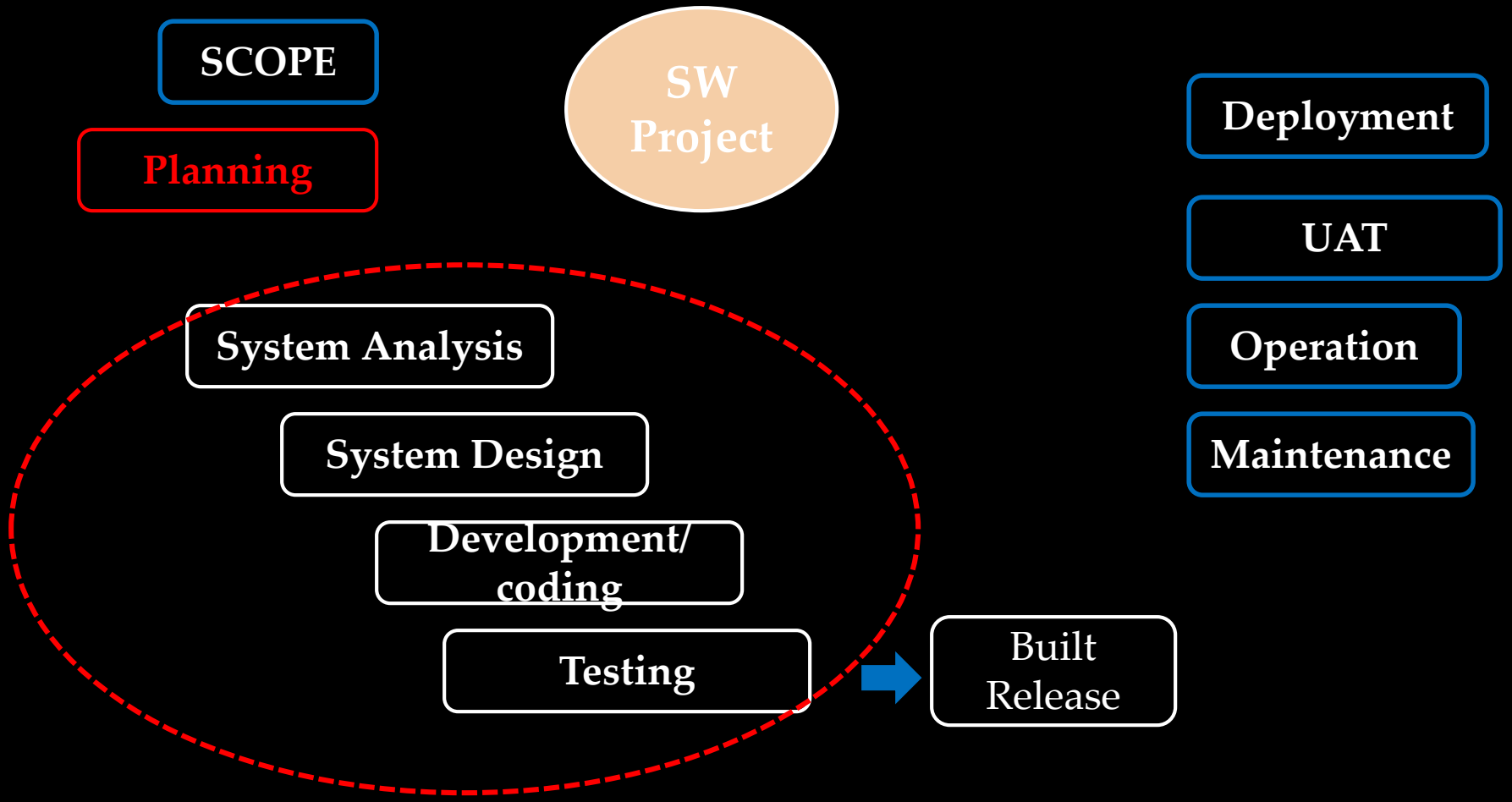


SDLC

- Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software.
- It consists of a detailed plan describing how to develop and maintain software.
- SDLC consists of many activities/ phases.
- Following are the major phases of SDLC.
  - System study
  - Feasibility study
  - System analysis
  - System design
  - Coding
  - Testing
  - Built release
  - Maintenance

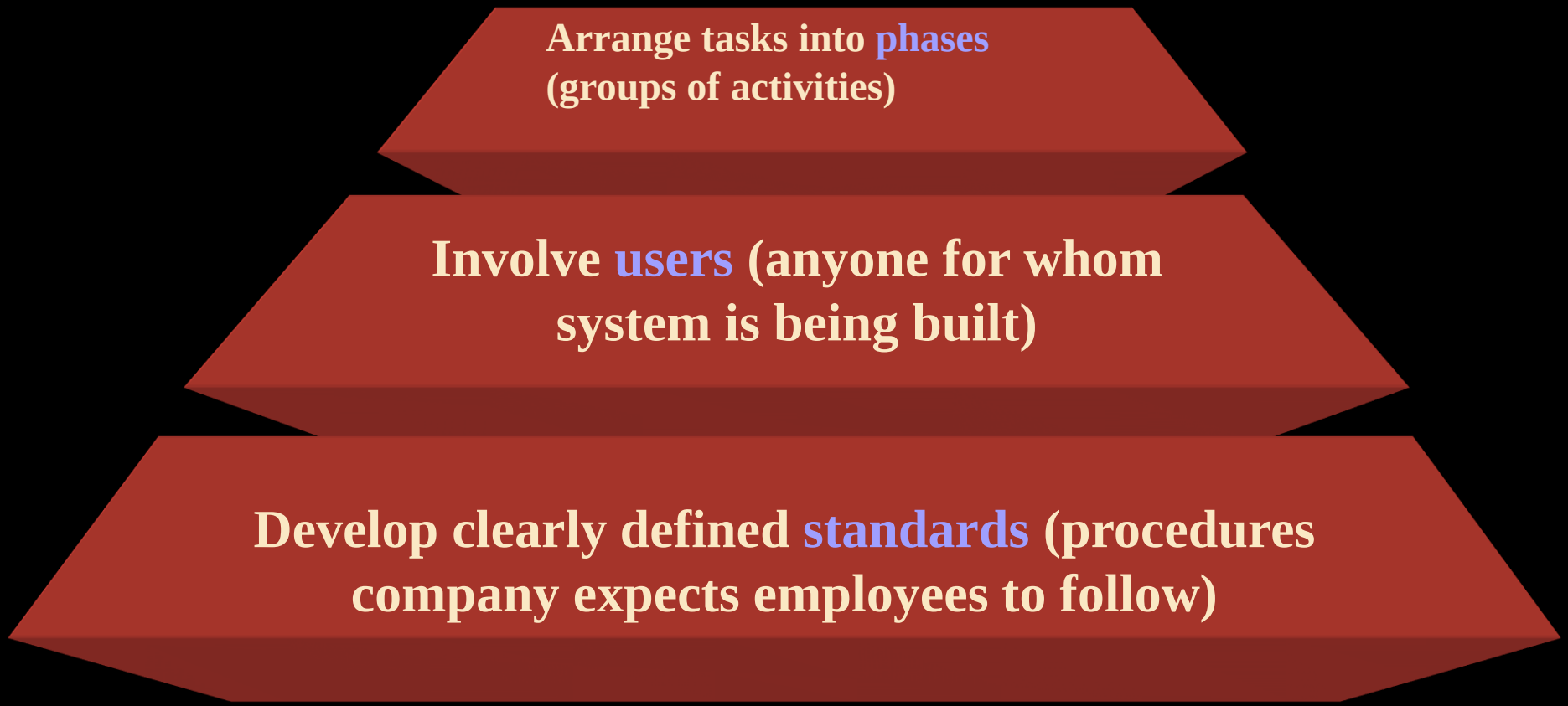


Software Development Life Cycle



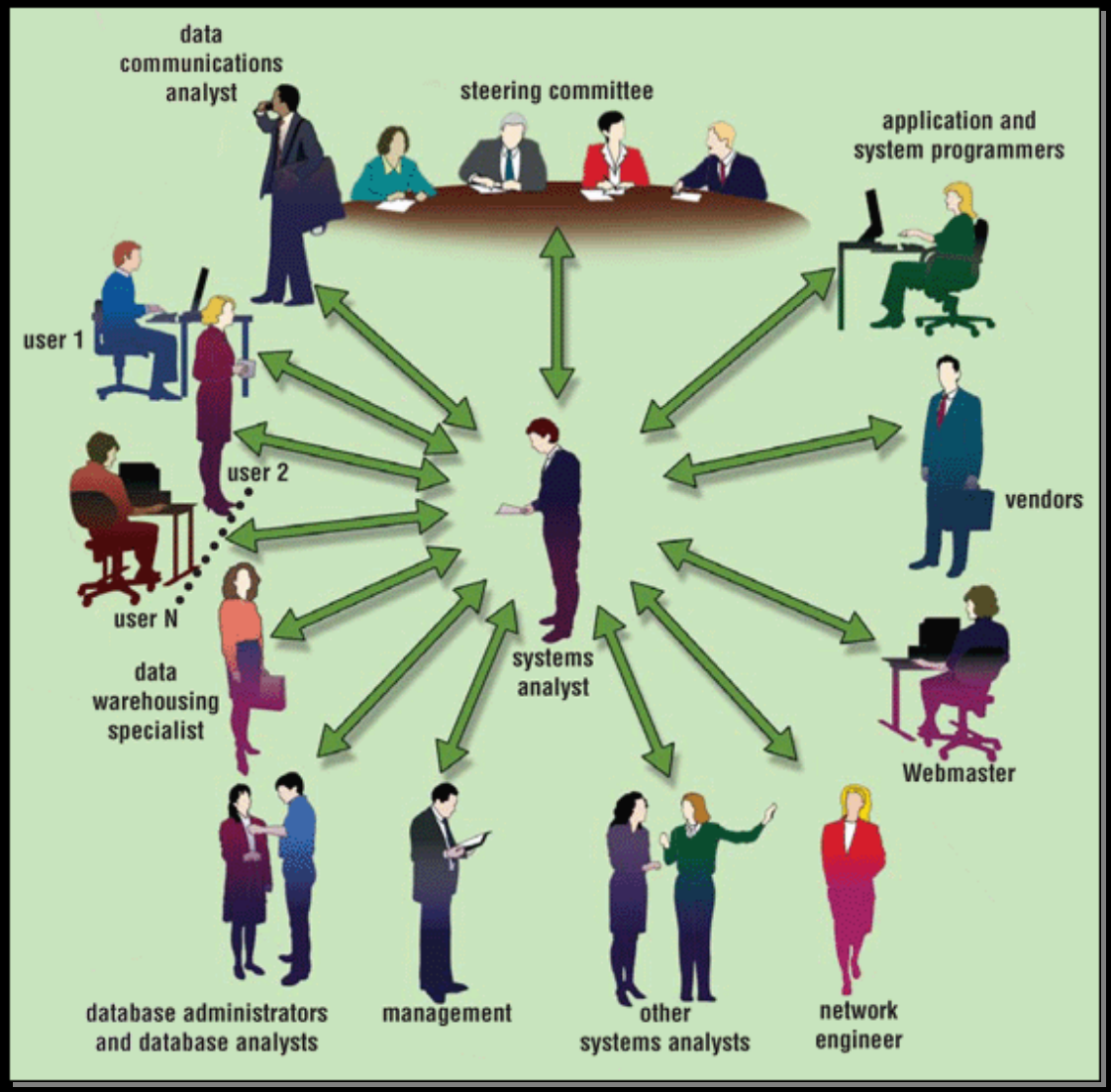
The System Development Life Cycle

What are guidelines for system development?



The System Development Life Cycle

Who participates in the system development life cycle?



The System Development Life Cycle

What is the project team?

Formed to work on project from beginning to end

Consists of users, systems analyst, and other IT professionals

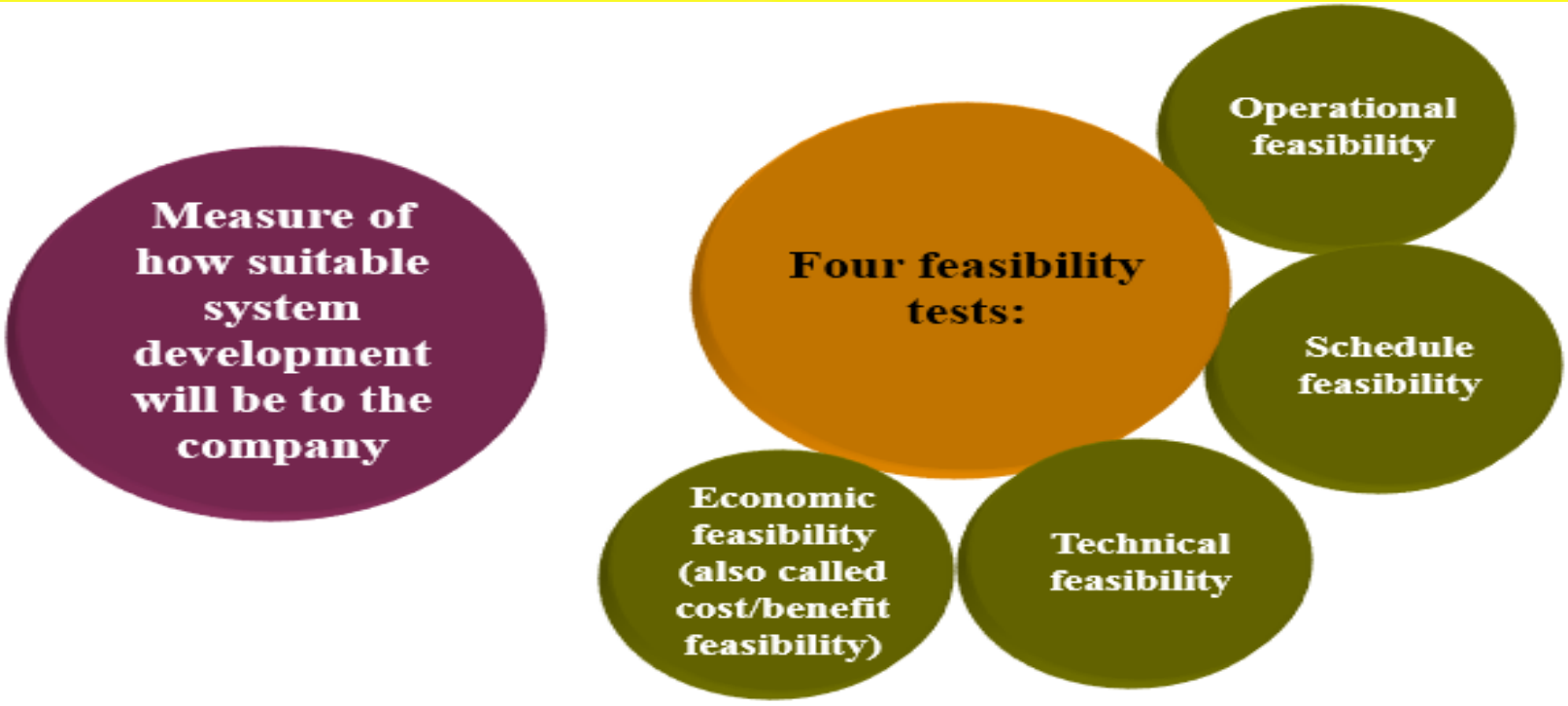
Project leader—one member of the team who manages and controls project budget and schedule

Stage 1: System Study

- Gives clear picture of what actually the physical system is.
- System study phases(I & II):
  - I: initial survey of the system(scope identification)
  - II: depth study of the system (requirement identification, limitation & issues of the current system). It also includes the back ground analysis and inference or findings of the system.
- Output: system proposal or recommendations to overcome the limitations / issues of the current system.

Feasibility Study

An analysis of a proposed project to determine whether it is feasible and should go ahead.



Stage 2: Feasibility Study

- A feasibility study precedes the decision to begin a project. It is an assessment of the practicality of a proposed system.
  - **Operational feasibility:** refers to the measure of solving problems with the help of a new proposed system. It helps in taking advantage of the opportunities and fulfills the requirements as identified during the development of the project. It takes care that the management and the users support the project.
  - Economical feasibility: A project is considered economically feasible when the benefits that will accrue to the broad community are greater than the cost of undertaking the project.
- A feasibility study leads to a decision: go or no-go.
- Output: FSR

Five common factors (TELOS)

1. **Technology and system feasibility**
2. **Economic feasibility**
3. **Legal feasibility**
4. **Operational feasibility**
5. **Schedule feasibility**

1. Technical Feasibility

- This assessment is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project.
  - Is the project possible with current technology?
  - What technical risk is there?
  - Availability of the technology:
    - Is it available locally?
    - Can it be obtained?
    - Will it be compatible with other systems?

2. Economic Feasibility

- Economic analysis is the most frequently used method for evaluating the effectiveness of a new system. More commonly known as cost/benefit analysis, the procedure is to determine the benefits and savings that are expected from a candidate system and compare them with costs. If benefits outweigh costs, then the.
  - What are the development and operational costs?
  - Are the benefits worth the costs?

3. Legal Feasibility

- Determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local Data Protection Acts.

4. Operational Feasibility

- Is a measure of how well a proposed system solves the problems, and takes advantages of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.
  - If the system is developed, will it be used?
  - Human and social issues…
    - Potential labour objections?
    - Manager resistance?
    - Organizational conflicts and policies?
    - Social acceptability?

5. Schedule Feasibility

- A project will fail if it takes too long to be completed before it is useful. Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Schedule feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. You need to determine whether the deadlines are mandatory or desirable.
  - Is it possible to build a solution in time to be useful?
    - What are the consequences of delay?
    - Any constraints on the schedule?
    - Can these constraints be met?

Other feasibility factors

- **Market and real estate feasibility**
- **Resource feasibility**
- **Cultural feasibility**

Stage 3: System Analysis

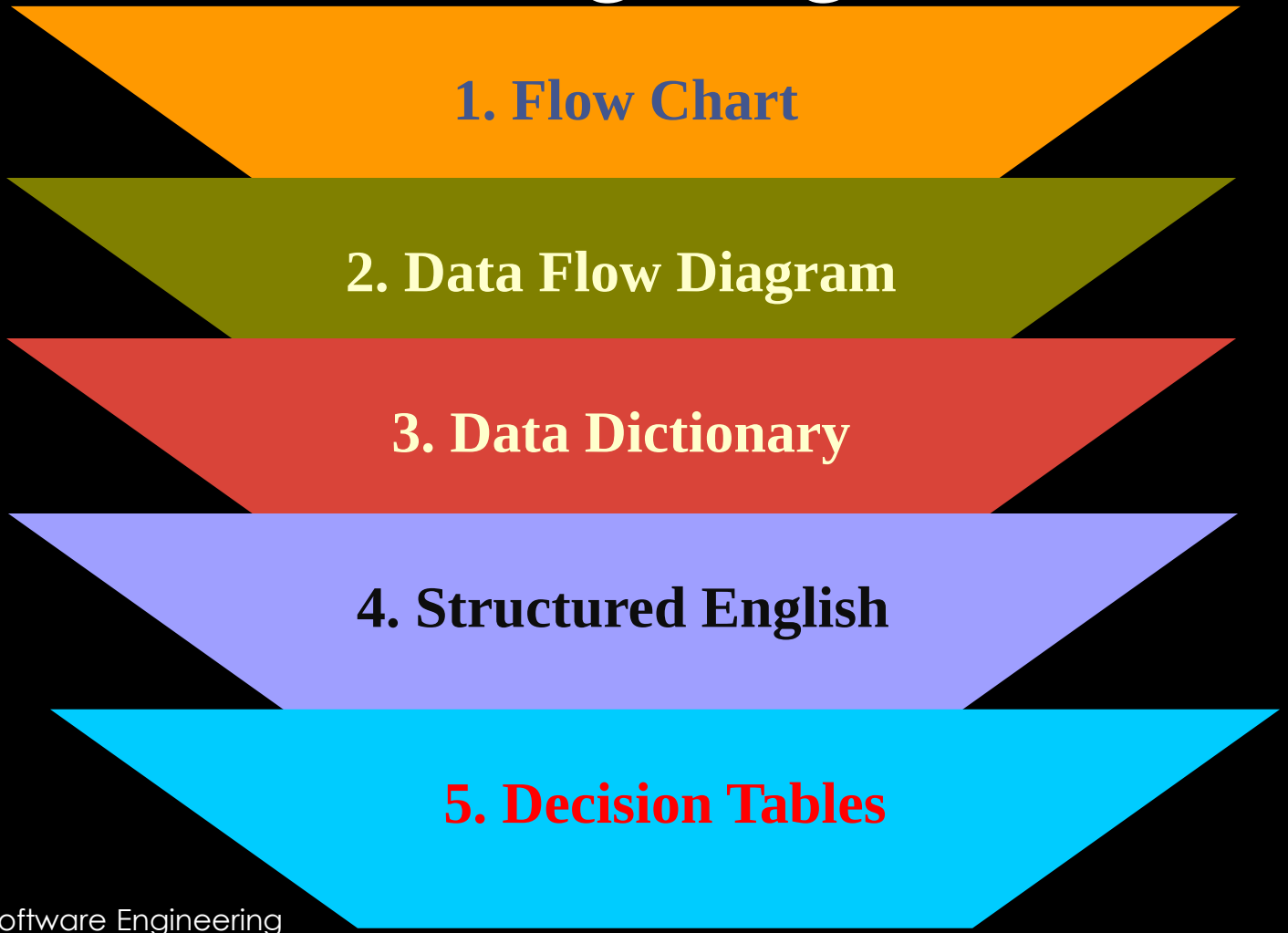
- Forms the basis of agreement between user and developer. System analysis establish the system's services, constraints and goals by consultation with users.
- It is the study of specifications, operations and relationships with in the system and outside the system.
- Specifies what not how. (Hard task)
- Define the boundary of the new system keeping in view the problems and the new requirement.
- Output: is the Software Requirements Specification (SRS) document.

Stage 4: System Design

- A major step in moving from *problem* to *solution*.
- Based on system analysis, the new system must be designed.
- *Two main tasks*
  - *General design:* (preliminary design) components and connectors that should be there in the system
  - *Detailed design: (Detailed Design)* logic of modules

Output: SDS(system Design Specification)

Tools and Techniques for Designing



Flow Chart

- Draw a flowchart to find the sum of 5 numbers
- Draw a flowchart to log in to face book account

Decision Tables

1) List all actions that can be associated with a specific procedure (or module)
2) List all conditions (or decisions made) during execution of the procedure

List of Conditions	Combination Of Conditions
List of Actions	**The Corresponding Set of Actions**

	Rules					
Conditions	**1**	**2**	**3**	**4**	**5**	**6**
Regular customer	T	T				
Silver customer			T	T		
Gold customer					T	T
Special discount	F	T	F	T	F	T
Actions						
No discount	✓					
Apply 8 percent discount			✓	✓		
Apply 15 percent discount					✓	✓
Apply additional *x* percent discount		✓		✓		✓

Decision Tables

Activity Task:

For the SafeHome problem, assume that the system is connected to the network. Write a decision table based on the following facts;-

The homeowner is supposed to get an E-Mail if and only if noise level goes beyond a level. If the temperature goes beyond a level not only homeowner will be getting an E-Mail, but also alarm has to be switched on along with a telephone call to a local police station. Same thing goes by for the fact when pressure goes beyond certain level.

Stage 5: Coding

- Converts design into ***code*** in specific language
- ***Goal:*** Implement the design with simple and easy to understand code
- programs must be modular in nature. This helps in fast development, maintenance and future changes, if required.
- Coding phase affects both ***testing*** and ***maintenance***.
  - Well written code reduces testing and maintenance effort.
- ***Output:*** is source-code.

Stage 6: Testing

- ***Defects*** are introduced in each phase
  - Must be found and removed to achieve high quality
- Software testing is a process of analyzing software for the purpose of finding bugs.
- Using test data, following test runs are carried out
  - Unit test: performed by the respective developers on the individual units of source code to ensure that the individual parts are correct in terms of requirements and functionality.
  - System test: done after unit test. System testing tests the system as a whole. Actual output of the system is matched with the expected outputs. Errors are identified and fixed.
  - *User acceptance testing (UAT)* – determines if the system satisfies the business requirements

- ***Outputs:*** are
  - Test plans/results
  - Final tested (reliable) code

Stage 7: Built Release

- After UAT, deployment phase begins.
- Final phase of SDLC, puts the product into production
- All programs of system are loaded onto the user's computer.
- Then training of user starts including
  - how to execute the package
  - how to enter the data
  - how to process data

Built Release Strategies

- Parallel run: computerized & manual systems are executed in parallel.
- Advantages of Parallel run:
  - Manual results comparison with the computerized one.
  - Failure of the computerized system at the early stage, does not affect the working of the organization.
- Pilot run: New system is installed in parts. Some part of the new system is installed first and executed successfully for considerable time period.
- Advantages:
  - When results are found satisfactory then only other parts are implemented.
  - This strategy builds the confidence and the errors are traced easily.

Stage 8: Maintenance

- Maintenance phase focuses on changes that associated with
  - Error Correction
  - Platform Adaptations required
  - Enhancement due to change
  - Re-engineering
- Maintenance is required to:
  - eliminate errors in the system during its working life
  - tune the system to any variations in its working environment.
- System Review: is necessary from time to time for:
  - knowing the full capabilities of the system
  - knowing the required changes or the additional requirements
  - studying the performance
- Major change during the review:
  - If a major change to a system is needed, a new project may have to be set up to carry out the change.
  - New project will then proceed through all above life cycle phases.

System Environments

- Development
- Test
- Staging
- Pre-Production
- Production
- Mirror
- Roles involved:
- D, PM, TM, BA.

That is all

Software Engineering
CS-303

Software Process & Process Models

Lecture # 7,8
04, 06 Feb

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- DFD
- Context Level DFD

Data Flow Diagrams

- DFDs are graphic representation of the **flow of data** or information into and out of a system
  - ***what does the system do to the data?***

- A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system.
- The DFD also provides information about the outputs and inputs of each entity and the process itself.
- A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart

Why DFD?

- DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer.
- DFD has often been used due to the following reasons:
  - Logical information flow of the system
  - Determination of physical system construction requirements
  - Simplicity of notation

DFD Symbols

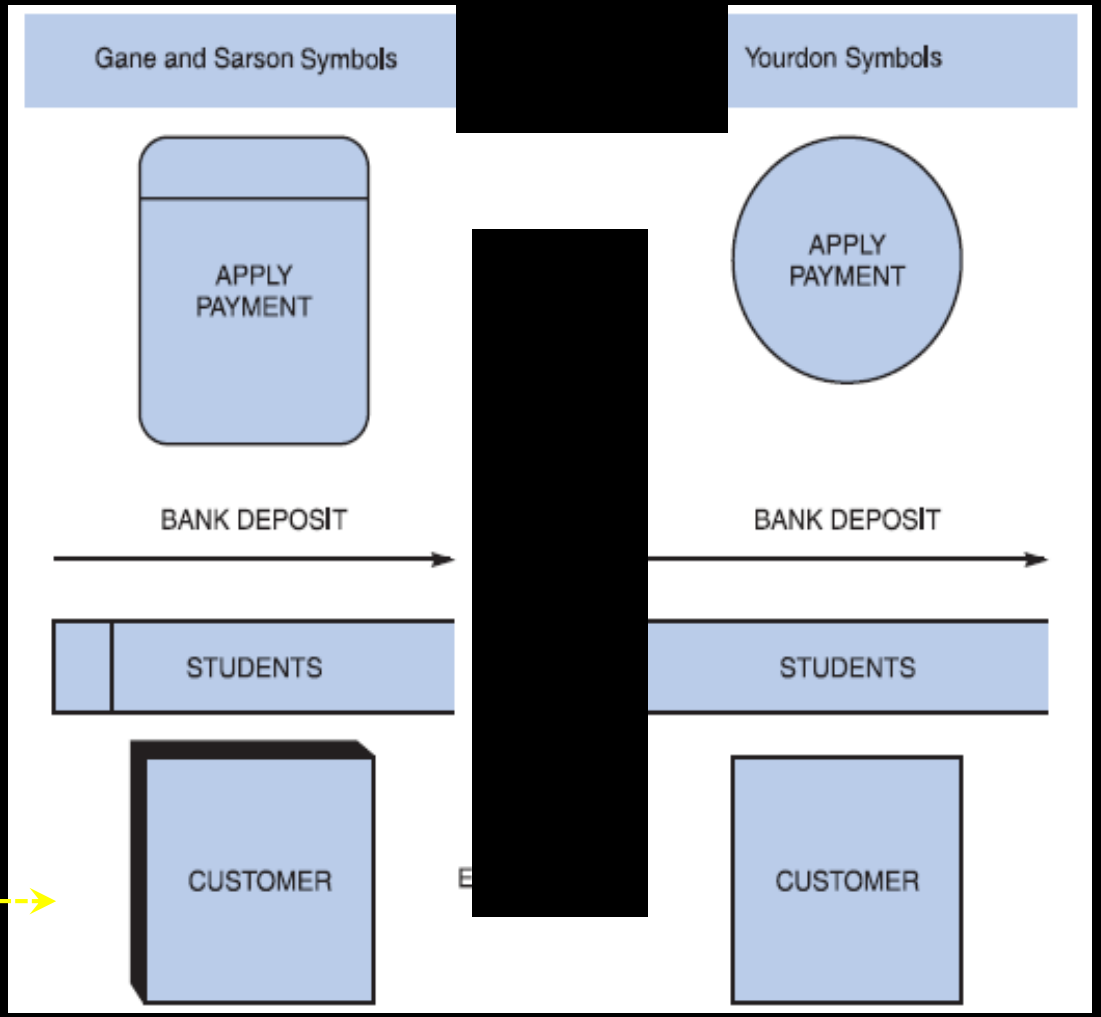
DFDs use ***four basic symbols*** that represent

Processes

Data flows

Data stores

External Entities

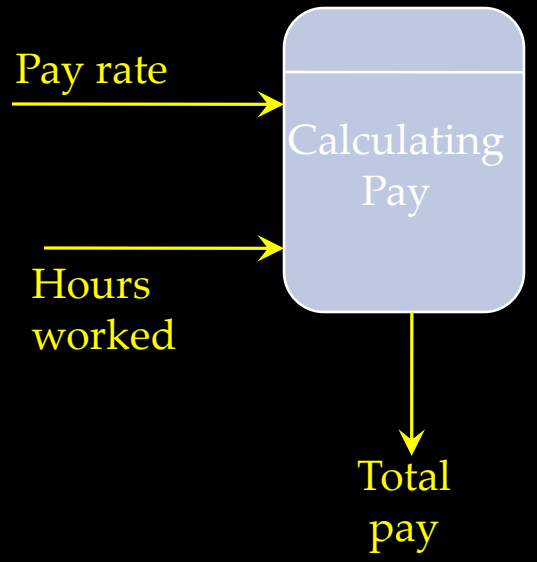


Data Flow Diagrams

Process Symbol

- A process receives ***input data*** and produces ***output*** .

- For instance, the process for ***calculating pay*** uses two inputs (pay rate and hours worked) to produce one output (total pay).
  - Processes can be very simple or quite complex.
  - Note: Processing details are not shown in a DFD.



Process Symbol

- A process symbol can be referred to as a **black box**, because the inputs, outputs, and general functions of the process are known, but the ***underlying details*** and ***logic*** of the process are ***hidden***.

- By showing processes as black boxes, an analyst can create DFDs that show how the ***system functions***, but ***avoid unnecessary detail***.

- When the analyst wishes to show additional levels of detail, he/she can zoom in on a process symbol and create a more in-depth DFD (leve-1 and level-2) that shows the process's internal workings.

Data Flow Diagrams

Data Flow Symbol 

- A data flow is a path for data to move from one part of the information system to another.

- A data flow in a DFD represents one or more ***data items***.

- In previous example we saw , the process for ***calculating pay*** uses two inputs (pay rate and hours worked) to produce one output (total pay).

Data Flow Symbol

- The symbol for a data flow **is a line** with a arrowhead.
- **Data flow name** appears above, below, or alongside the line.
- A **data flow name** consists of a singular noun and an adjective, if needed.

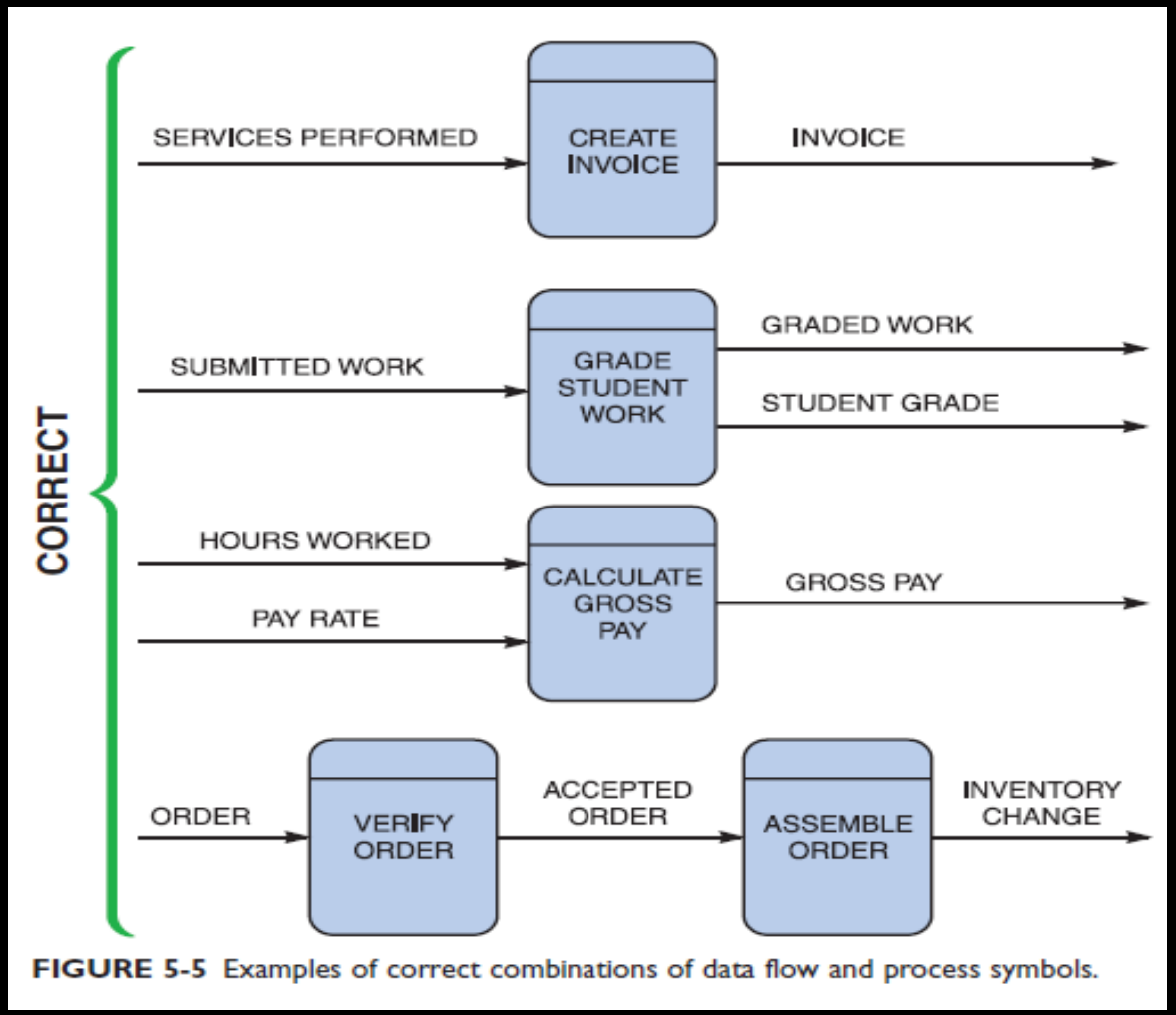


FIGURE 5-5 Examples of correct combinations of data flow and process symbols.

Data Flow Diagrams

Data Store Symbol

- A ***data store*** is used in a DFD to represent data that the system stores because one or more processes need to use the data at a later time.

- For instance, ***teachers*** need to store student scores on tests and assignments during the semester so they can assign final grades at the end of the term.

- Similarly, a ***company*** stores employee salary and deduction data during the year in order to print W-2 forms with total earnings and deductions at the end of the year.

- A DFD ***does not show*** the detailed contents of a data store.

Data Store Symbol

Examples of **data store** names are

DAILY PAYMENTS

ACCOUNTS RECEIVABLE

PATIENTS

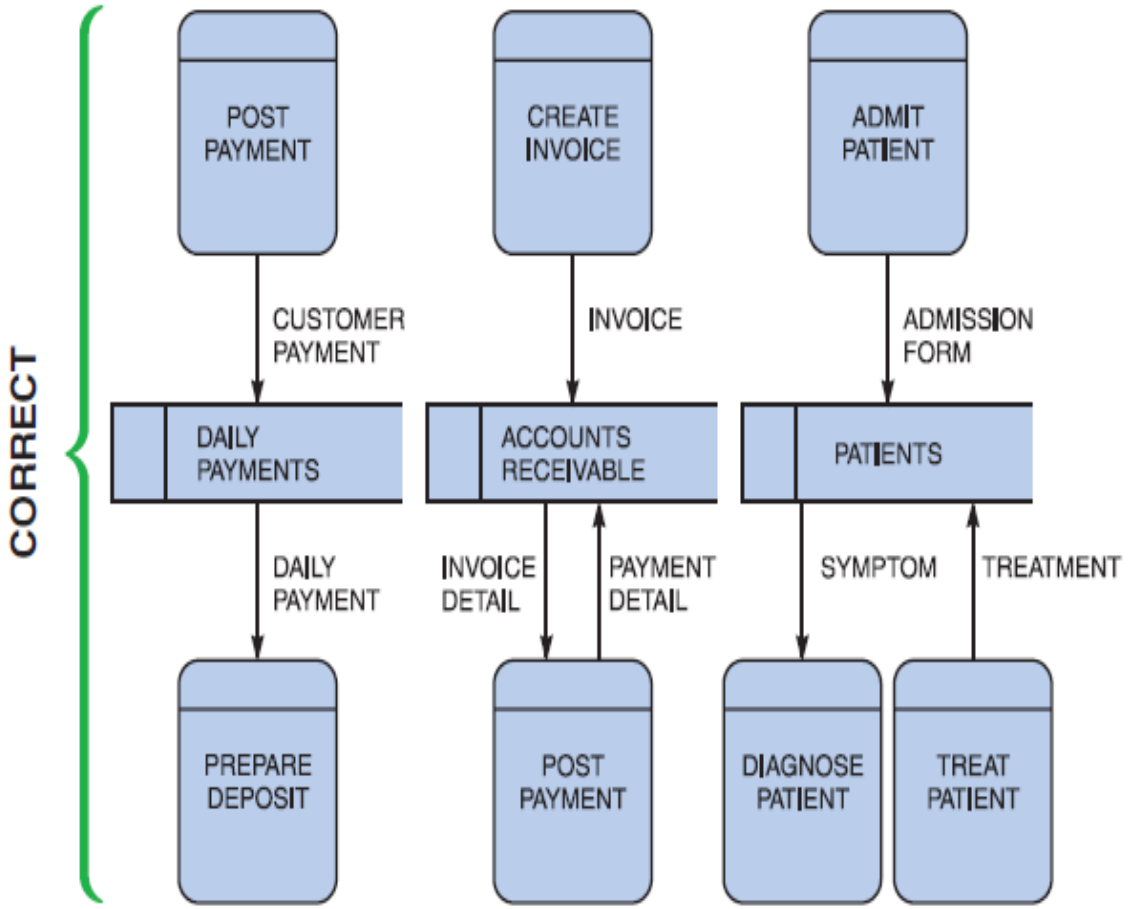


FIGURE 5-7 Examples of correct uses of data store symbols in a data flow diagram.

Data Flow Diagrams

External Entity Symbol

- The symbol for an ***entity*** is a ***rectangle.*** The name of the entity appears inside the symbol.
- DFD shows only ***external entities*** that ***provide data*** to the system or ***receive output*** from the system.
- DFD shows the ***boundaries*** of the system and how the system interfaces with the outside world.
- DFD entities also are called ***terminators***, because they are ***data origins*** or ***final destinations***.

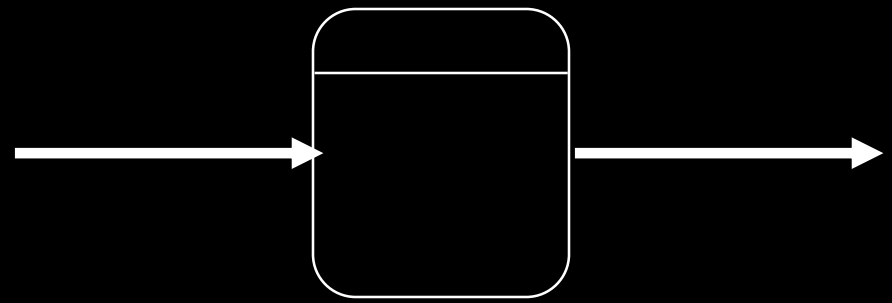
External Entity Symbol

- Systems analysts call an entity that supplies data to the system ***a source***, and an entity that receives data from the system **a sink**.

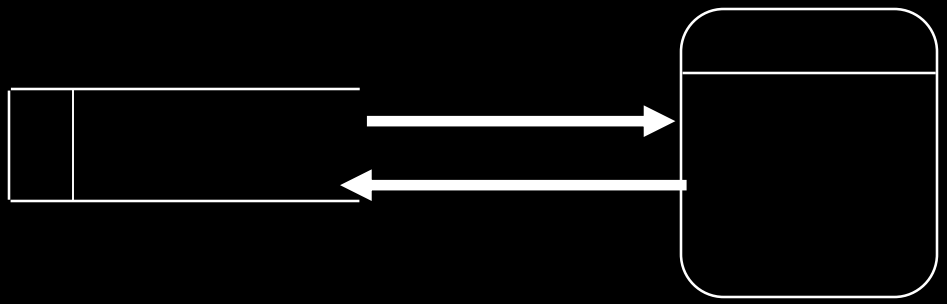
- An entity name is the singular form of a
  - department, outside organization,
  - other information system, or person.

- An external entity can be a source or a sink or both, but each entity must be connected to a process ***by a data flow***.

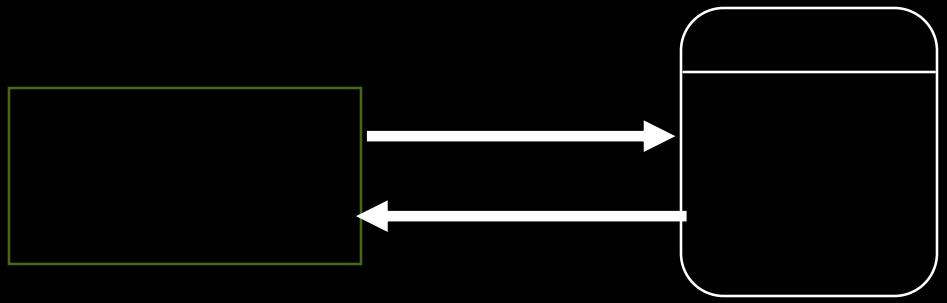
Rules for Drawing DFD's



A **minimum** of one data flow in and one data flow out of a process



A datastore **must** be connected to a process (either in, out, or both)

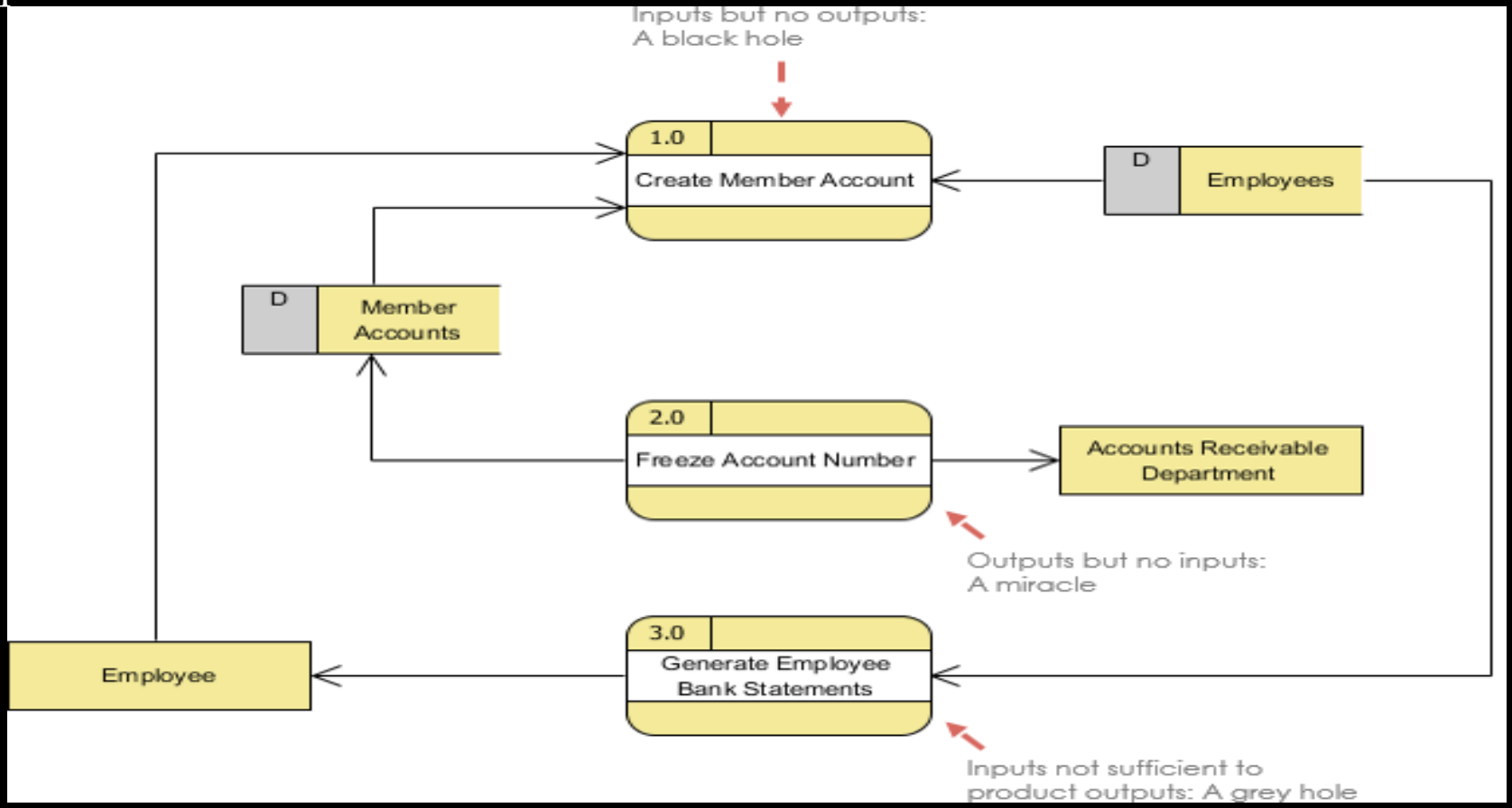


An external entity **must** be connected to a process (either in, out, or both)

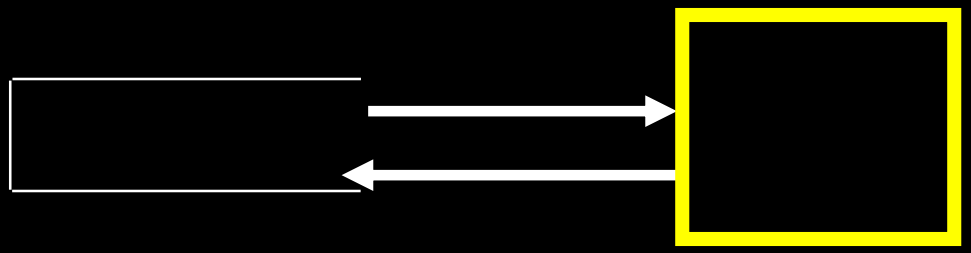
A single data flow **must** only flow one way

DFD: Common Mistakes

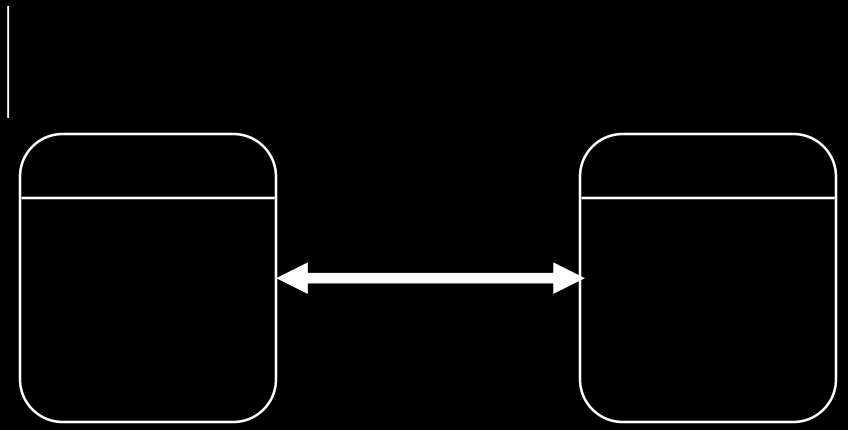
Black holes - A processing step may have input flows but no output flows.
Miracles - A processing step may have output flows but no input flows.
Grey holes - A processing step may have outputs that are greater than the sum of its inputs.



DFD: Common Mistakes



- Data store is hooked to external entity. This means external entity can read and write to your data file without auditing!!



- The data flow goes in two directions at once. Two or more arrows should be used to show the flow to and from each process.

DFD Levels: Context Diagram

- First we can start to draw a general overview. This general overview is called a "Context Diagram".
- A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. At this level, there is only one visible process node that represents the functions of a complete system in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:
  - Shows the overview of the boundaries of a system
  - No technical knowledge is required to understand with the simple notation
  - Simple to draw,
- A Context Diagram shows three things:
  - all external entities
  - a single process labeled "0"that represents the entire system
  - the major information flows between the external entities and the system.

Data Flow Diagram Tips and Cautions

1. Process labels should be verb phrases; data stores are represented by nouns
2. A data store must be associated with at least a process
3. An external entity must be associated with at least a process
4. Don't let it get too complex; normally 5 - 7 average people can manage processes
5. Datastores should not be connected to an external entity, otherwise, it would mean that you're giving an external entity direct access to your data files
6. Data flows should not exist between 2 external entities without going through a process

Building a DFD

- It would be impossible to understand all of the data flows, and to identify all of the 'external entities' relating to our information system in one pass, so we tend to draw DFD's incrementally.
- We tend to start at the context level, break processes down to Level 0, and then start to consider where data enters and exits our information system, where it is stored, and how a process converts it from one form to another. We are interested here in the movement of *data* and its conversion.

DFD Example: Bus Garage Repairs

- Buses come to a garage for repairs.
- A mechanic and helper perform the repair, record the reason for the repair and record the total cost of all parts used on a Shop Repair Order.
- Information on labor, parts and repair outcome is used for billing by the Accounting Department, parts monitoring by the inventory management computer system and a performance review by the supervisor.

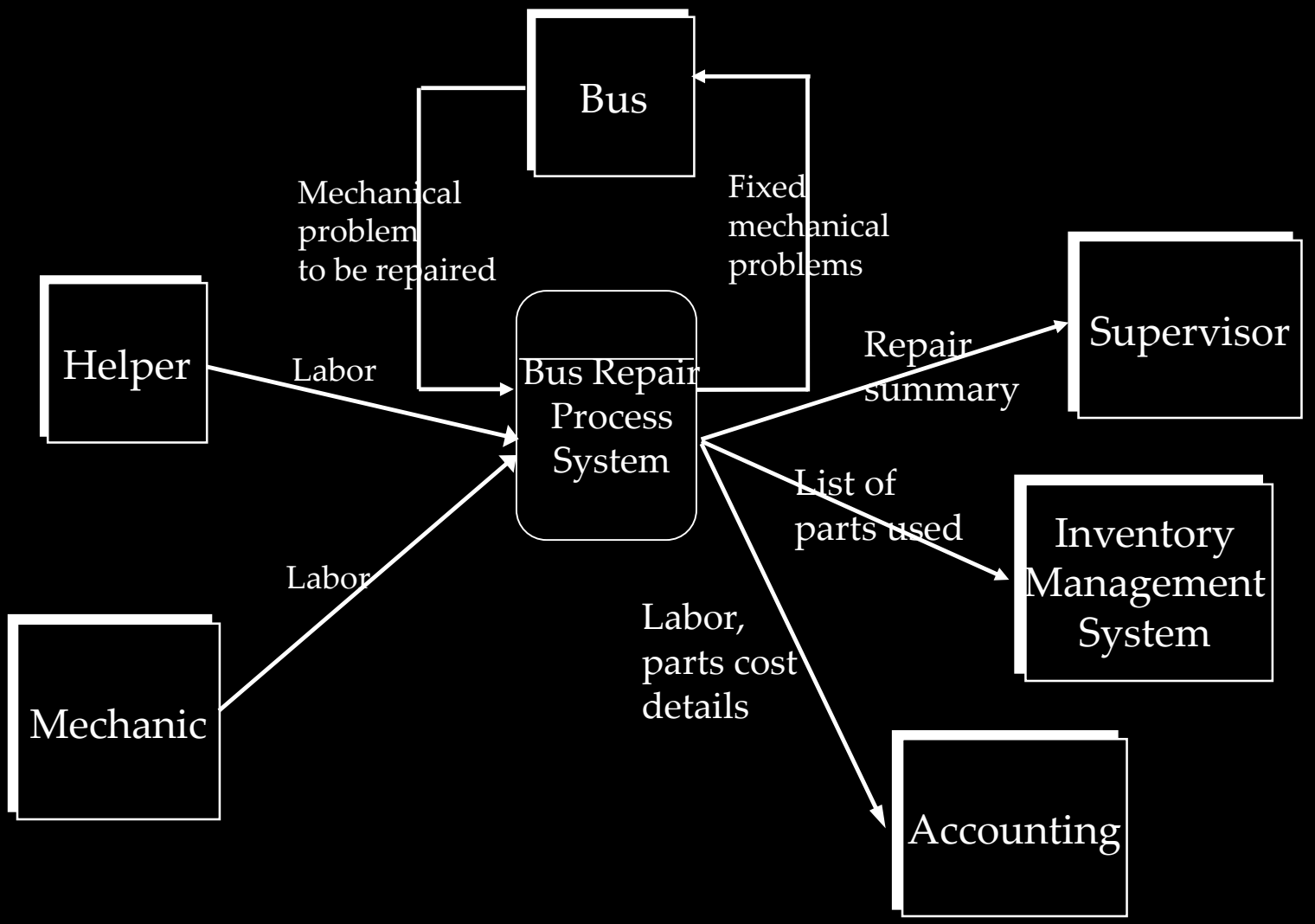
DFD Example: Bus Garage Repairs (cont'd)

- *External Entities*: Bus, Mechanic, Helper, Supervisor, Inventory Management System, Accounting Department, etc.
- *Key process* ("the system"): performing repairs and storing information related to repairs
- *Processes*:
  - Record Bus ID and reason for repair
  - Determine parts needed
  - Perform repair
  - Calculate parts extended and total cost
  - Record labor hours, cost

DFD Example: Bus Garage Repairs (cont'd)

- ***Data stores:***
  - **Personnel file**
  - **Repairs file**
  - **Parts list**
- ***Data flows:***
  - **Repair order**
  - **Bus record**
  - **Parts record**
  - **Employee timecard**
  - **Invoices**

Bus Garage Context Diagram



Building a DFD

- A DFD is NOT a flowchart. It simply shows how and where data itself progresses through our system.

Today's Outline

- Software process models
  - Water Fall Model
  - Evolutionary Development
  - Component base Software Engineering / Reuse-oriented development
- Process iteration
  - Incremental Model
  - Spiral Model
- Process activities

The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Plan-driven and Agile Processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software Process Model

- ***Software Process*** : is coherent sets of activities for specifying, designing, implementing and testing software systems.
- ***A software process model*** is an abstract representation of a software process.
- *It is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.* It presents a description of a process from some particular perspective.

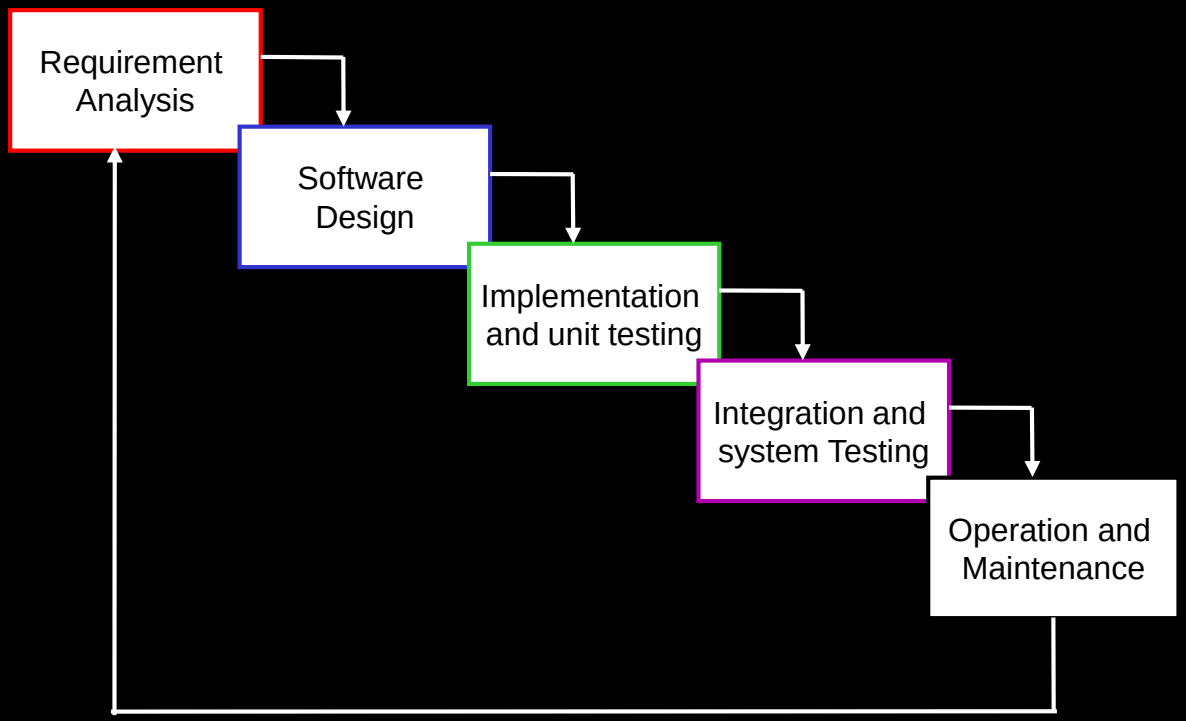
Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development. It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.
- Incremental development
  - Specification, development and validation are interleaved.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

Waterfall

Each box represents a **set of tasks** that results in the production of each or more work products.

Each new phase begins when the **work products** of the previous phase as completed, frozen and signed off.



Waterfall (when to use)

- Well suited for projects where requirements can be understood easily and technology decisions are easy
- Has been used widely
- For standard/familiar type of projects it still may be the most optimum.

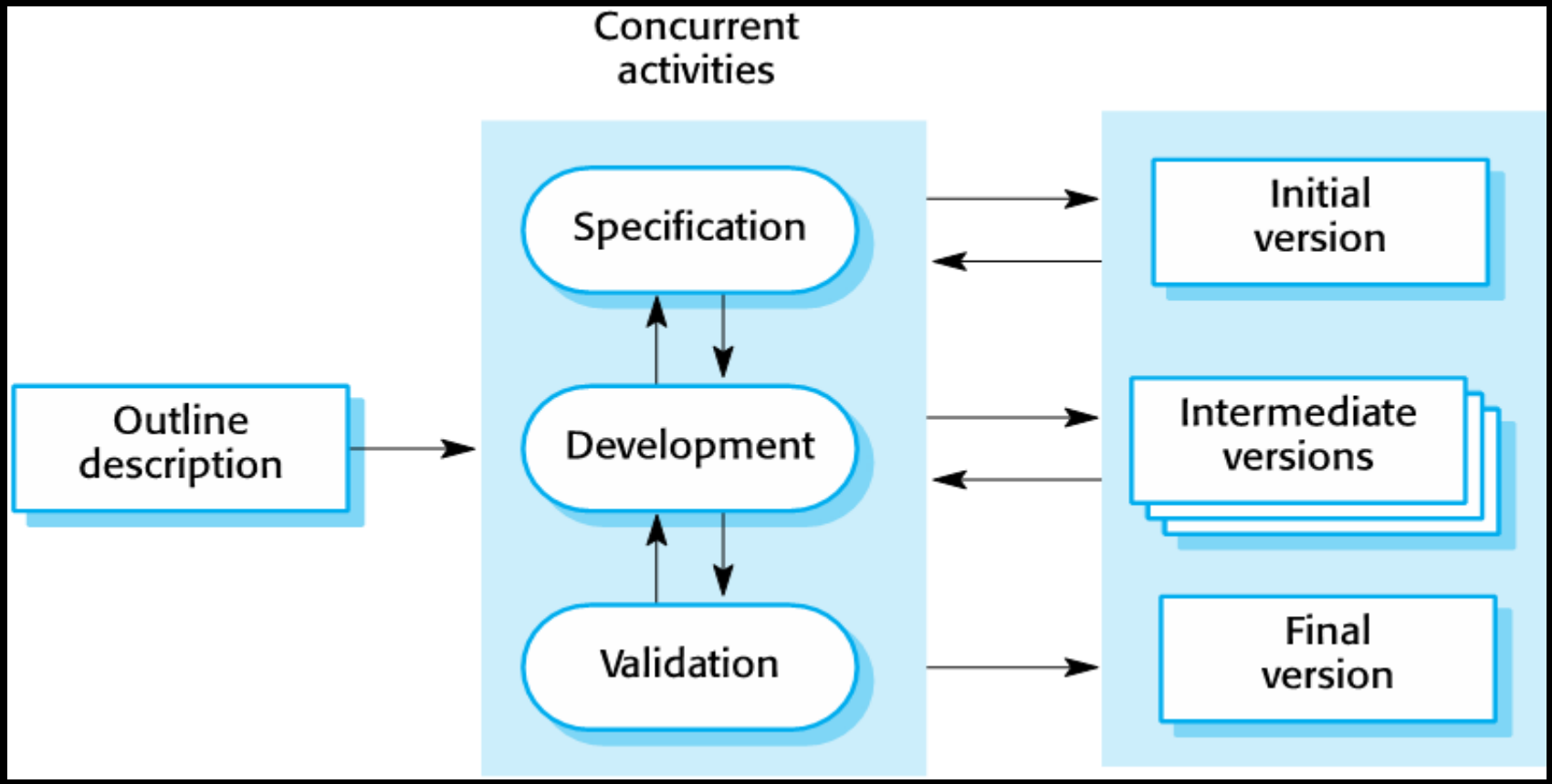
Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Disadvantages of Waterfall Model

- Unable to work where high level of uncertainty is involved
- Requirements need to be stable hence making model rigid
- Unrealistic to state all requirements at the beginning
- Does not handle concurrent events – development teams are delayed waiting for others
- Difficult and expensive to change decisions
- The user is involved only in the beginning phase of requirement gathering and than during acceptance phase

Iterative Development



Iterative Development Benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Iterative development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

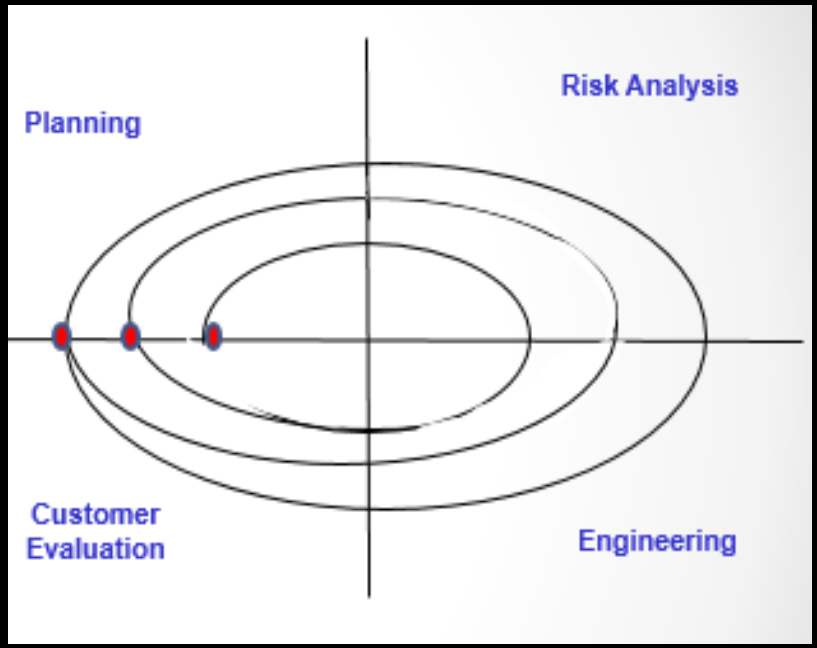
Evolutionary Models: Spiral

- Spiral is primary **Risk Driven** approach. Spiral model consist of different **cycle. In each cycle we try to address some risk elements.**

- Planning: Determines objectives, alternatives and constraints.
- Risk Analysis: Analysis of alternatives as well as an identification and/or resolution of risks.
- Engineering: Development of the next level of product
- Customer evaluation: Assessment of the results of engineering

Spiral

- With each iteration around the spiral progressively more complete versions of the software are built.
- Spiral model enables the developer, and the customer, to understand and ***react to risk*** at each evolutionary level.
- Each loop around the spiral implies that project costs and schedules may be modified.



- This creates problems in fixed-price project.

That is all