OBJECT ORIENTED ANALYSIS AND DESIGN WITH SOFTWARE ENGINEERING

INTRODUCTION TO SOFTWARE ENGINEERING

Vinay Joshi
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

Look at the Pictures… What do you see?

What do they have in common? Or, what you don't see?

Code

(and lots of It)

Looking at it in more detail

A Boeing 747 has more lines of code than the parts of the aeroplane including nuts and bolts

~ 5 Million lines of Code

Most Games have

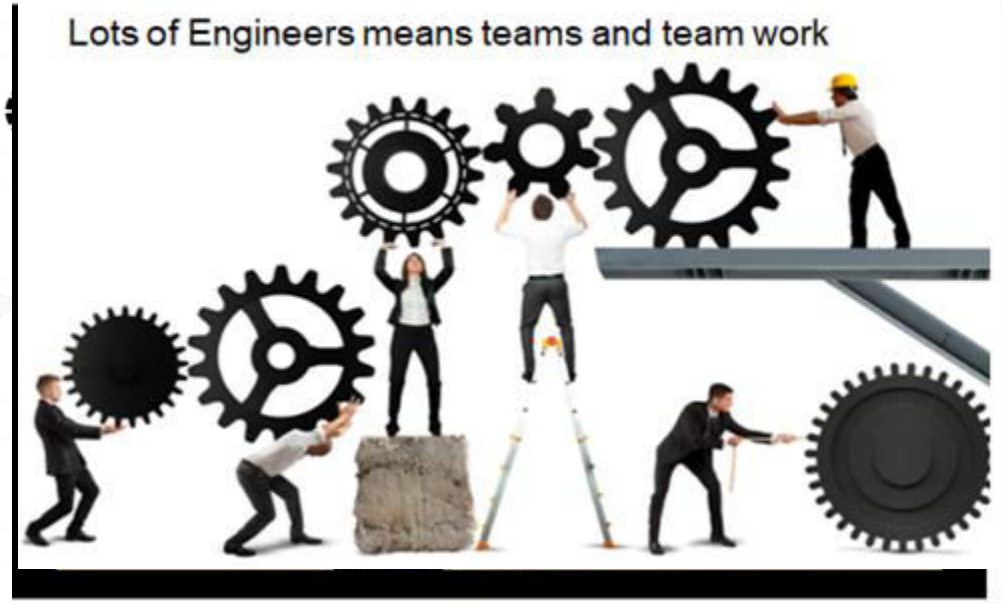
~ 6 Million or more lines of code

(and this has to be done on different systems and different OSs etc.)

INTRODUCTION TO SOFTWARE ENGINEERING

Leading to

- Need for interacting with customers and other stake holders on what is their requirements/needs
- Need for understanding how is this going to be used, and by whom
- Need Experts in different domains to support this (Cross-domain)
- Need for good Planning
- Need for good Teamwork
- Need to have an ability to scale/support



E

SE

ng period of time

INTRODUCTION TO SOFTWARE ENGINEERING

Some Definitions…

What is Software?

What is a Software Product?

What is Software Engineering?

Given all of this context

If we look at some of the definitions involved

Software can be collection of executable computer programs (code), their configuration files and associated libraries and their documentations serving a computational purpose.

Software Product is Software when made for a specific or specific group of requirements

- Software can include prefixes as **Generic, Custom, System** and **Application**

Engineering is all about acquiring and using well defined scientific principles and systematic methods for developing products, with economic sense, social perspective and practical considerations.

- Involves making **decisions** or making choices given a context

INTRODUCTION TO SOFTWARE ENGINEERING

Software Engineering (SE)

- Software Engineering is the **systematic, disciplined, quantifiable** approach towards the development, operation, and maintenance of software products.
- A software product developed using Software Engineering principles has a **higher probability** of being efficient and reliable.
- Software Engineering principle drives **usage of appropriate tools and techniques** depending on the problem to be solved, while considering the constraints and resources available
- Focuses more on **techniques for developing and maintaining** software that is correct from its inception

INTRODUCTION TO SOFTWARE ENGINEERING

Is Computer Science == Software Engineering ?

Both of them solve problems using code but how they go about are different

E.g. A bridge collapse

Scientist build something to learn something new

Whereas

Engineer learns things to design and build quality products

Is it Scientist's problem or an Engineering problem

Scientists want to achieve scientific breakthroughs

Whereas

Engineers want to avoid engineering failures

INTRODUCTION TO SOFTWARE ENGINEERING

Is Computer Science == Software Engineering ?

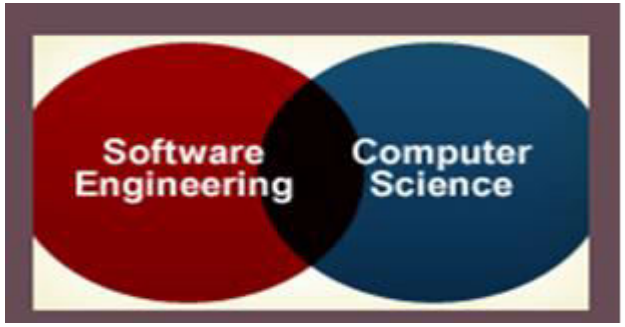
Content differences:

Computer science

- Focuses on **foundations of computing** including, algorithms, programming languages, theories of computing, artificial intelligence,.., and some hardware design.

Software engineering

- Focuses on technical and managerial leadership for large and complex systems
- It's the foundation of **enduring engineering principles** which will support a lifetime of practice, amid emerging technologies.



COMPUTER SCIENCE AT A GLANCE:

- Algorithms
- Theories of Computation
- Compilers
- Operating Systems
- Artifical Intelligence

SOFTWARE ENGINEERING AT A GLANCE:

- Software Architecture
- Project Management
- Technical Planning
- Risk Management
- Software Assurance

INTRODUCTION TO SOFTWARE ENGINEERING

Fundamental Drivers of Software Engineering (the Why's)

1. Industrial Strength Software
2. Software is Expensive
3. Late and Unreliable
4. Can influence the life and death of a person
5. Heterogeneity
6. Diversity
7. Business and Social change
8. Security and Trust
9. Scale
10. Quality and Productivity
11. Consistency and Repeatability

Fundamental Drivers

1. Industrial Strength Software

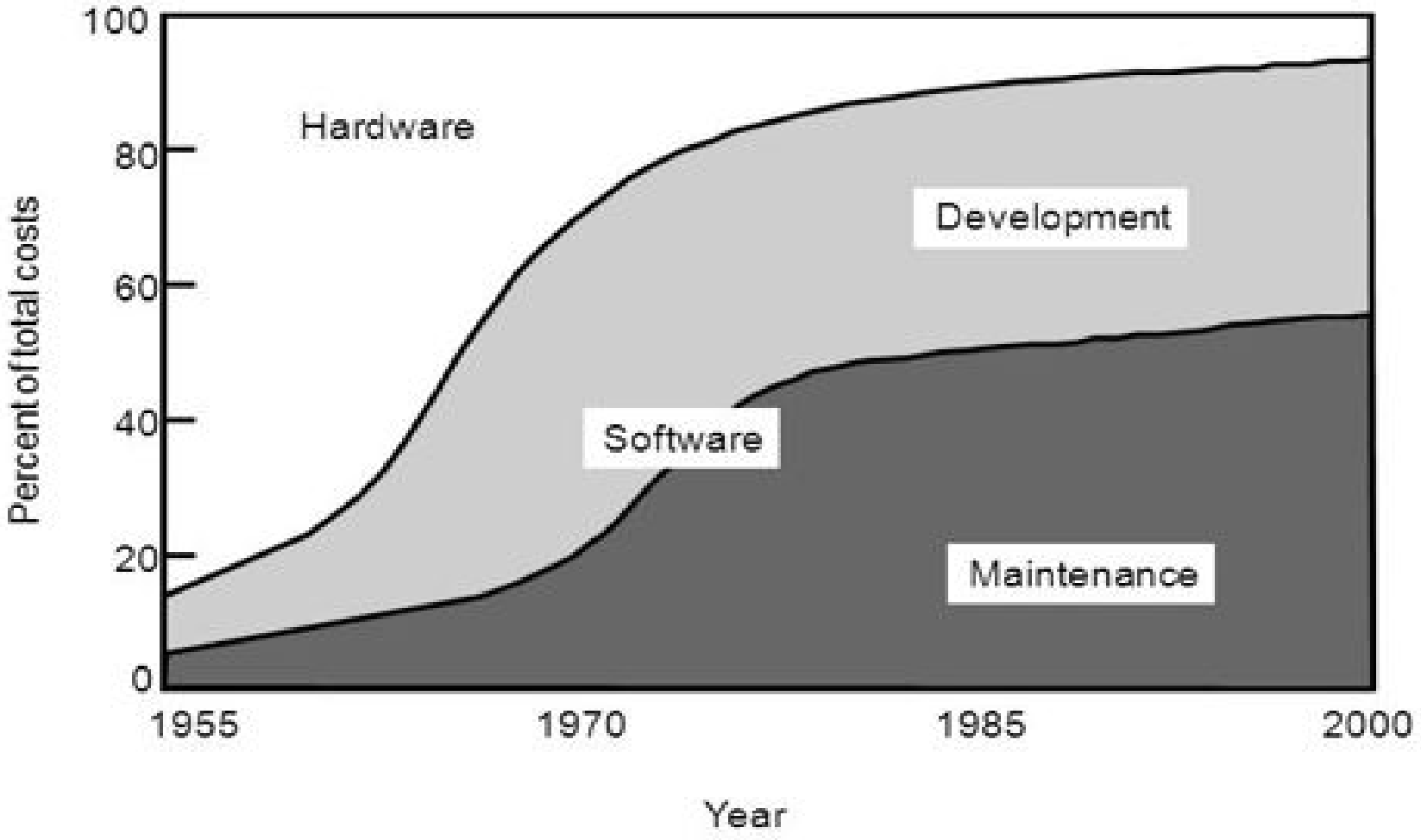
- needs to be
  - **Operational :** Functionality, Usability, Correctness …
  - **Capable of being moved:** Portability, Interoperability ..
  - **Maintainable:** Modularity, Maintainability, Scalability ..
- have elaborate Documentation
- absence/minimal number, of bugs is of primary importance
- are Impactful to the business's leading to the need for strong processes for quality and resiliency

2. Software is Expensive

- Software is labour intensive, Industrial-strength software is very expensive
- Each line of code can cost from $5 to $35 when calculated across the lifecycle could go as high as $400
- Software also has loads of Maintenance and Rework which costs money
  - Maintenance could be corrective, adaptive or rework
  - Includes cost of Hardware-Software-Maintenance

INTRODUCTION TO SOFTWARE ENGINEERING

Relative distribution of software/hardware costs

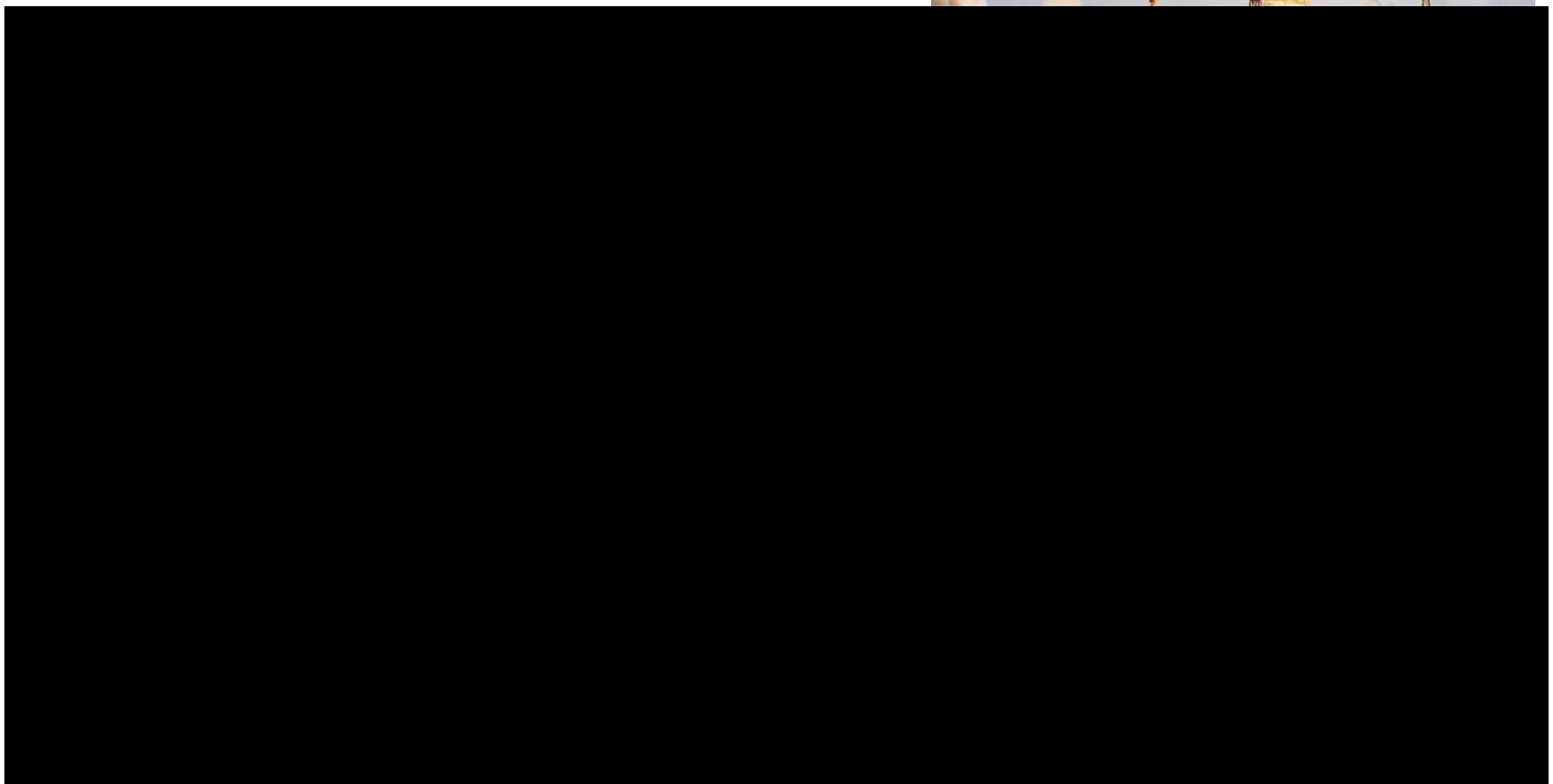


INTRODUCTION TO SOFTWARE ENGINEERING

Fundamental Drivers

3. Late and Unreliable

- Typically 35% of the computer based projects are runaway



INTRODUCTION TO SOFTWARE ENGINEERING

ARIANE Flight 501

- Disintegration after 39 sec
- Caused by large correction for attitude deviation
- Caused by wrong data being sent to On Board Computer
- Caused by software exception in Inertial Reference System after 36 sec

The details

- Overflow in conversion of variable BH from 64-bit floating point to 16-bit signed integer
- Of 7 risky conversions, 4 were protected; BH was not
- Reasoning: physically limited, or large margin of safety
- In case of exception: report failure on data-bus and shut down

INTRODUCTION TO SOFTWARE ENGINEERING

Possible explanations

- **Inadequate testing**
  - Specification did not contain Ariane 5 trajectory data as a functional requirement
  - In tests, the SRI's (components that measure altitude and movements of the launcher) were simulated by software modules
- **Improper reuse**
  - If a component works perfectly well in one environment, it doesn't necessarily do so in another
  - Ariane 5 was much faster than Ariane 4, and horizontal velocity builds up more rapidly
    ⇒ excessive values for parameter in question
  - Wish for quick alignment after hold in shutdown
    ⇒ this software runs for a while after lift-off. It doesn't have any purpose for the Ariane 5, but was still kept
- **Wrong design philosophy**
  - If something breaks down, such is caused by a random hardware failure
  - Action: shutdown that part
  - Then: provision for design errors

Further Information if interested

- IEEE Computer, Jan. 1997, p. 129-130
- http://www.cs.vu.nl/~hans/ariane5report.html

INTRODUCTION TO SOFTWARE ENGINEERING

Fundamental Drivers

4. Can influence the life and death of a person

Therac 25 – (Radiation Therapy M/C) 6 People died due to over exposure to radiation – One cause a S/W bug

5. Heterogeneity

Systems are required to operate as distributed systems across networks that include different types of computer & mobile devices.

6. Diversity

There are many different types of software systems necessitating different techniques, methods and tools.

7. Business and Social change

Changes are happening quickly with economies & organizations getting more global, and availability of new technologies. Organizations need to have the ability to change their existing software to rapidly develop new S/W

INTRODUCTION TO SOFTWARE ENGINEERING

Fundamental Drivers

8. Security and Trust

SW is intertwined with all aspects of our lives, it is essential that we can trust that software

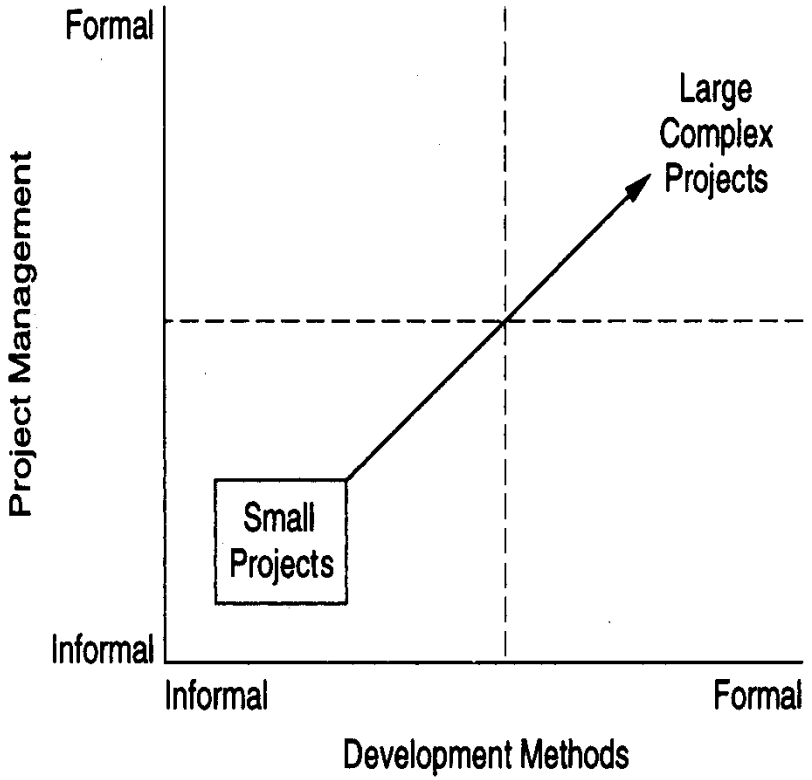
9. Scale

Processes and methodologies needs to scale easily with size

10. Quality and Productivity

Software quality which could be looked at as FLURPS+(efficiency or Maintainability) + Portability

11. Consistency and Repeatability



INTRODUCTION TO SOFTWARE ENGINEERING

Summarily Software Engineering is for

- Development of BIG programs
- Mastering complexity of these Large programs/Systems
- Development of Software that is Evolving
- Efficient development
- Supporting large teams, teamwork and global development
- Ensuring a process through which software being developed, supports the users effectively
- Ensuring that the right choices or decisions are made while working under constraints or supports the balancing act
- Ensuring the needs of visibility and continuity

INTRODUCTION TO SOFTWARE ENGINEERING

Comparison with Cooking…

INTRODUCTION TO SOFTWARE ENGINEERING

Course Content

1. Introduction, Requirements Engineering and Project Management
2. Architecture and Design (with Object Oriented Analysis and Design)
3. Development and Implementation
4. Software Testing, Quality and Ethics
5. IT Services Management and Dev Ops

INTRODUCTION TO SOFTWARE ENGINEERING

Evaluation Policy

1. ISA 1 (60 marks reduced to 25 marks)
2. ISA 2 (40 marks reduced to 15 marks)
3. Quiz (10 marks)

4. ESA (100 marks reduced to 50 marks)

All assessments (except Quiz) will be conducted in Pen and Paper mode.

THANK YOU

Vinay Joshi
Department of Computer Science and Engineering
vinayj@pes.edu

SOFTWARE ENGINEERING

INTRODUCTION TO SOFTWARE ENGINEERING

Phalachandra H. L.
Department of Computer Science

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

Evolving into a software development lifecycle & process

Phalachandra H. L
Department of Computer Science

House Construction Lifecycle

Could you think of the steps involved in the construction of a House

House Construction Lifecycle

1. You realize that you have the means or you have a want or you want to save money on rentals and think you want to build your own house
2. Collect information on the where you want to build it and the cost of site, cost of building, commute, ambience, green, investment. - distance to family/friends, etc.
3. Analyse whether it makes financial sense – rental/loan repayment/interest which can be earned ..

Decision Made

4. Hire a broker .. look at sites and select a site and buy a Site
5. Hire an Architect to evolve an conversion/rendering of your dream to an implementable Plan.
6. Meetings between you and the Architect to describe what you want
    1. Includes what you want it to look like
    2. Features to be included
7. Architect draws up the rendering/models/drawings - elevation/plans
8. Interactions between you and architect and the requirements are finalized post relooking at renderings/models/drawings and get regulatory approvals
9. Time plan with what are different specializations needed and dependencies are identified ..
   Scope – Schedule - Resources

House Construction Lifecycle

10. Architect – and you together identify and hire a contractor to build a house .. additional specializations like leads towards a structural engineer, plumbing, electrical, carpentry etc.
11. Architect provides different views and drawings/requirements for each of the person Electrical drawing, Plumbing drawing, Structural diagram etc.
12. Further discussions, changes and everyone agrees to the plans and parallelly contractor begins constructing the house
13. In between you would visit the site and indicate changes you would like .. Several changes and then house is constructed
14. There are different phases .. Columns, rest of Foundation, Walls, Roof, Plastering .. dependencies and implementation
15. Contractor believes its complete .. Has each of the things checked independently as a unit by the respective actors … electrician has checked wiring - points, plumber has checked for leakages … etc .. painter has done his parts etc.
16. Contractor (and maybe you too) look to see after all of them have completed (Integration and as complete function as a system) are there issues which are cropping up .. say wires showing up .. missing cable or network points or door being obstructed .. something not being able to be closed etc.
17. Before you move in, you go out to check several things (Switch too close to wardrobe and not easy to operate, Switch not next to the door etc..) (User acceptance testing) and then accept and do .. Grihapravesha ..
18. Initial hiccups when you would call the contractor A .. water not getting drained away .. and then you move in.
19. Now you are responsible for the house and you maintain the house later by calling the respective plumber etc.

House Construction Lifecycle

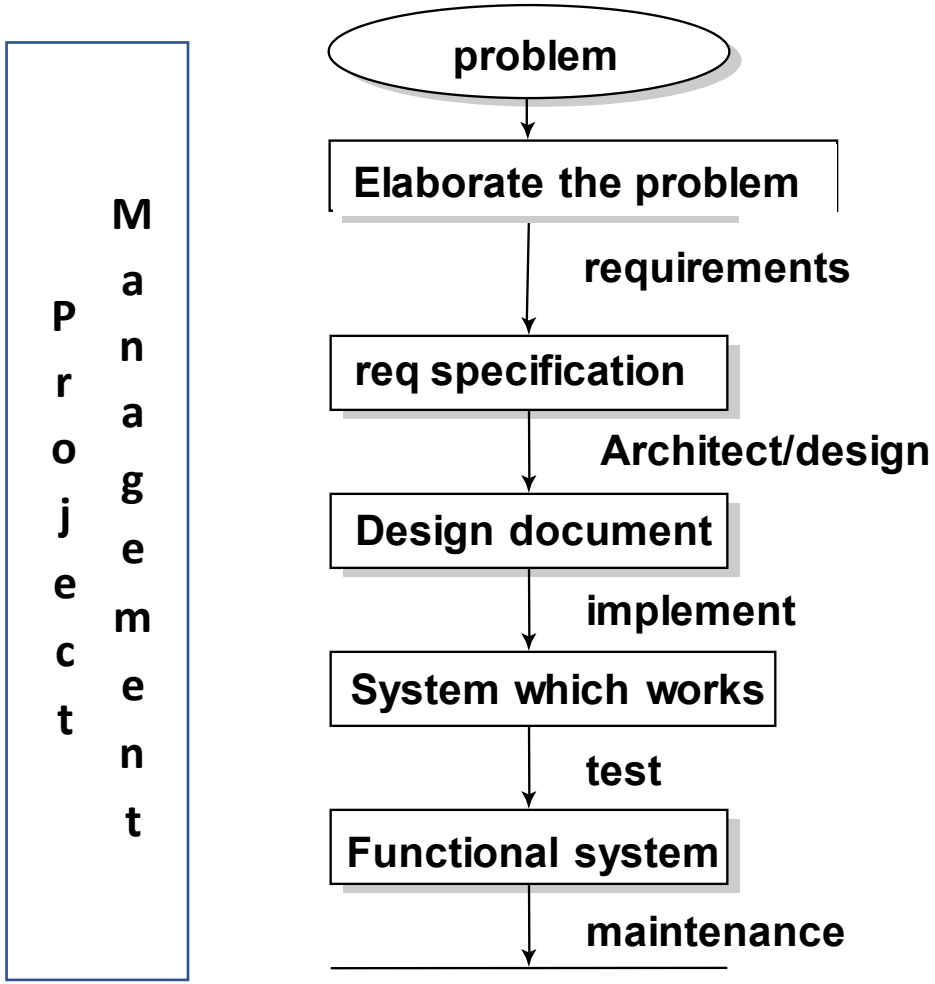
Few things to keep in context

1. You could get into the steps of constructing based on understanding the feasibility and decisioning to go ahead.
2. There could have been multiple rounds of discussions, validating your requirements, rendering/converting of your dream into a set of requirement which could be acted on.
3. Since there are number of stakeholders contributing to ensure that they all are working towards the same goal .. Lots of diagrams with different views details, designs, plans are done and shared [Documentation]
4. Give that everything is not easily visualizable, there may be changes, refinements which might need to be done during the construction of the house. Say like
   - Sky light or a anteroom etc
   - Cost escalations which may need to reduction of scope
   - Non-availability of some material leading to change in the plan etc

House Construction Lifecycle

5. You would definitely not want to finalize the plan and come back in the end to see the house .. You want to come in periodically see the progress which gives better understanding .. Which may need to make changes which needs to be done as and when and not waited till the end.
6. You would like to see the house after different phases of execution of the construction
    1. Columns are constructed
    2. Foundation is done
    3. Lintel and after the walls and before the roof is done
    4. Once after the supports are removed and the electrical and plumbing points are identified
    5. Once carpentry is done before final painting …etc
7. Validation at different levels, Acceptance and Maintenance

A simplistic view of a Software Development Lifecycle



The software Lifecycle

- Software Process (is also called Lifecycle or process model or lifecycle model)
  - Involves structured set (procedure/recipe) of activities (steps or phases mostly in a particular order) producing intermediate and final products
  - Every lifecycle step has a guiding principle that explain the goal of each phase. E.g. Requirements Engineering defines what the system should do.
    - Each of the steps in a phase can be a process by itself
- Products are outcomes of executing a process (or a set of processes) on a project

The Software Lifecycle

Each of these activities/steps/phases produces an intended output of some kind, required to develop a software system and either implicitly/explicitly has :

Entry criteria	: What conditions must be satisfied for initiating this phase
Task and its deliverable	: What should to be done in this phase
Exit criteria	: When can this phase be considered done successful
Who	: Who is responsible
Dependencies	: What are the dependencies for this phase ..etc.
Constraints	: e.g. Schedule

The Software Process

- Structure allows us to examine, understand, control and improve the activities in the process
- There are other relevant processes like management process, configuration management process, change management process, Inspection process etc. which are part of the Software Development too

THANK YOU

Phalachandra H. L.
Department of Computer Science
phalachandra@pes.edu

SOFTWARE ENGINEERING

INTRODUCTION TO SOFTWARE ENGINEERING

Phalachandra H.L.
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

SDLC, PDLC, PMLC & SMLC

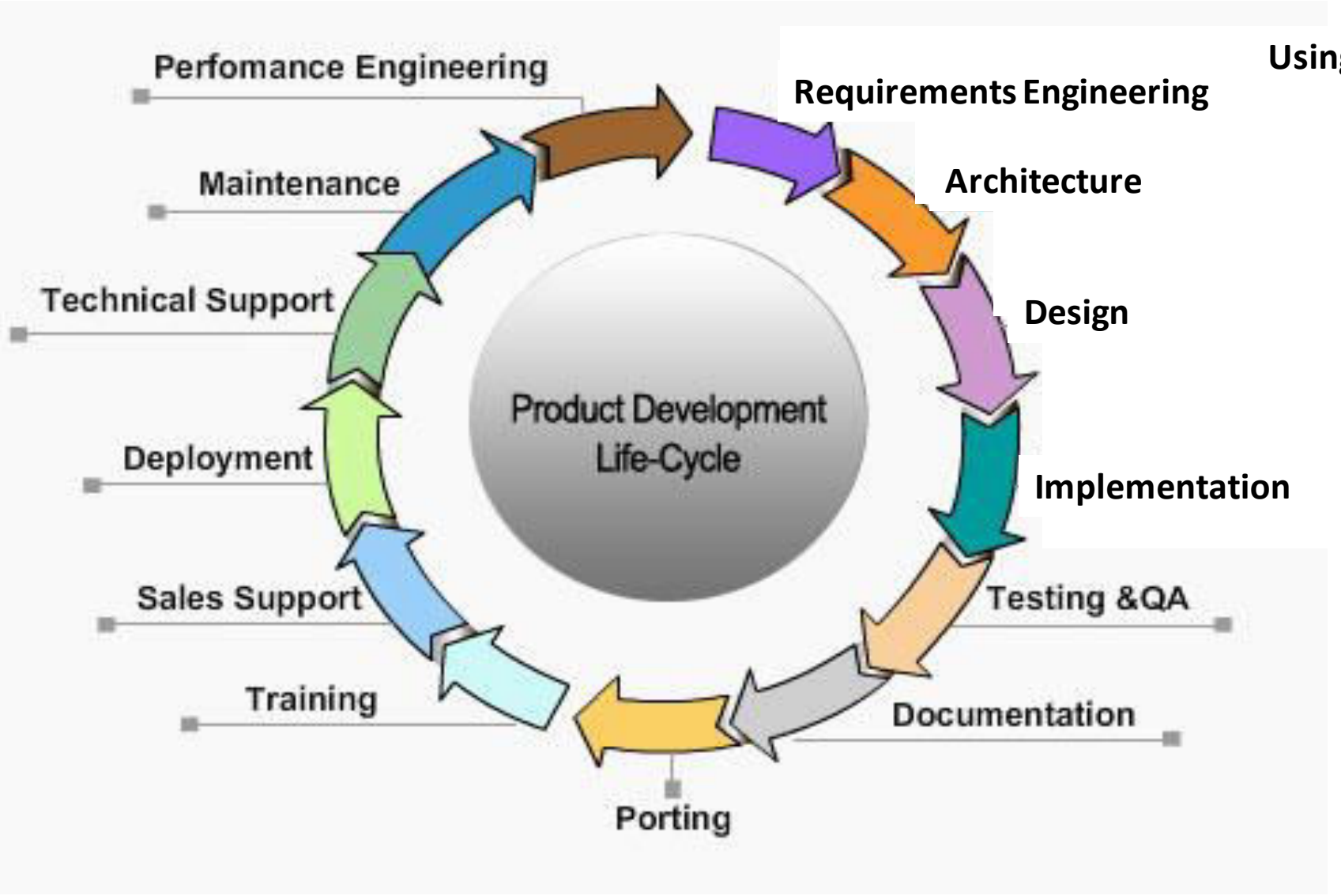
Phalachandra H. L
Department of Computer Science

INTRODUCTION TO SOFTWARE ENGINEERING

SDLC (Software Development Lifecyle)

INTRODUCTION TO SOFTWARE ENGINEERING

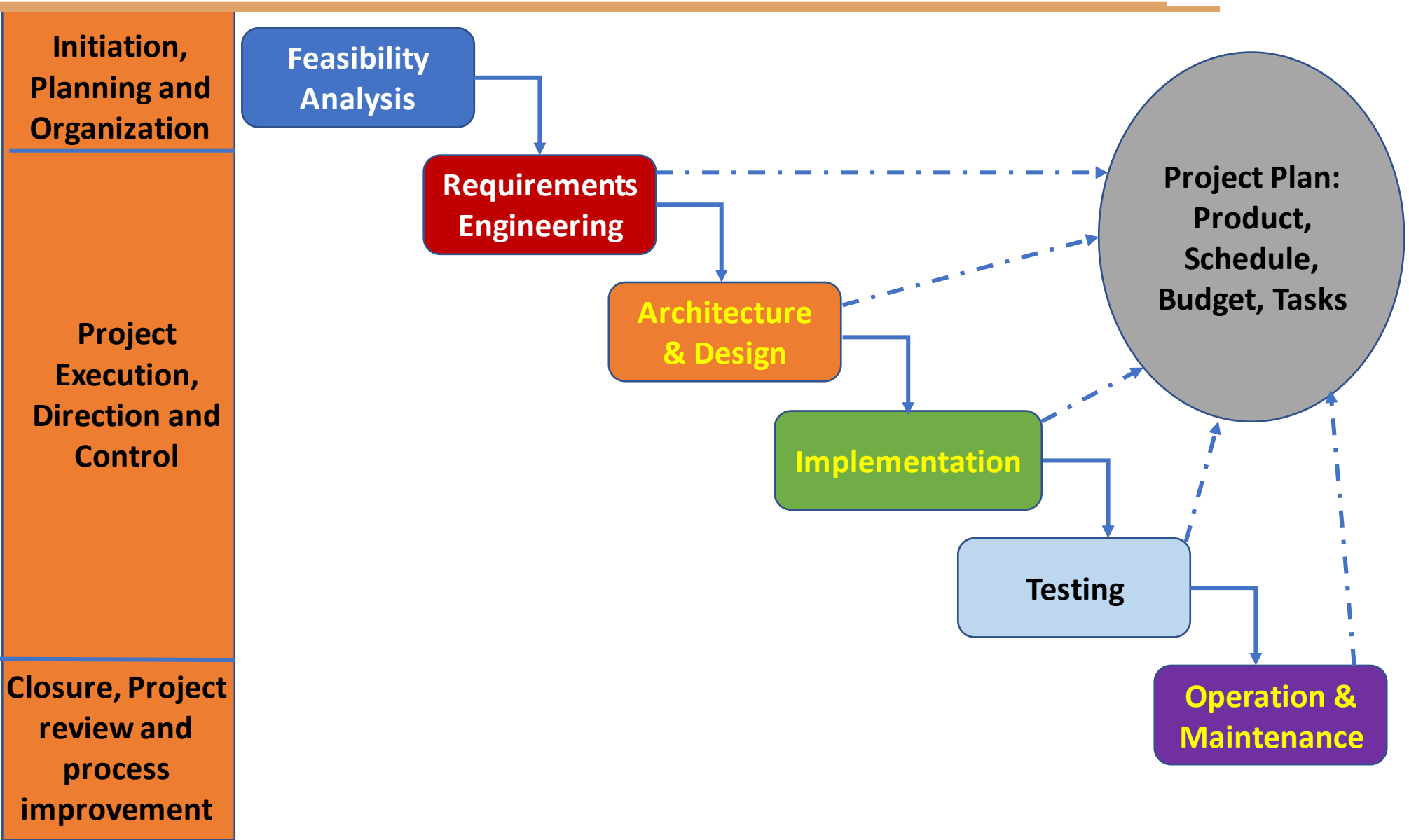
PDLC (Product Development Lifecyle)



Using a different lifecycle

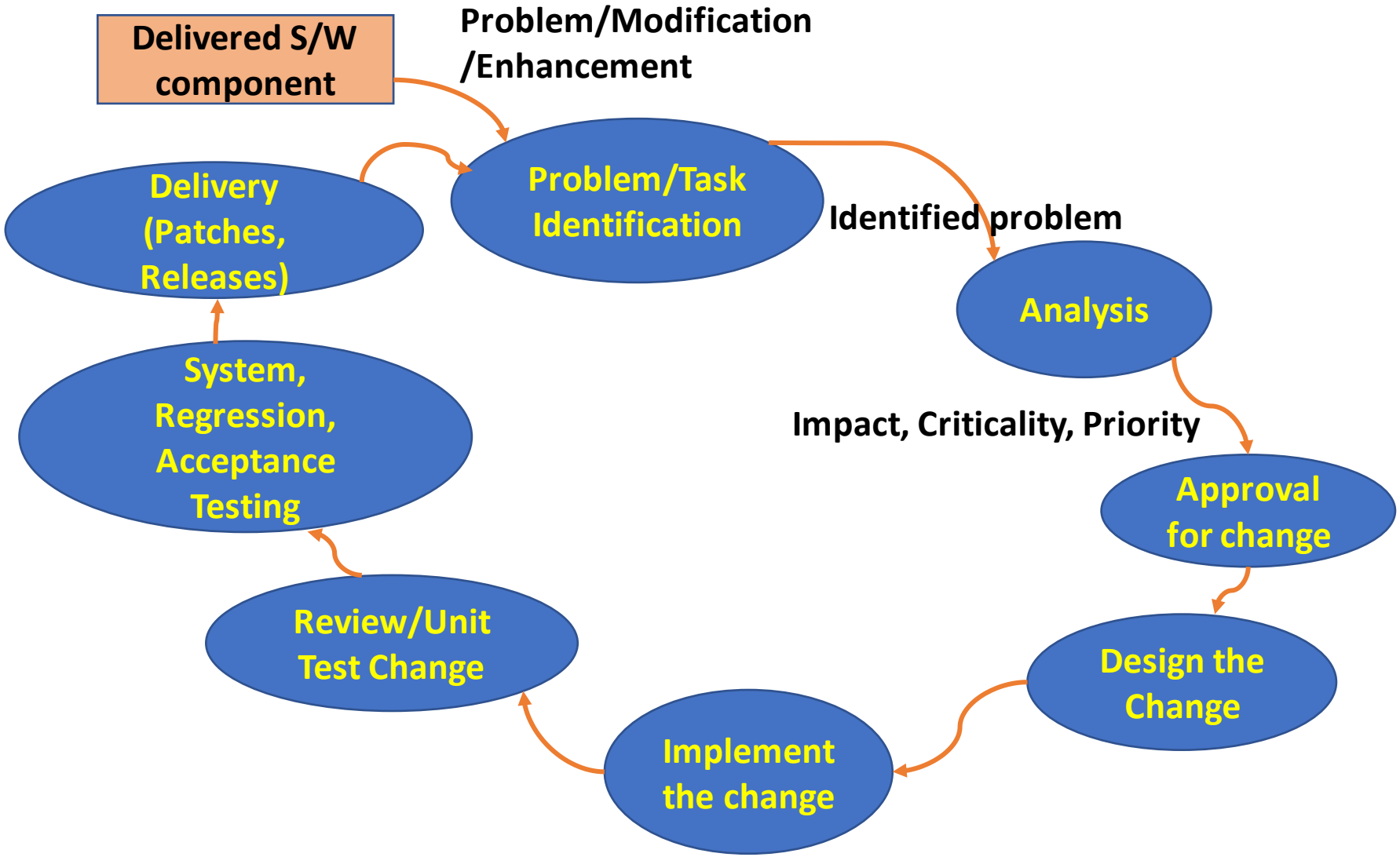
INTRODUCTION TO SOFTWARE ENGINEERING

Project Management in System Dev Lifecycle



INTRODUCTION TO SOFTWARE ENGINEERING

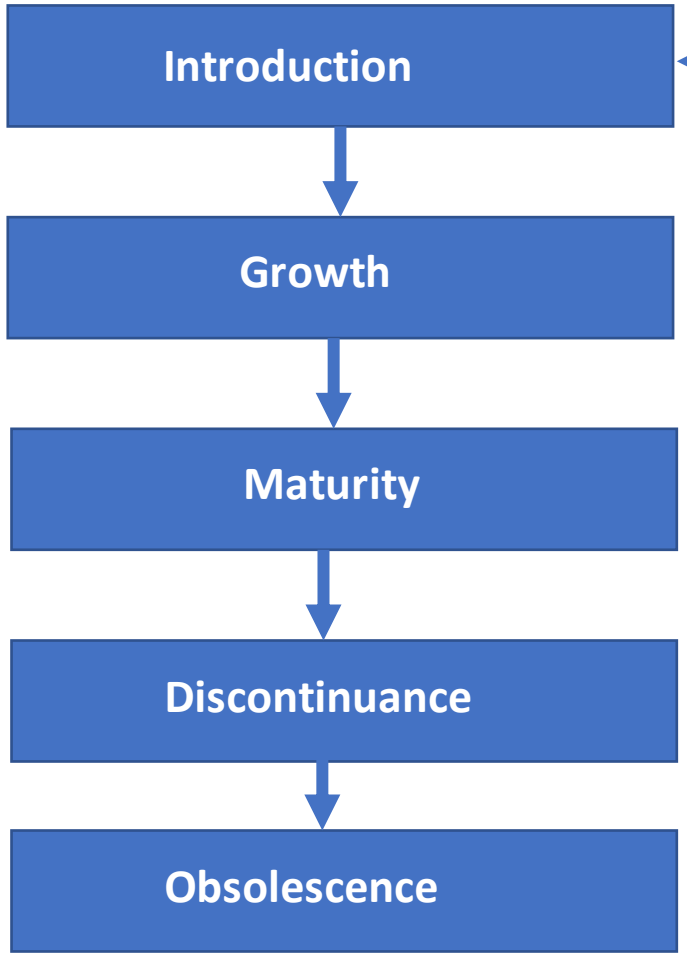
Software Maintenance Lifecycle



Product Lifecycle

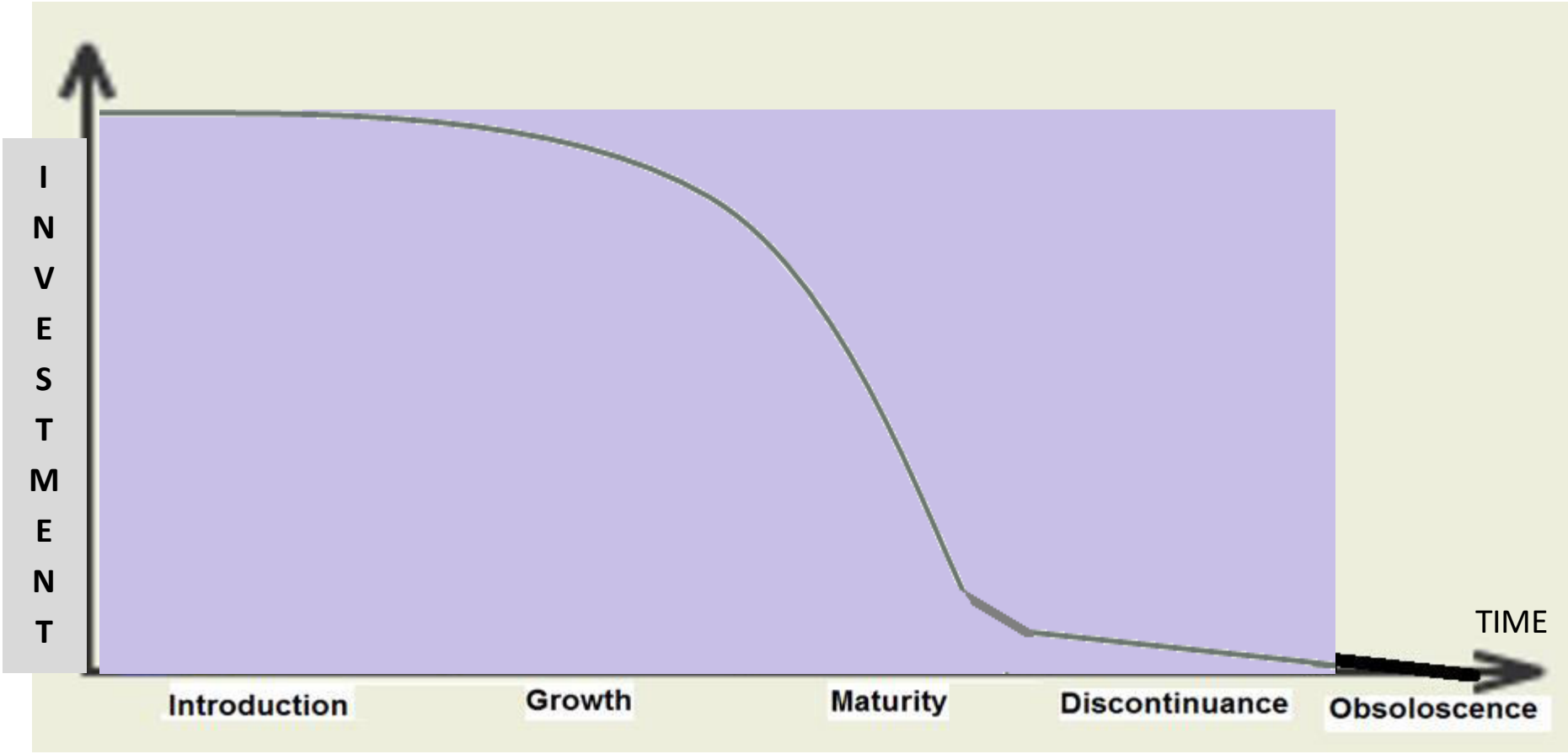
Attributes to Consider

1. Market capitalization
2. Sales
3. Investment
4. Competition
5. Profit
6. Support



Product built using a SDLC is initially released into market

Product Life Cycle Characteristics



THANK YOU

Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

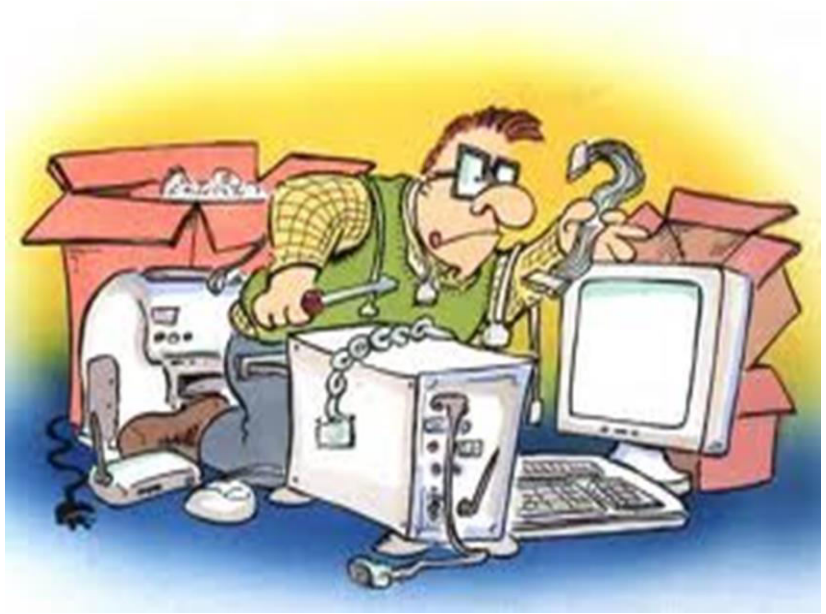
INTRODUCTION TO SOFTWARE ENGINEERING

Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

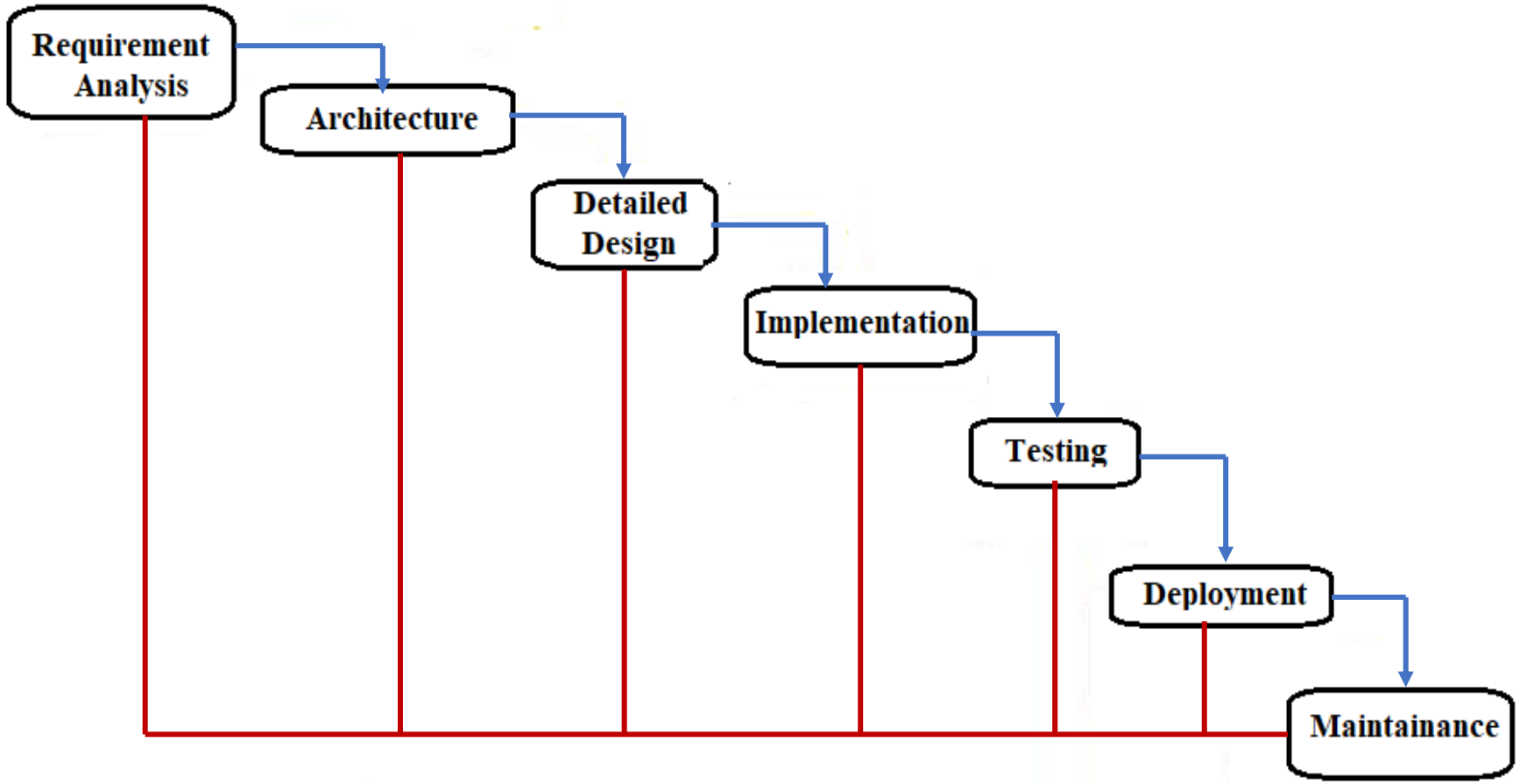
Legacy SDLCs : Waterfall, V Model, Prototyping, Incremental and Iterative Models



Phalachandra H. L
Department of Computer Science and Engineering

INTRODUCTION TO SOFTWARE ENGINEERING

Waterfall Model



INTRODUCTION TO SOFTWARE ENGINEERING

Waterfall Model

Advantages

- Simple
- Clear Identified phases as shown and easy to departmentalize and control
- Easy to manage due to the rigidity of the model
- Each phase has specific deliverables and a review process

Dis-Advantages

- Assumes requirements (both user & hardware) can be frozen
- Difficult to accommodate change and hence not very flexible
- Sequential. Move from one phase to another only after the completion of the previous phase
- Big bang approach .. No working software till late in the lifecycle
- High risk and uncertainty
- Poor model for long and ongoing projects

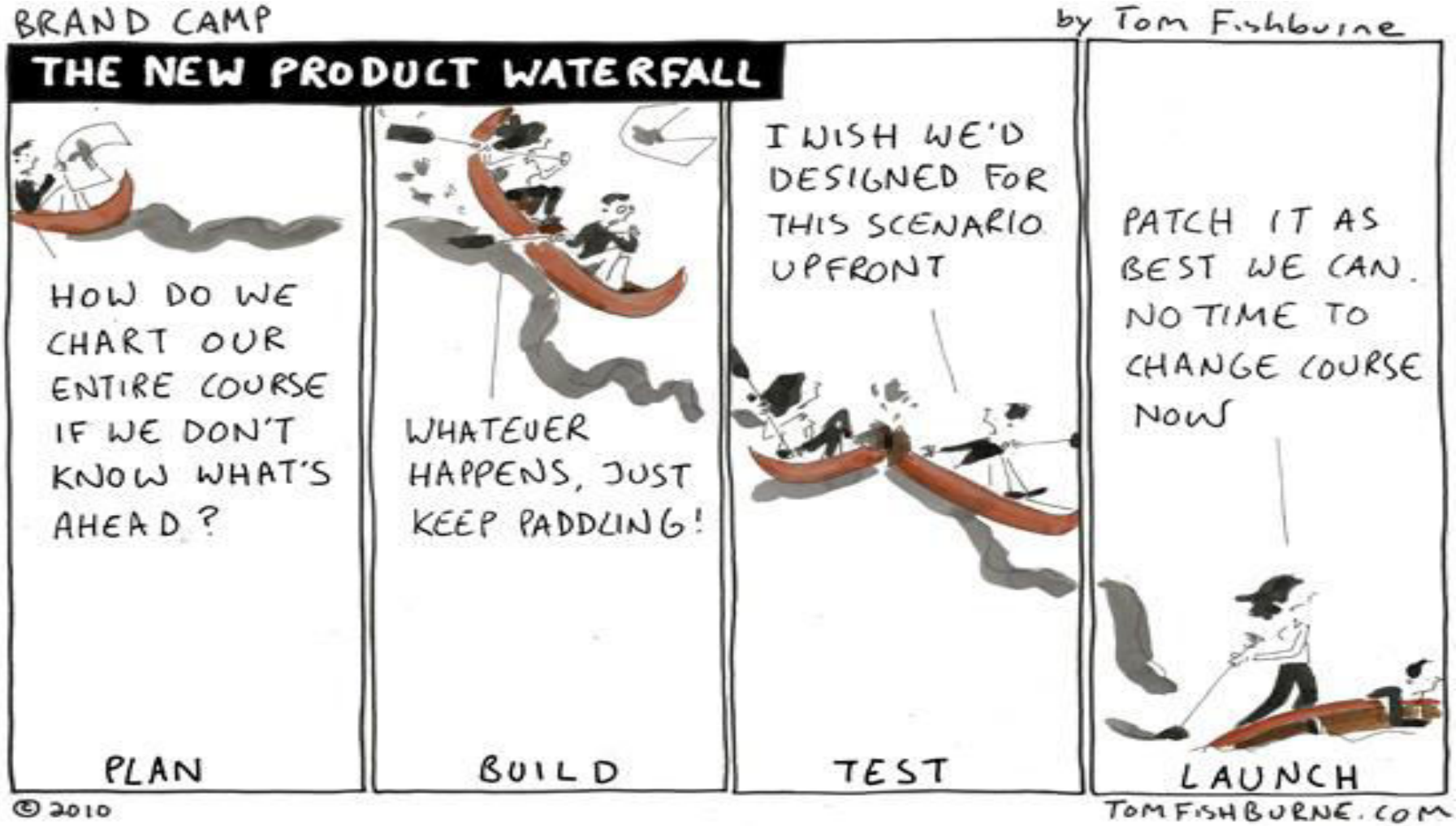
Waterfall Model

Usage

- Used (in pure form) for short projects where requirements are well understood
- Variants are used at a high level in large projects
- Product definition is stable and technology understood
- Variants used in large globally developed projects with other LC

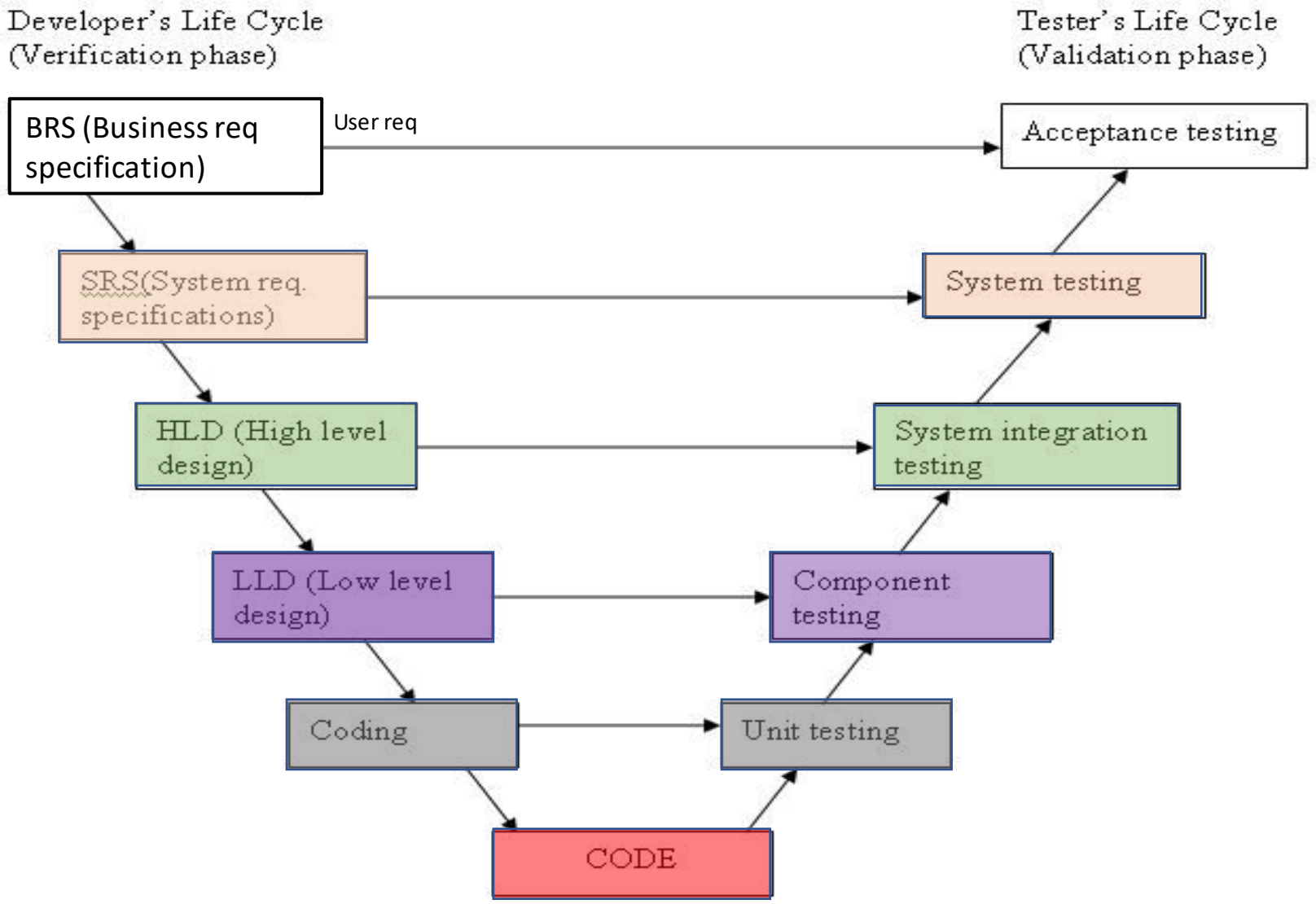
INTRODUCTION TO SOFTWARE ENGINEERING

Waterfall Model



INTRODUCTION TO SOFTWARE ENGINEERING

V Model



INTRODUCTION TO SOFTWARE ENGINEERING

V Model

Advantages

- Similar to waterfall model
- Test development activities like test design and static testing can happen before coding and the formal testing lifecycle
- Higher probability of success

Dis-Advantages

- Similar to waterfall model
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- Change through the process, necessitates changes in test documentation too.

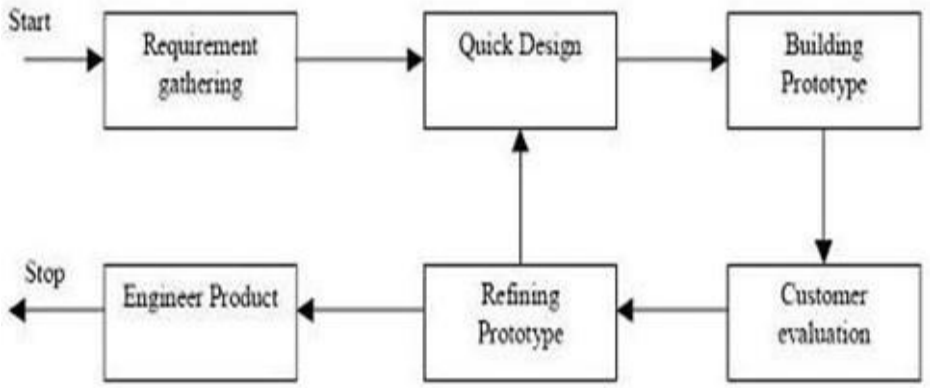
Usage

- Similar to waterfall model

INTRODUCTION TO SOFTWARE ENGINEERING

Prototyping

- Prototyping should be a relatively a cheap process (subset, non production quality etc.)
- Instead of freezing the requirements before a design or coding can proceed, a complete system prototype (with no or very less details) is built and to understand the requirements
- Throw-away and evolutionary prototypes



Prototyping Model

INTRODUCTION TO SOFTWARE ENGINEERING

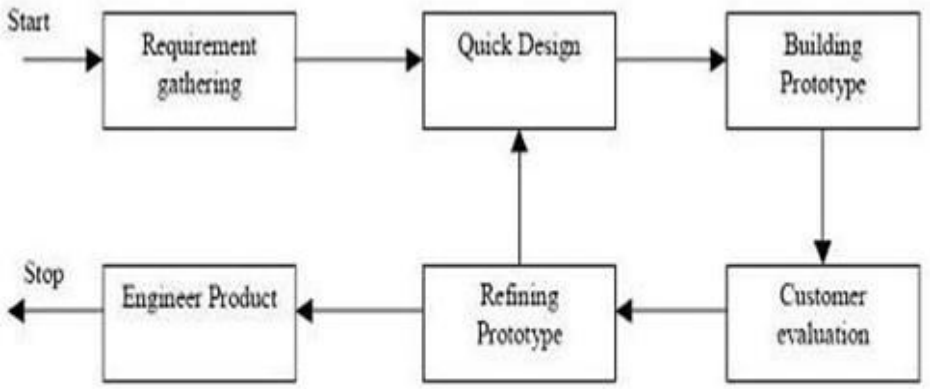
Prototyping

Advantages :

- Active involvement of the users
- Provides better risk mitigation (cost, understanding of the user requirements)
- Problems/defects are detected earlier and hence reduces time and cost
- Missing functionality easily identified and hence resulting system is full featured
- Systems are mostly more stable

Dis-Advantages:

- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans
- Performance of the resulting system may not be optimal

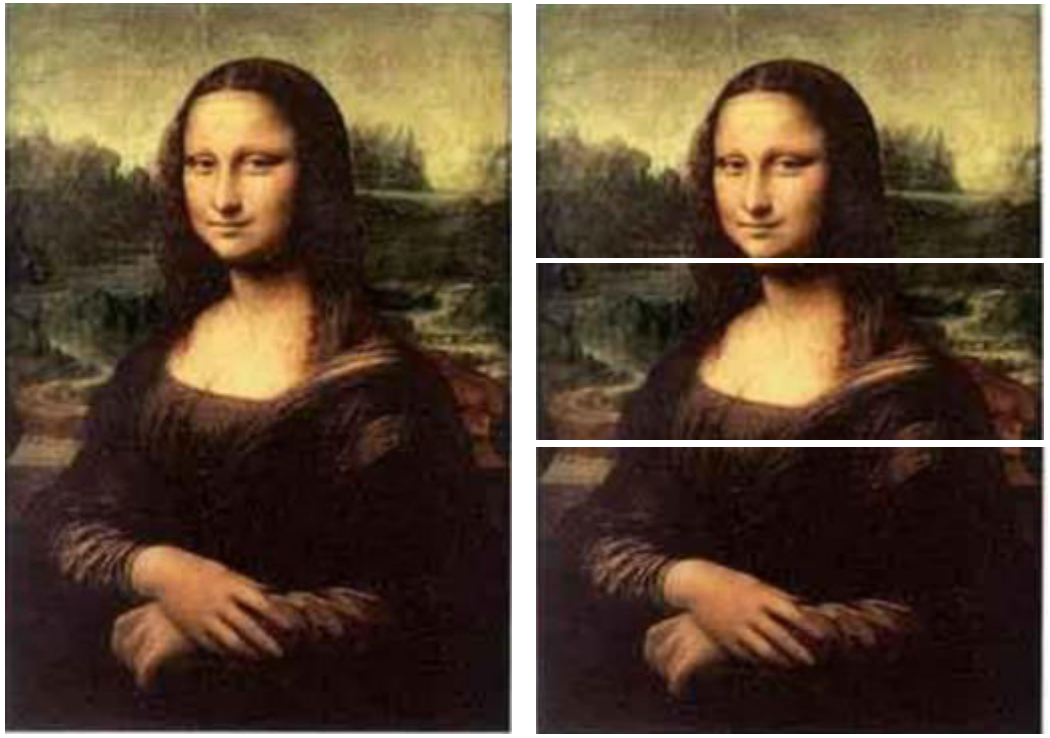


Prototyping Model

INTRODUCTION TO SOFTWARE ENGINEERING

Incremental Model

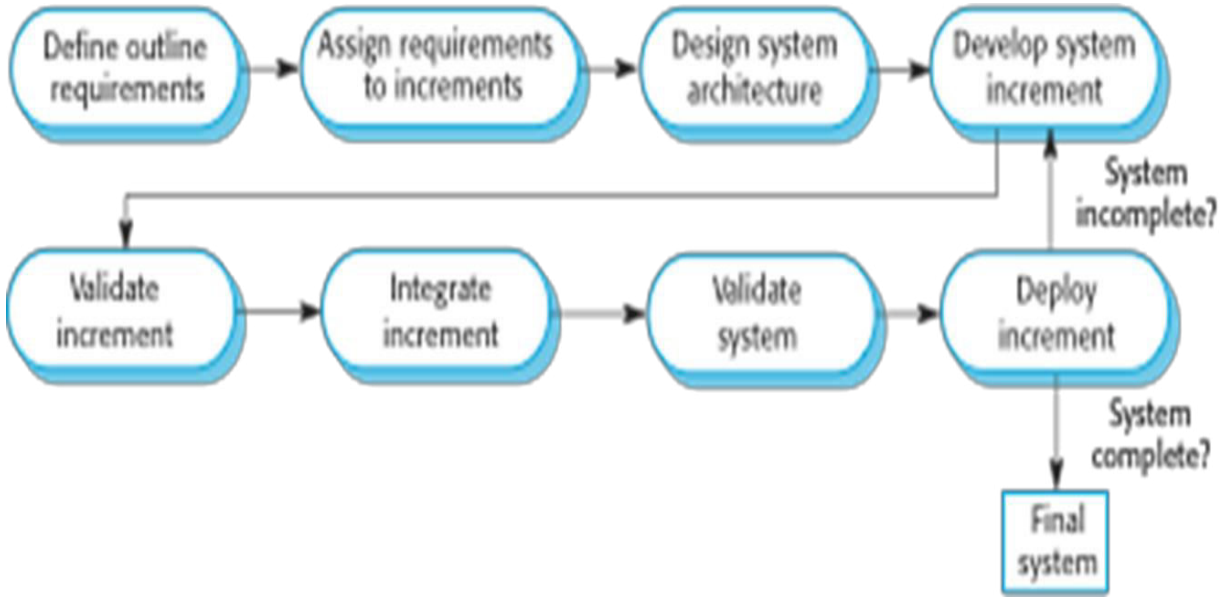
- In this model requirements are partitioned and an incremental plan is made



- Working version of software is produced during the first module, so we have a working software early on in the software life cycle.
- Each subsequent release of the module adds functionality to the previous release.
- This continuous integration continues till the complete system is achieved.

INTRODUCTION TO SOFTWARE ENGINEERING

Incremental Model



- The partitioned requirements can each have a development lifecycle
- Models like waterfall could be used/employed for each of the partition/increment
- Leveraged prototype for additional features would be a form of incremental model

Incremental Model

Advantages :

- Customer value – as early versions can be delivered to customers
- More flexible – less costly to change scope and requirements as, early increments act like prototypes
- Easier to test and debug during a smaller iteration.
- Easier to manage risk .. risky pieces are identified and handled during iteration ..customer can respond to it .. no big bang
- Continuous increments rather than monolithic
- Reduces over-functionality. Essential features first and additional when needed

Dis-Advantages:

- Needs good planning and design
- Needs clear and complete definition of the whole system before it can be broken down and built incrementally
- Total cost is higher than waterfall.
- Hard to identify common facilities that are needed by all increments
- Management visibility is reduced which could lead to surprises

Incremental Model

Usage :

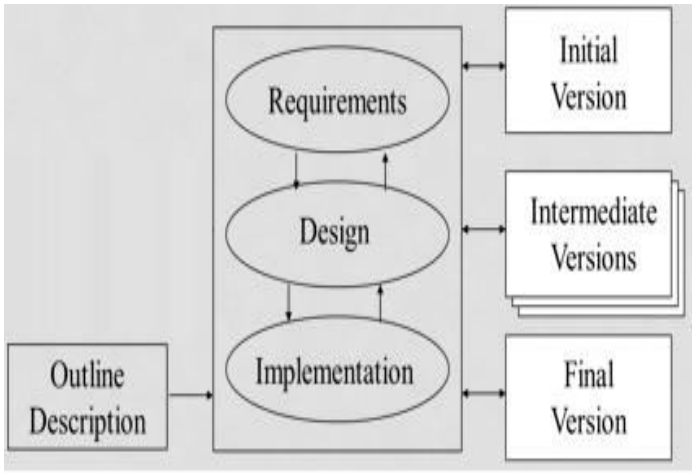
- In projects where major requirements are defined (although some details can evolve with time)
- When there is a need to get a product to the market early.
- When a new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals.

INTRODUCTION TO SOFTWARE ENGINEERING

Iterative Model (Evolutionary)

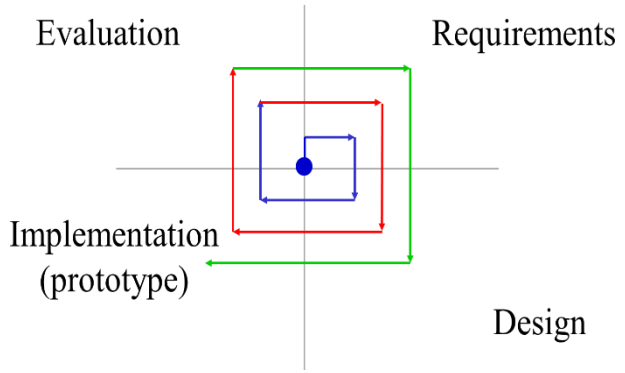
Initial implementation attempts to start with a skeleton of the product, followed by refinement through user feedback and evolution until system is complete.

- Built with dummy modules
- Rapid prototyping
- Successive refinement



Advantages

- Helps in identifying requirements, visualization of solution
- Supports risk mitigation
- Redesigning/Rework and defect flowing down the process is reduced
- Incremental investment



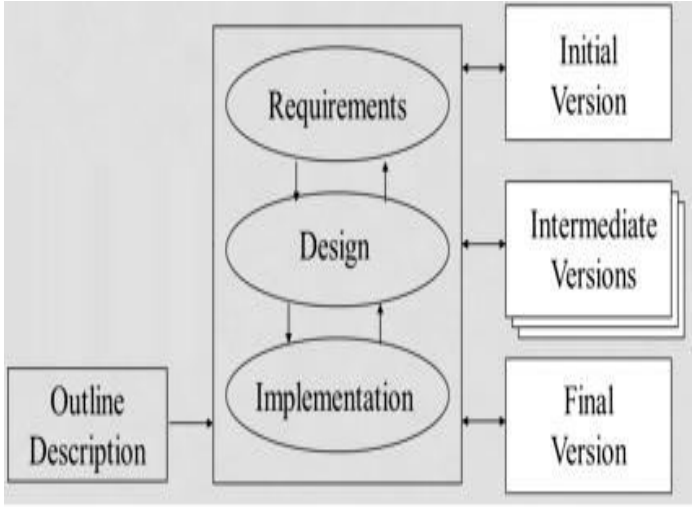
Iterative Model (Evolutionary)

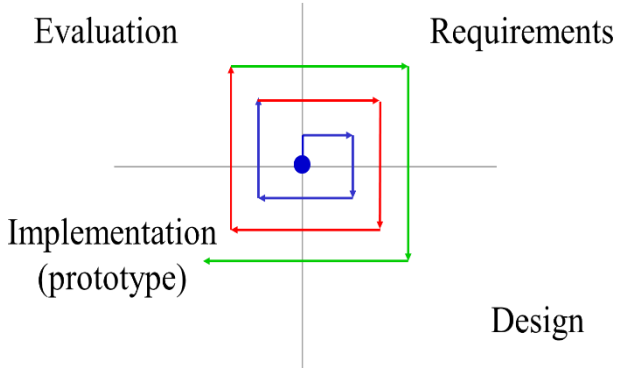
Disadvantages

- Each phase of an iteration is rigid with overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

Usage:

- Large projects

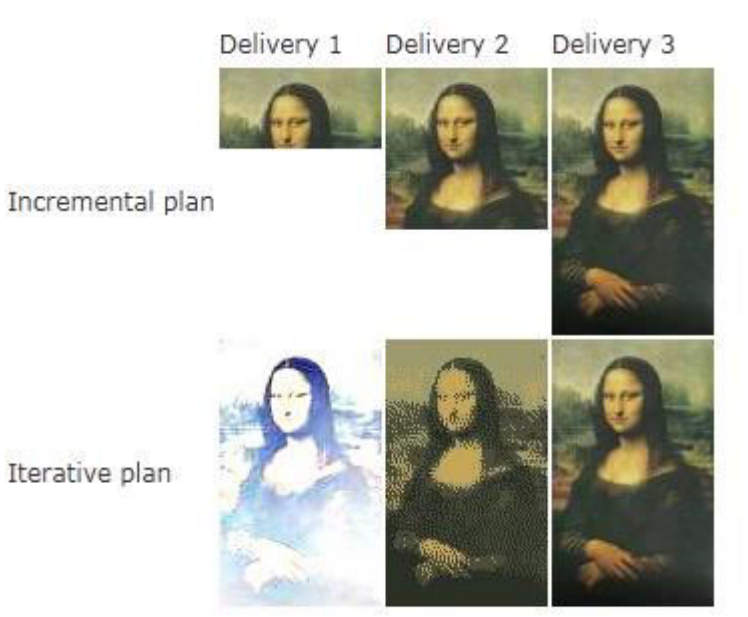


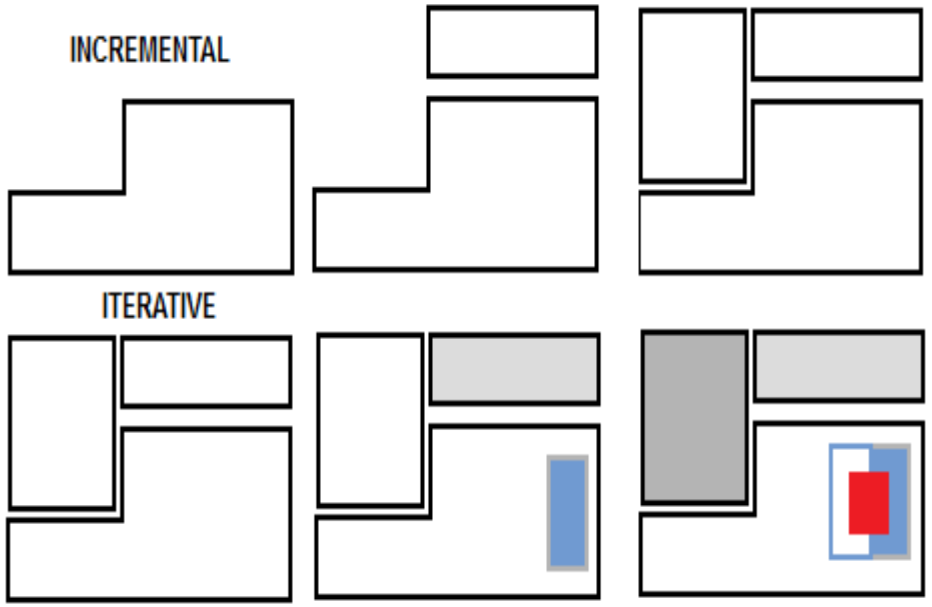


INTRODUCTION TO SOFTWARE ENGINEERING

Contrasting Incremental and Iterative Models

- Incremental approach does not require going back and changing things that are already delivered whereas Iterative continues to revisit and refining everything every time
- Incremental focuses on things not implemented, whereas Iterative is more details of the same
- Incremental does not leverage on the experience/knowledge till then, whereas iterative leverages learning's.





THANK YOU

Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

INTRODUCTION TO SOFTWARE ENGINEERING

Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

PES UNIVERSITY ONLINE

Agile Philosophy

Phalachandra H. L
Department of Computer Science and Engineering

Limitations of most legacy Lifecycles

- Traditional models are based on predictive software development methods
- Upfront planning
- More suitable where there is a clear definition of what needs to be achieved and things are not changing fast
- Models do not facilitate periodic customer interactions
- Suited for very large projects with complex dependencies
- Suitable for some kinds of organizational environments like global, distributed, to some organizational structures etc.
- Product lifecycle and its eco-system
- People and Skill perspectives
- Regulatory perspectives

INTRODUCTION TO SOFTWARE ENGINEERING

Agile

"Agile" is an umbrella term used to describe a variety of methods that encourage

- Continual realignment of development goals with the needs and expectations of the customer.
- Its also aimed at reducing massive planning overhead to allow fast reactions to change.

CONTEXT

- World is chaotic
- Change is inevitable and is the constant
- Deliver value to customer as quickly as possible

INTRODUCTION TO SOFTWARE ENGINEERING

Agile Manifesto

read - Is valued more than

- **Individuals and interactions** over **Processes and tools**

- **Working software** over **Comprehensive documentation**

- **Customer collaboration** over **Contract negotiation**

- **Responding to change** over **Following a plan**

- **Focus on Simplicity** in **Both the product and the process**

INTRODUCTION TO SOFTWARE ENGINEERING

Pun on Agile

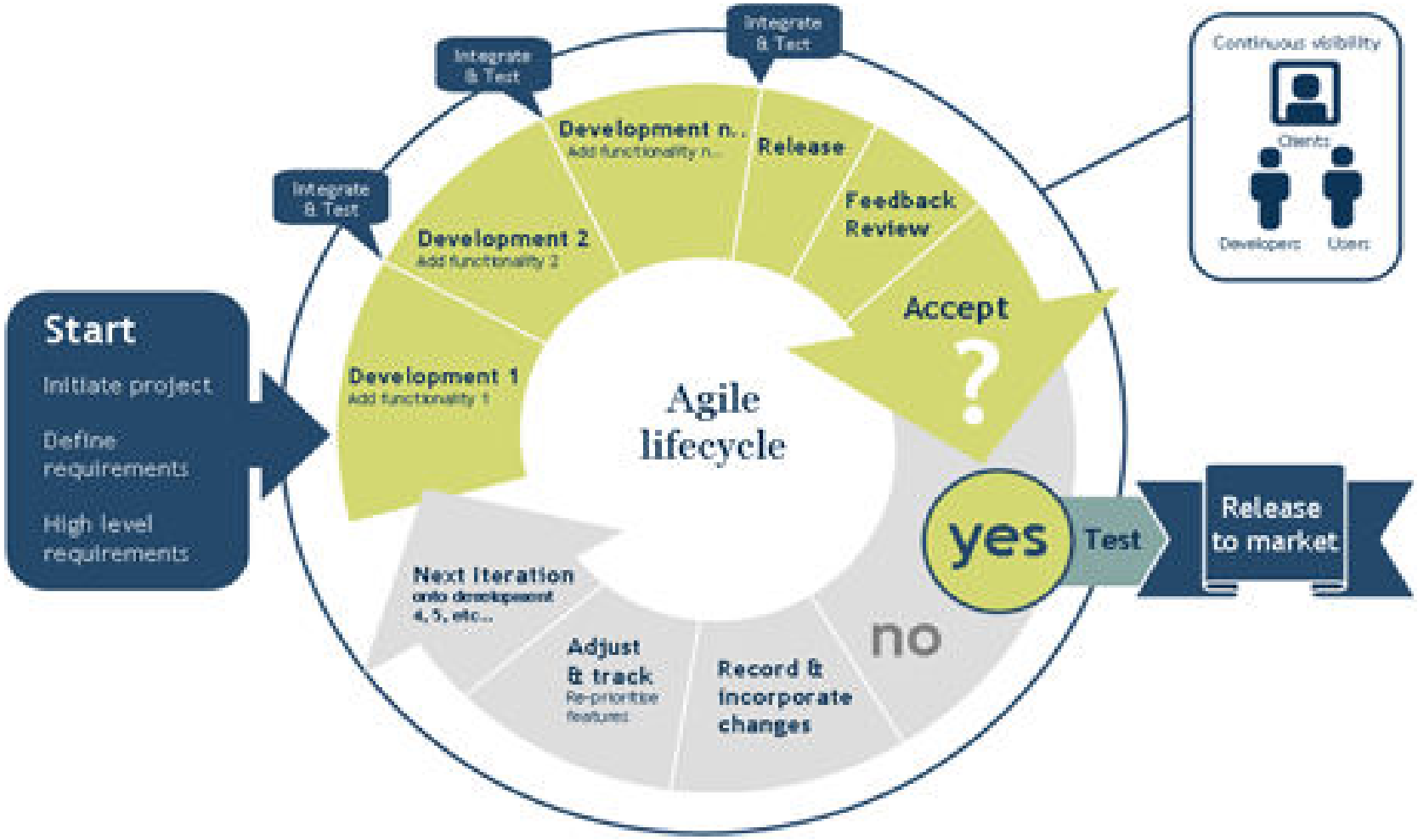
INTRODUCTION TO SOFTWARE ENGINEERING

Agile Methodologies

Pros	Cons
▪ Is a very realistic approach to software development ▪ Promotes teamwork and cross training. ▪ Functionality can be developed rapidly and demonstrated. ▪ Resource requirements are minimum. ▪ Suitable for fixed or changing requirements ▪ Delivers early partial working solutions. ▪ Good model for environments that change steadily. ▪ Minimal rules, documentation easily employed. ▪ Enables concurrent development and delivery within an overall planned context. ▪ Little or no planning required ▪ Easy to manage ▪ Gives flexibility to developers	▪ Not suitable for handling complex dependencies. ▪ More risk of sustainability, maintainability and extensibility. ▪ An overall plan, an agile leader and agile PM practice is a must without which it will not work. ▪ Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. ▪ Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. ▪ There is very high individual dependency, since there is minimum documentation generated. ▪ Transfer of technology to new team members may be quite challenging due to lack of documentation.

INTRODUCTION TO SOFTWARE ENGINEERING

Agile based Lifecycles



SCRUM

- Scrum is an agile methodology or framework for developing, delivering and sustaining software components and products
- Scrum is an iterative approach towards software development
- Scrum words comes from rugby where it's a formulation of play where everyone has a part and you huddle periodically to take stock
- Scrum provides mechanisms to apply agile practices
- Scrum is defined by
  - Roles
  - Events (ceremonies)
  - Artifacts
  - Rules

SCRUM : Roles

Organization is split into small, cross-functional, self-organizing teams.

Scrum Team

Scrum Team

Scrum Team

Scrum Master

Product/
Project Owner

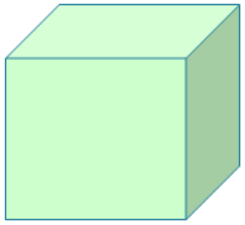
Scrum Team
- made up of contributors to the deliverable
- responsible for delivering shippable increments
- Self organized
- Cross Functional

Scrum Master
- not a manager
- facilitator
- removes impediments
- facilitates meetings
- ensures team adheres to scrum theory and practices

Product Owner
- is the voice or rep of the stakeholder/team
- Create/manages Product Backlog

SCRUM : Artifacts

Split your work into a list of small, concrete deliverables. Sort the list by priority and estimate the relative effort of each item.



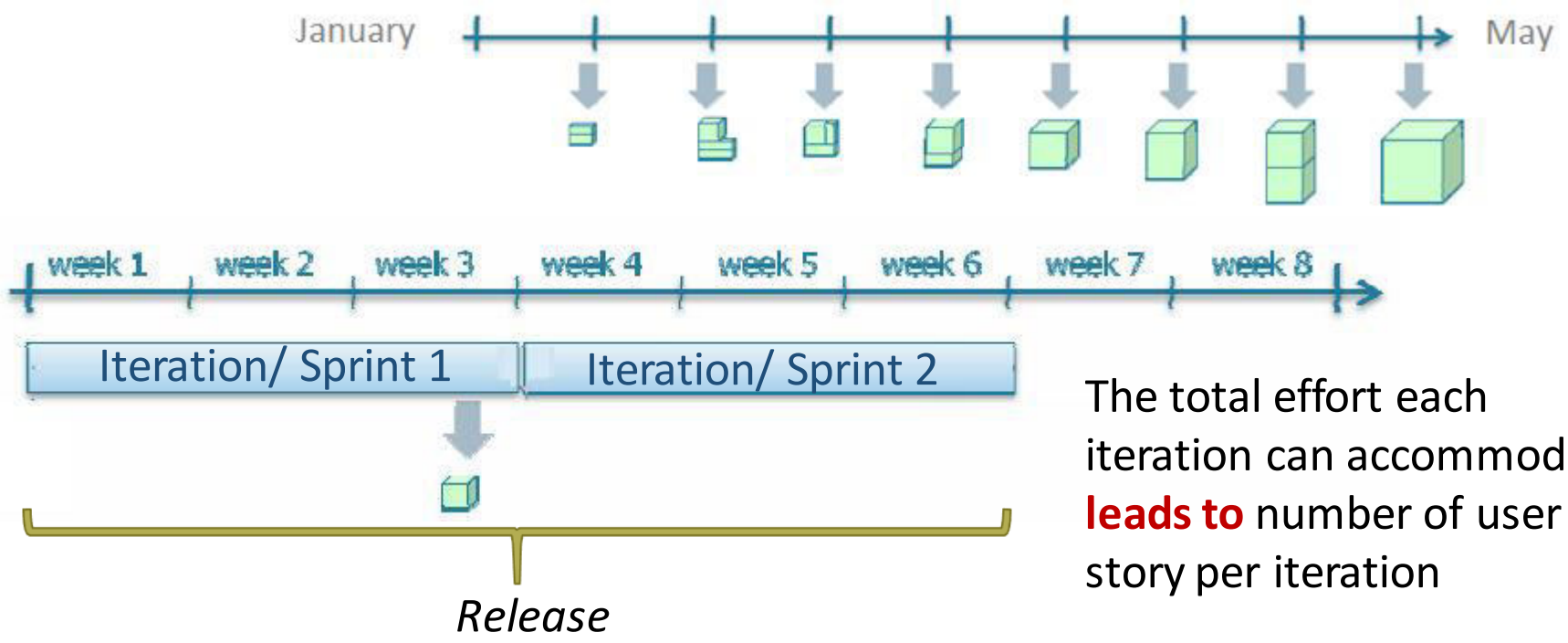
Product backlog

- The project/product is described as a list of features also called **Product backlog**
- The features are described in terms of **user stories**
- The scrum team **estimates** the **work** associated with each story and **ranks** them in the order of importance.
- The **weighted-ranked** features/stories in the backlog results in **roadmap**
- The features/stories in the backlog planned for a sprint is the **sprint backlog**

INTRODUCTION TO SOFTWARE ENGINEERING

SCRUM : Events

Split time into short fixed-length iterations (called **Sprints**) usually 2 – 4 weeks, with potentially shippable code demonstrated after each iteration. (its also called time-boxing)



The total effort each iteration can accommodate **leads to** number of user story per iteration

One **release** may contains **number of iterations**

Sprint planning meeting is the planning meeting to determine which of the product backlog items will be worked on and delivered in the sprint iteration

INTRODUCTION TO SOFTWARE ENGINEERING

SCRUM : Events

Daily scrum meeting or **Standup meeting** to discuss

- **What did you do yesterday?**
- **What will you do today?**
- **Any obstacles?**

End of the sprint has a finished "**shippable product**"

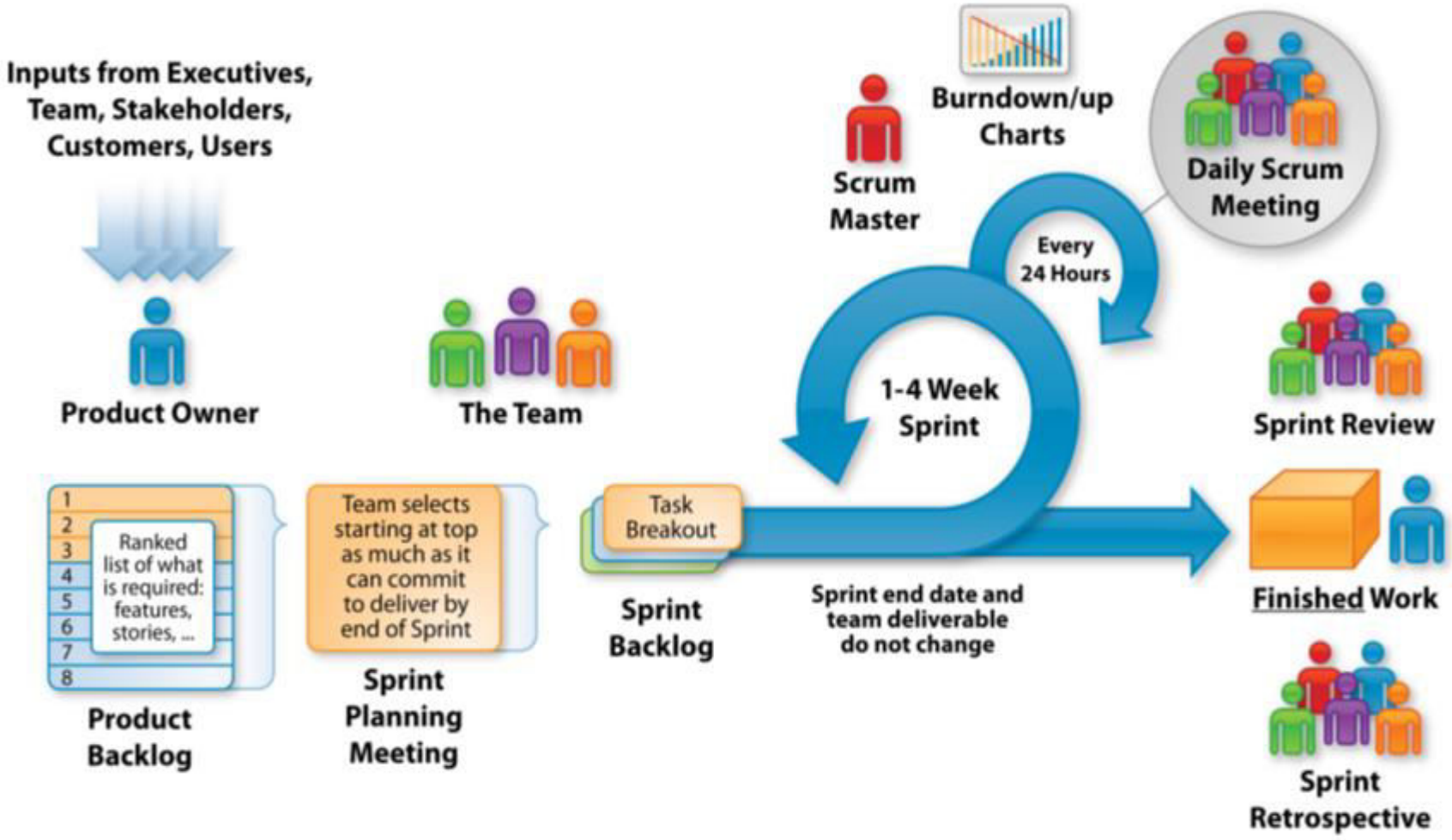
Sprint review meeting demonstrates story features. Product owner evaluates against preset criteria's. Gets feedback from clients and stakeholders and ensures the delivered increment meets the business need and helps support reprioritizing of the product backlog and optimize the release plan if needed.

Optimize the process by having a **Sprint retrospective** after each iteration. It is the final team meeting in the Sprint to determine what went well, what didn't go well, and how the team can improve in the next Sprint. Attended by the team and the ScrumMaster, the Retrospective is an important opportunity for the team to focus on its overall performance and identify strategies for continuous improvement on its processes.

INTRODUCTION TO SOFTWARE ENGINEERING

SCRUM : In a nutshell

So instead of a **large group** spending **a long time** building a **big thing**, we have a **small team** spending a **short time** building a **small thing** but integrating regularly to see the whole



INTRODUCTION TO SOFTWARE ENGINEERING

SCRUM : Summarizing the framework

- Scrum is a lightweight agile framework with broad applicability for managing and controlling *iterative* and *incremental* projects of all types.
- Demos can be provided after every Sprint
- Small working teams to maximize communication, minimize overheads, sharing of tacit and informal knowledge
- Adaptable to technical and business changes
- Yields frequent software increments that can be inspected, adjusted, tested, documented and built on.
- Development work and people who perform it are partitioned into clean low coupling partitions/packets
- Constant testing and documentation is performed as the product is built.
- Ability to declare a product done whenever done
- Teams start with simple Scrum tools like a whiteboard or a spreadsheet to manage the product backlog and the progress of the sprint backlog items.

THANK YOU

Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

INTRODUCTION TO SOFTWARE ENGINEERING

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

INTRODUCTION TO SOFTWARE ENGINEERING

PES UNIVERSITY ONLINE

CBSE - Component Based Software Engineering

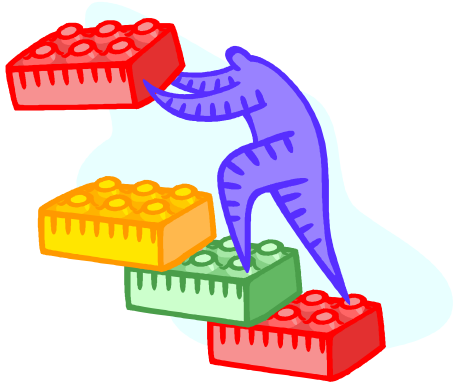
Prof. Phalachandra H. L
Department of Computer Science and Engineering

INTRODUCTION TO SOFTWARE ENGINEERING

Lego

- Set of building blocks in different shapes and colors
- Can be
- Con
  corr block

Common Theme
- **Reuse**
- **Quickly assemble Models**
- **Build Complex models**



⇒ LEGO blocks are generic and easily composable

- LEGO can be combined with LEGO, not with Meccano or any other blocks

INTRODUCTION TO SOFTWARE ENGINEERING

What and why CBSE

- **What is CBSE**

  Component-based software Engineering approach is a reuse based approach to define, implement or select of-the shelf components and integrate/compose loosely coupled independent components into systems

- **Motivation or the need for CBSE**
  - Increase in complexity of systems
  - Need for a quicker TAT .. So reuse rather than re-implement and shorten development time

Advantages and Problems with CBSE

Advantages of CBSE

- Components could be used as a black box and hence reduces complexity of systems
- Since developers reuse components, this reduces the development time
- Increases quality (due to explicit dependencies, reuse) especially evolvability and maintainability
- Increases productivity

Problems with CBSE

- Component trustworthiness
- Component certification
- Emergent property prediction
- Requirement trade off

INTRODUCTION TO SOFTWARE ENGINEERING

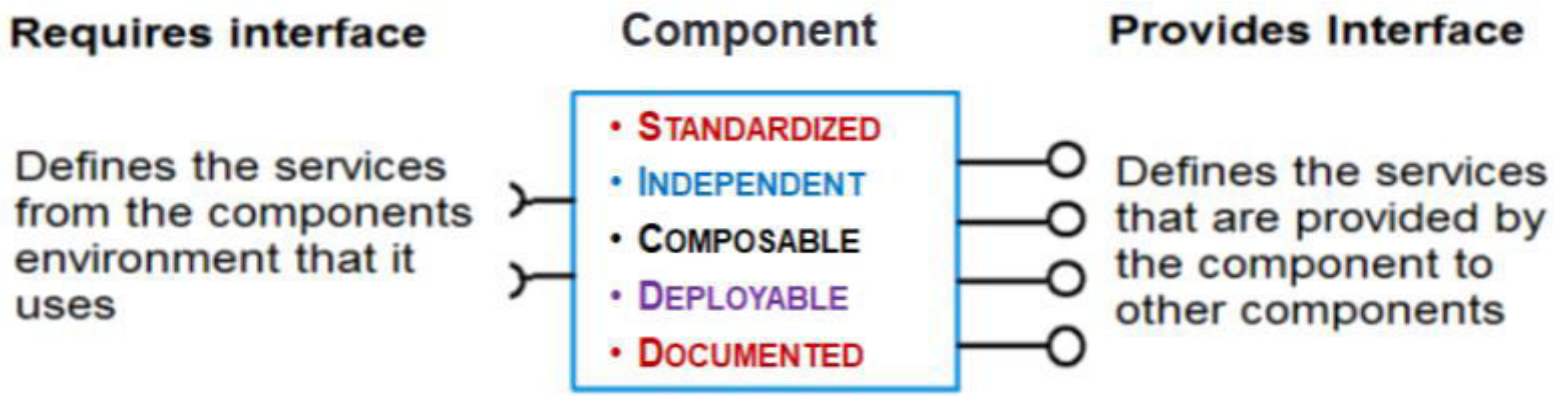
Essentials of CBSE to be successful

- Independent components that are completely specified by the public interfaces (implementation is hidden and hence can be changed)
- Component standards that facilitate the integration of components
- Middleware that provides software support for component integration
- Development process that is geared up to CBSE (interleaving of requirement and system design –trade-off requirements/existing services etc.)

INTRODUCTION TO SOFTWARE ENGINEERING

A Software Component

- Implements a functionality without regard to where the component is executing or its programming language
  - Its an independent executable entity that can be made up of one or more executable objects;
  - The component interface is published and all interactions are through the published interface;
  - Has explicit dependencies through “required” interfaces and “provides” (the services that are provided by the component to other components)



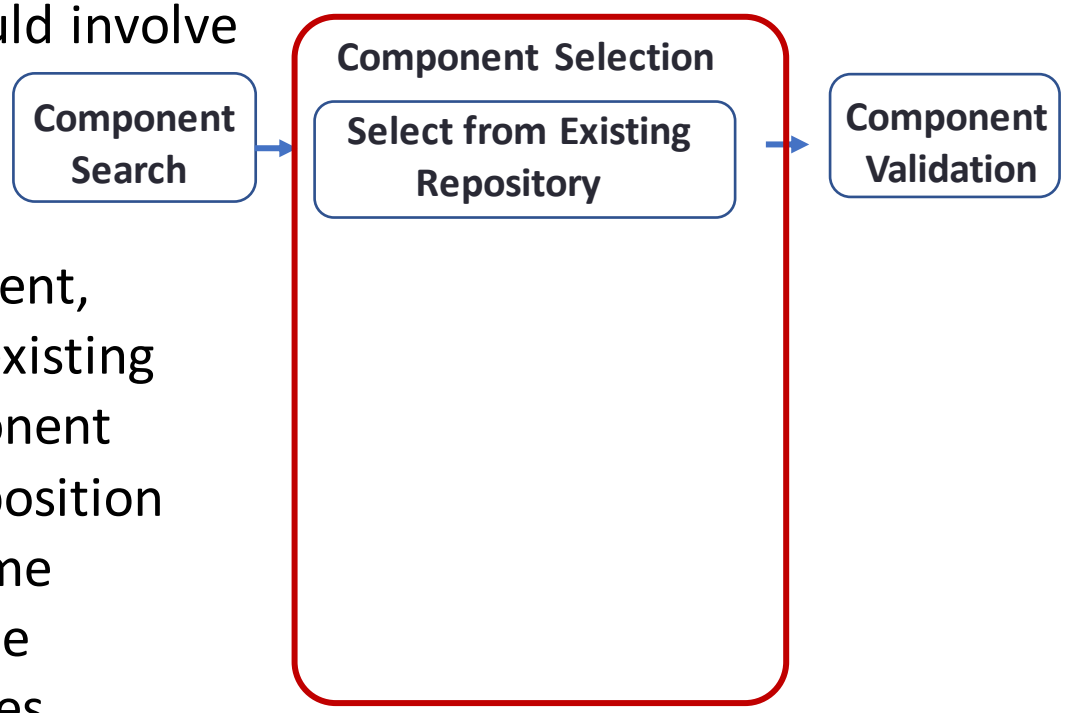
INTRODUCTION TO SOFTWARE ENGINEERING

Identifying the Software Component

Identifying the components could involve

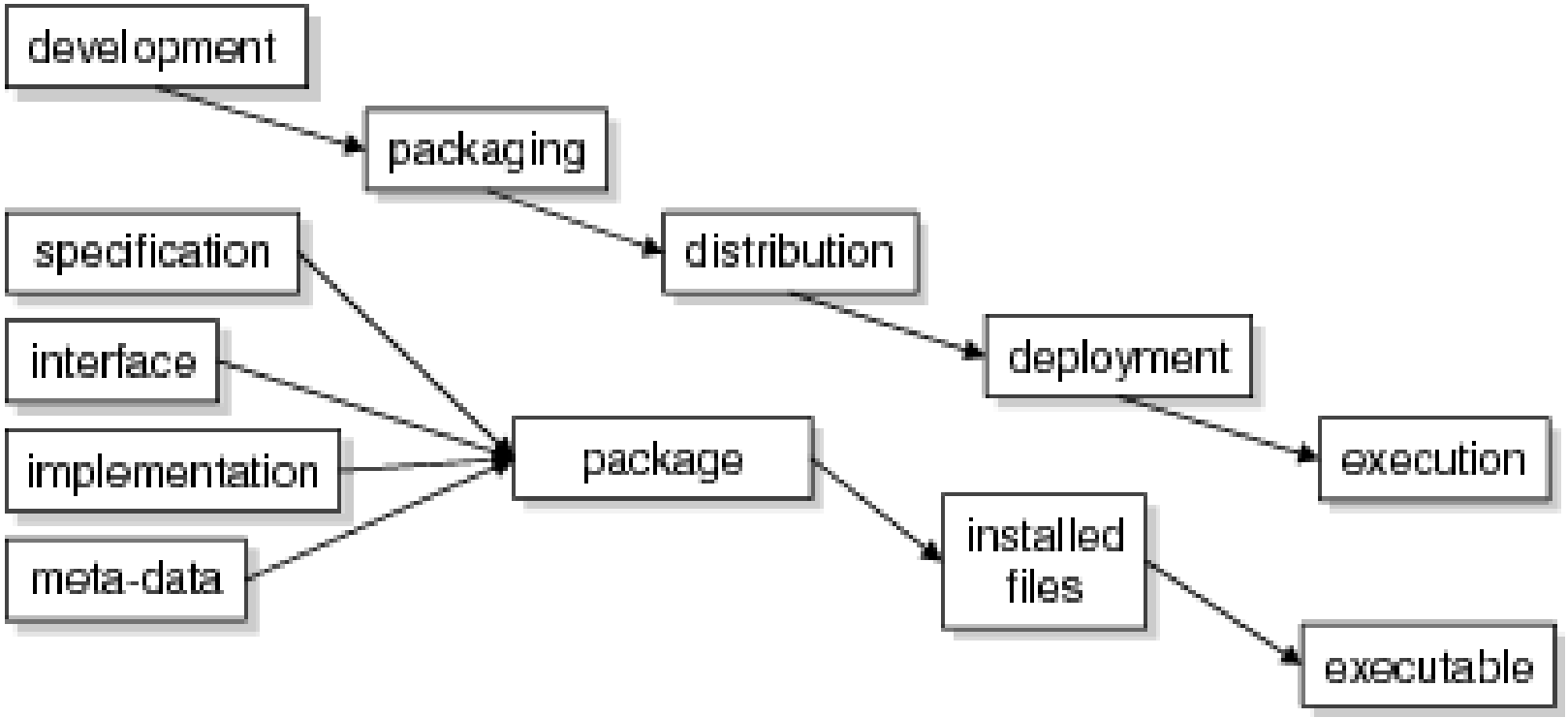
1. Searching for the component
2. Selection of the component
3. If there are no ready component, Compose components with existing components to create component
4. It could be a sequential composition and additive or may need some adapters or “glue” to reconcile different component interfaces

5. Validate the component



INTRODUCTION TO SOFTWARE ENGINEERING

Component Development Stages



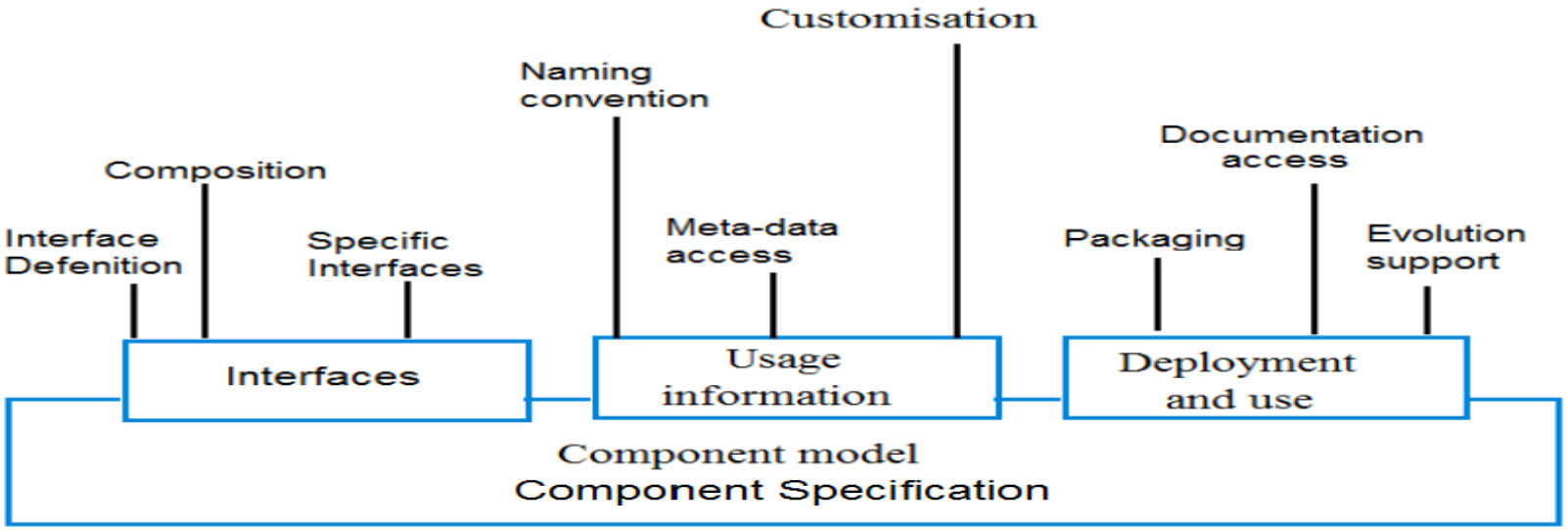
- Across these stages, components are represented in different forms:

  During development : UML etc.
  When packaging : in a .zip file etc.
  In the execution stage: blocks of code and data

INTRODUCTION TO SOFTWARE ENGINEERING

Elements of a Component Model

Defines the types of building block, which can be composed with other components to create a software system



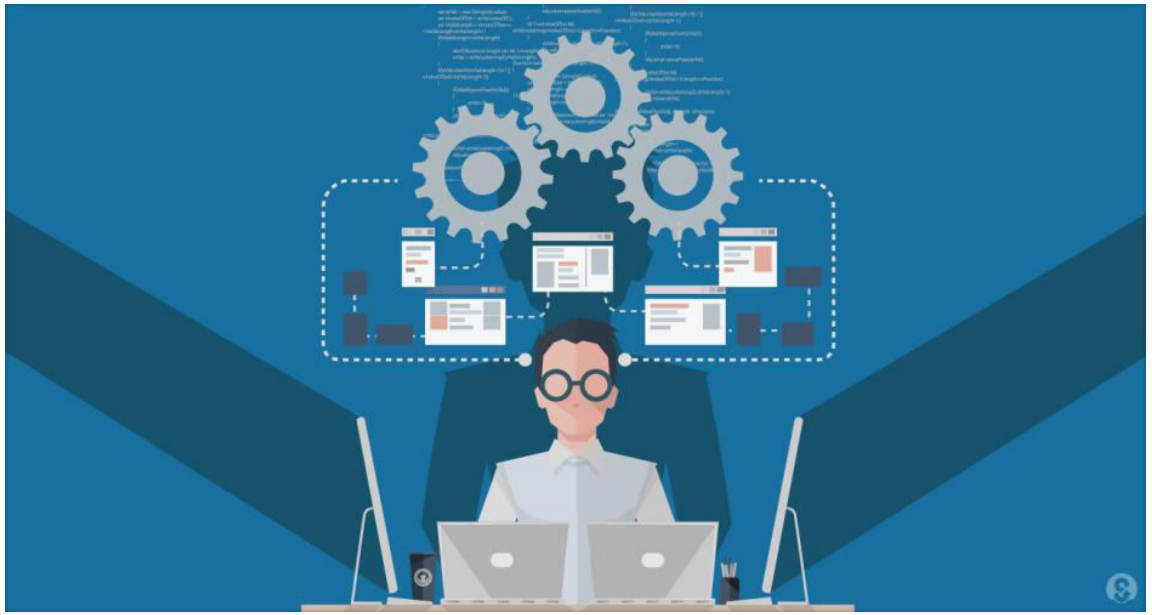
Interfaces : Defines how component can interact and also defines operation names, parameters and exceptions, which should be included in the interface definition

Usage : In order for components to be distributed and accessed remotely, they need to have a globally unique name or handle associated with them

Deployment : The component model includes a specification of how components should be packaged for deployment as independent, executable entities

INTRODUCTION TO SOFTWARE ENGINEERING

Service Oriented Architecture



Prof. Phalachandra H. L
Department of Computer Science and Engineering

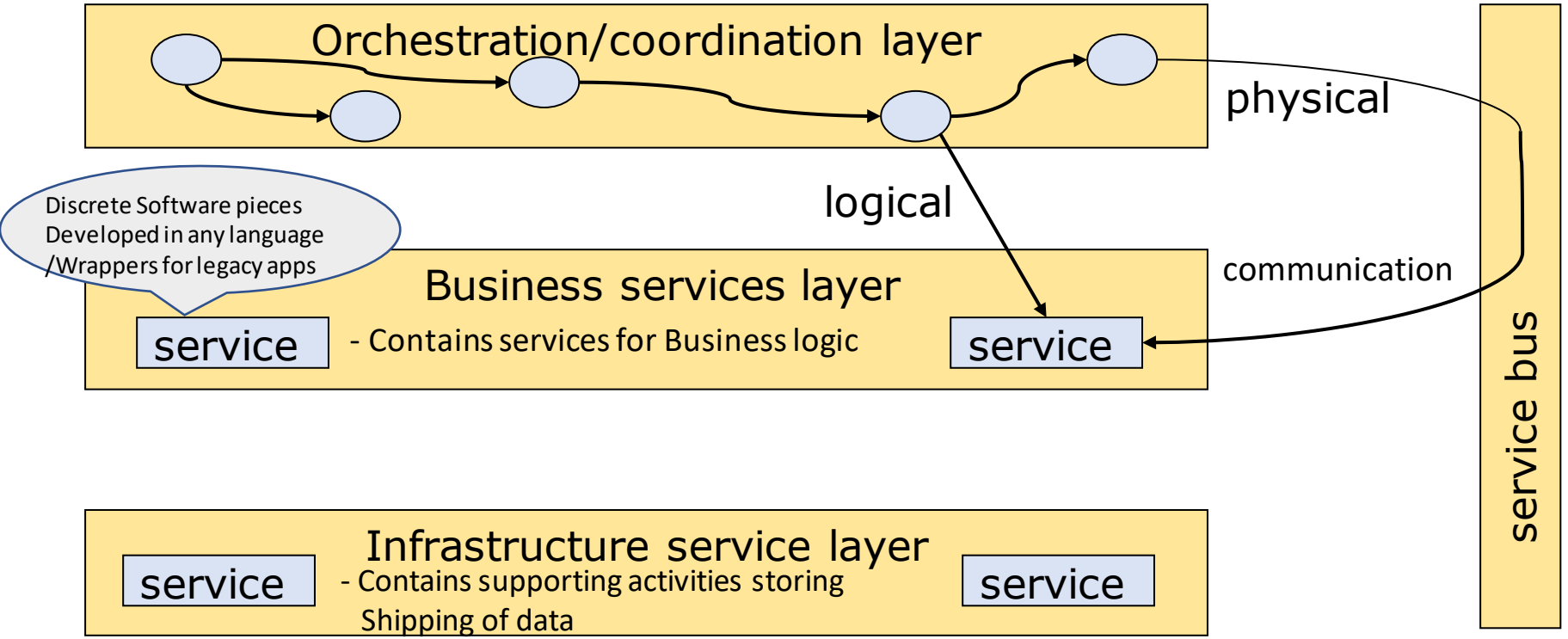
INTRODUCTION TO SOFTWARE ENGINEERING

Service Oriented Architecture

- SOA is a software design methodology based on structured collections of discrete software modules, known as services, that collectively provide the complete functionality of a large or complex software application.
- Its a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
- These could be implemented as discrete pieces of software or components written in any language capable of performing a task
- These could be implemented as “callable entities” or application functionalities accessed via exchange of messages
- These may also be application functionality with wrappers which can communicate through messages
- Software is deployed “somewhere”, on-need basis .. SaaS

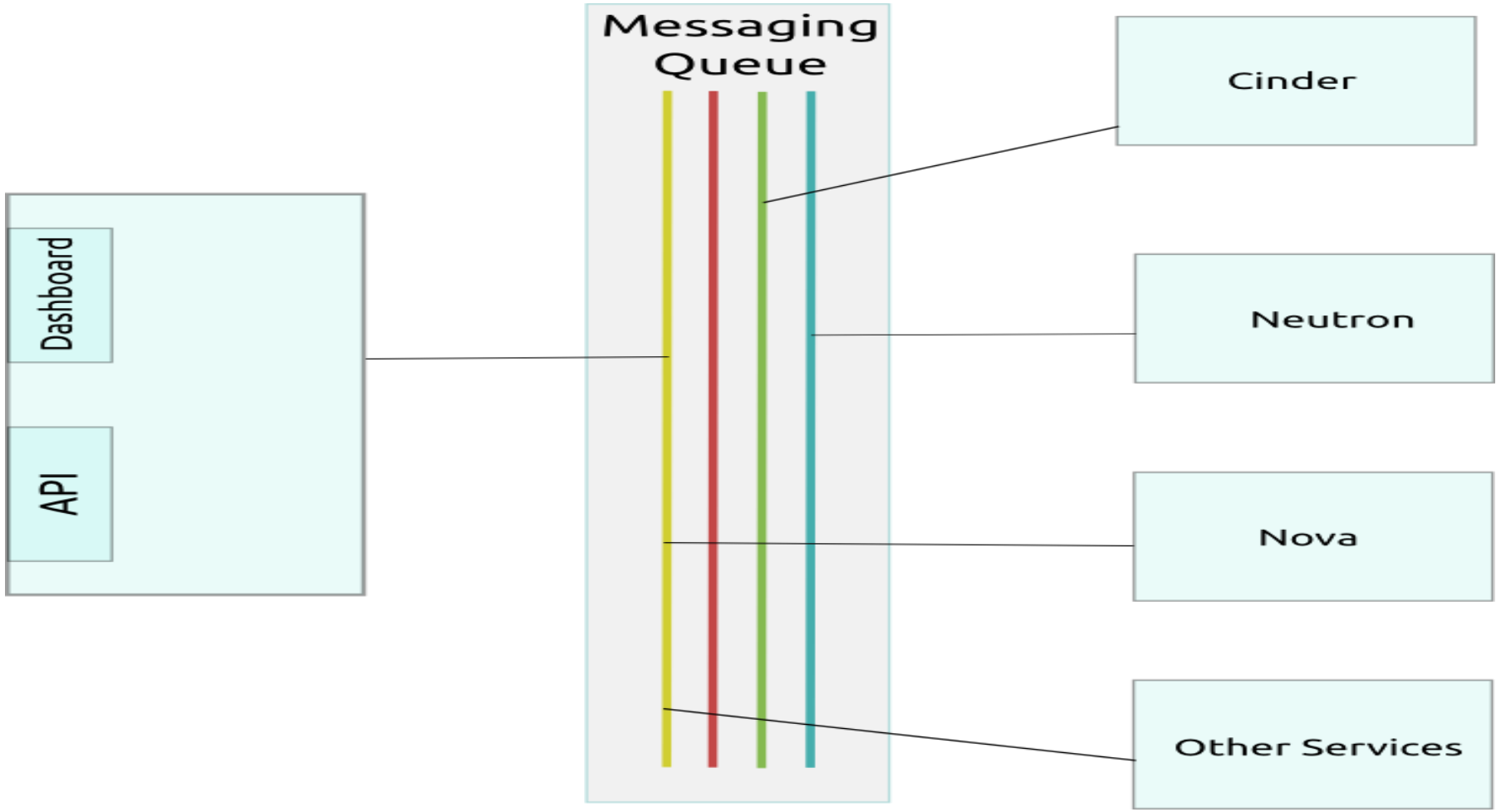
Service-oriented architecture (SOA)

A typical architecture contains an Orchestration/coordination layer, business service layer and infrastructure service layer or the service composition layer. There also exists a Service bus through which the communication takes place



INTRODUCTION TO SOFTWARE ENGINEERING

Example : Openstack

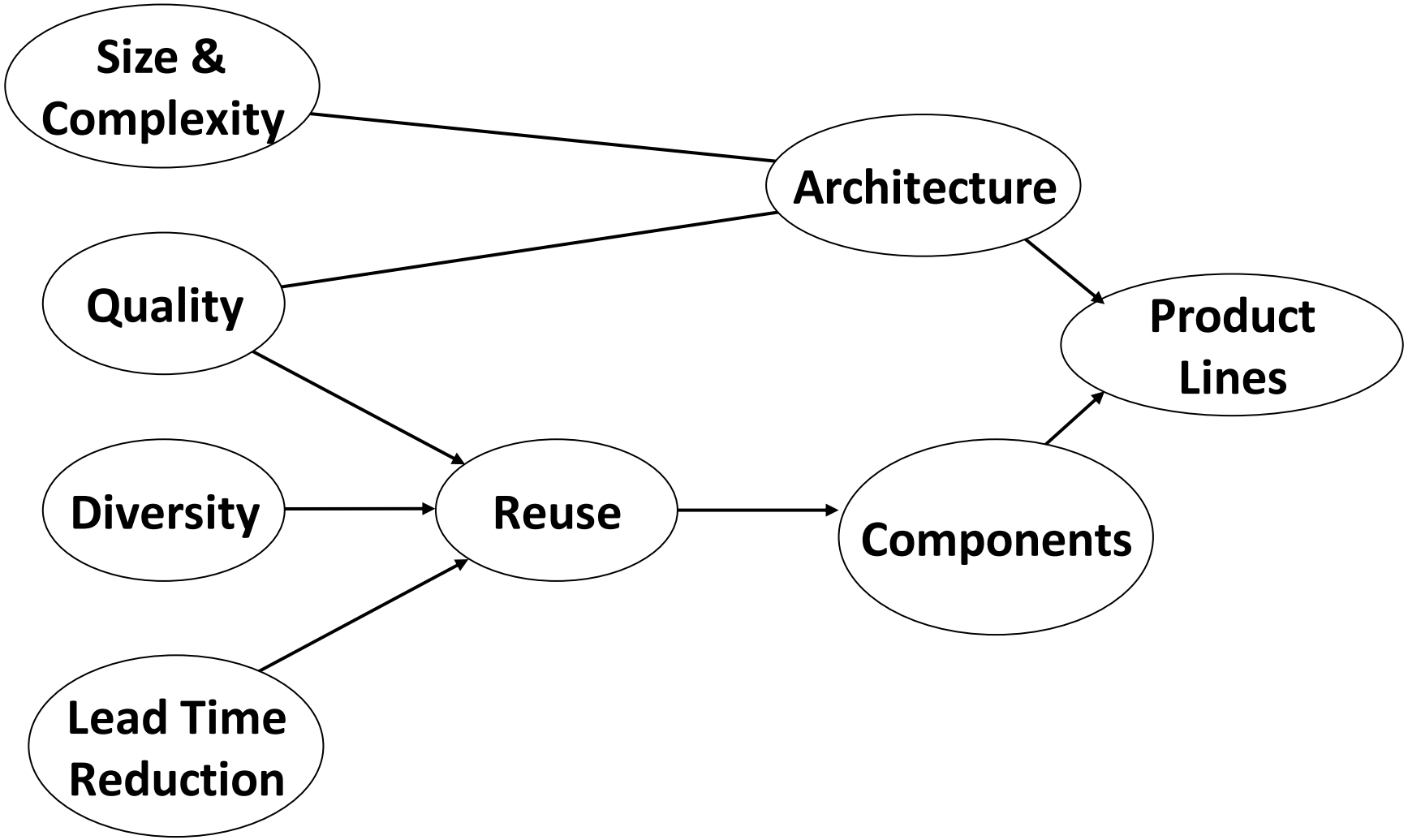


INTRODUCTION TO SOFTWARE ENGINEERING

Product Lines

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Motivation for Product Lines



INTRODUCTION TO SOFTWARE ENGINEERING

Software Product Line Engineering

So **Reuse** is an imperative

Software product lines refers to engineering techniques for creating a portfolio of similar software systems from a shared set of software assets

- A **product line** represents a family of manufactured products
- A **product line architecture** explicitly captures the **commonality** and **variability** of a product line components and their compositions

Software Product Line Engineering makes it possible to

- create software for different products
- use variability to customize the software to each different product

INTRODUCTION TO SOFTWARE ENGINEERING

Key Drivers for effective product lifecycle Re-Use :

- Software product lines enhance reuse through predictive software reuse (rather than opportunistic)
- Software artifacts are created when reuse is predicted in one or more products in a well defined product line

- These artifacts could be built as components which are reusable or could be looked at as design patterns which could be built using some fine grained components for a particular solution

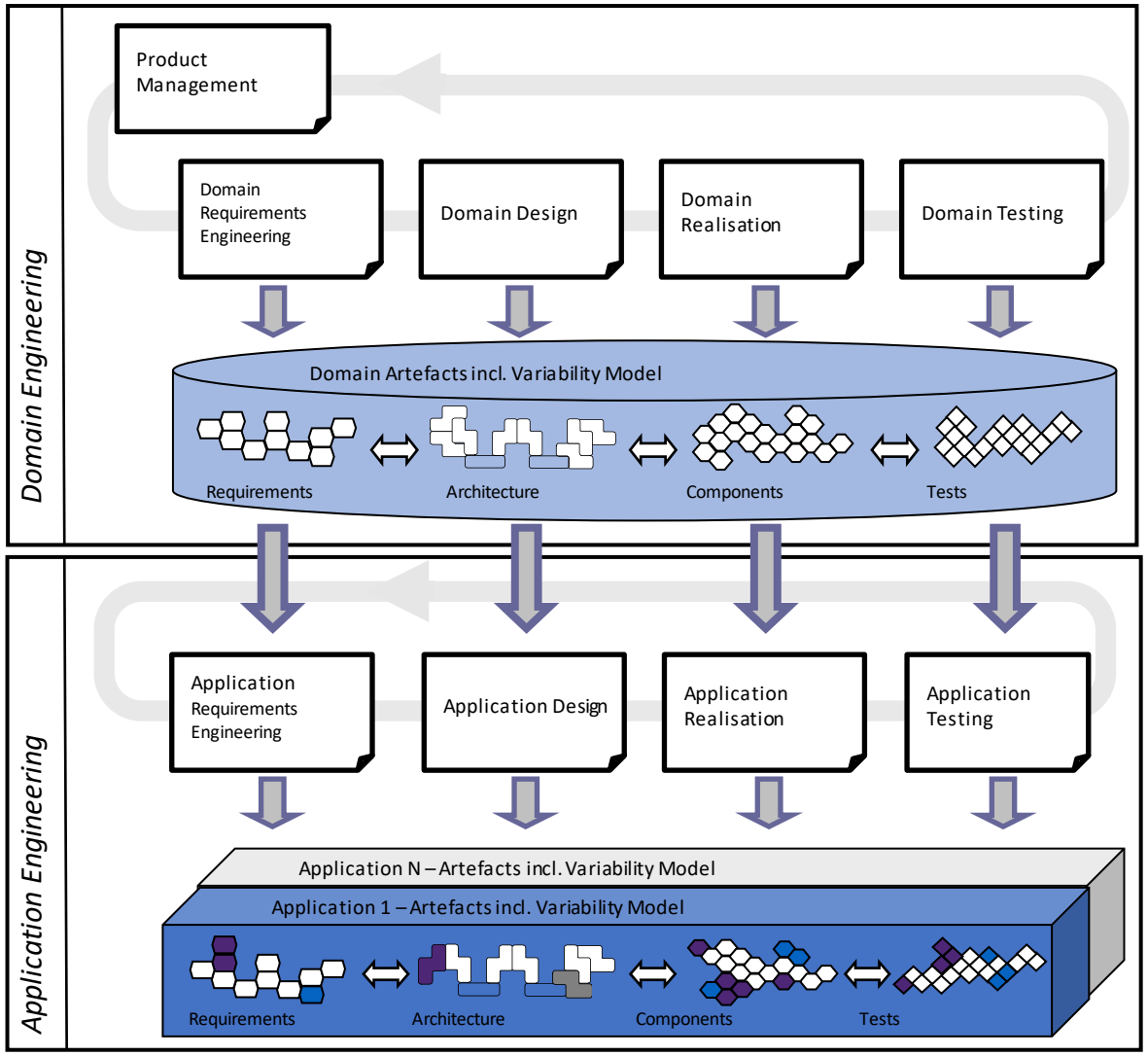
INTRODUCTION TO SOFTWARE ENGINEERING

Product Line Engineering Framework

Domain Engineering:
Define and realize the commonality and variability

The goal is to establish a reusable platform

Application Engineering:
Reuse domain artifacts, exploiting variability to build a product.
The goal is to derive a product from the platform established in the Domain Engineering phase



Based on the "Software Product Line Engineering" book by Klaus Pohl, Günter Böckle and Frank J. van der Linden

* All credits go to an IBM presentation on the Internet from where this has been leveraged

THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

REQUIREMENTS ENGINEERING

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

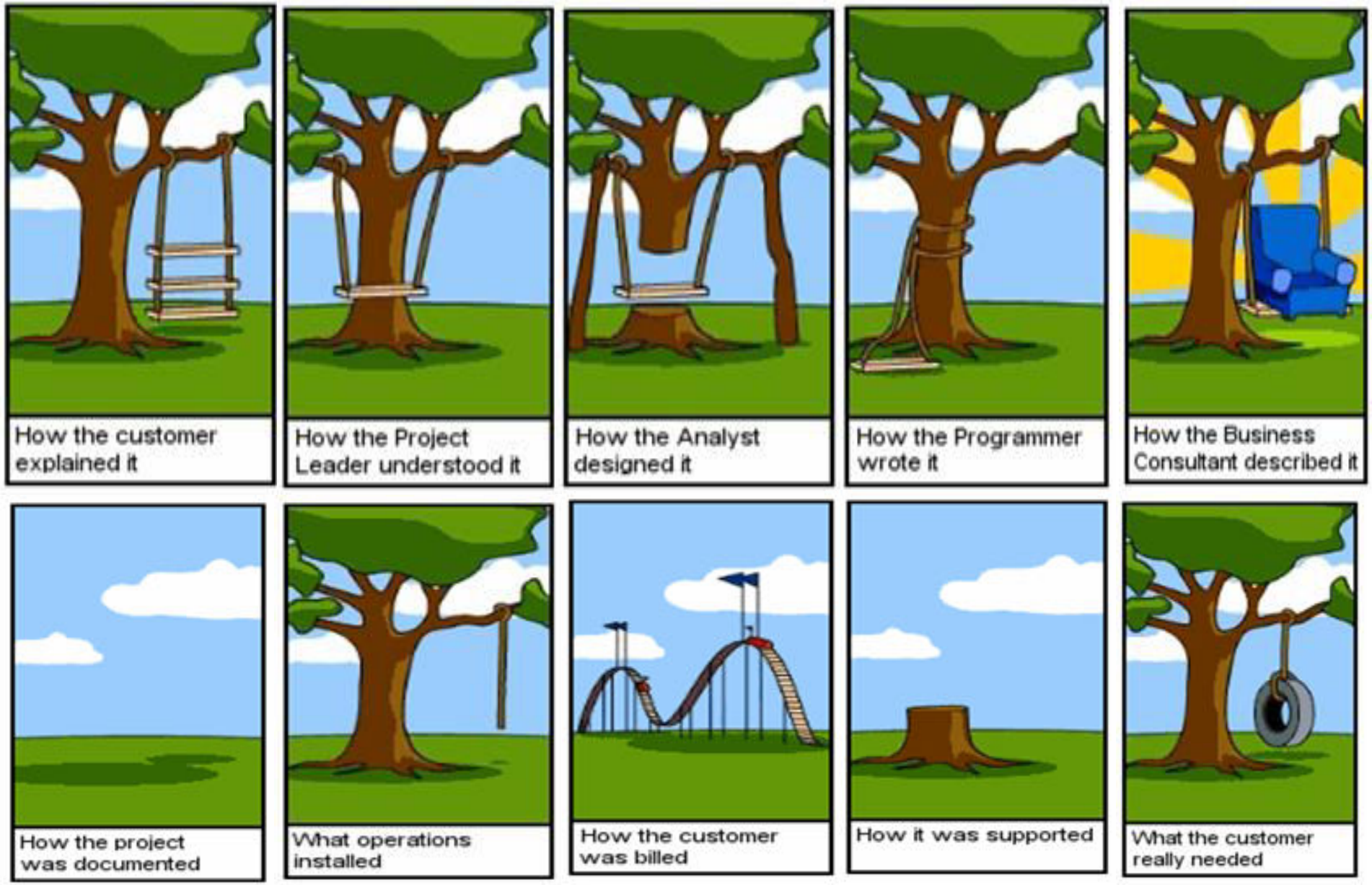
REQUIREMENTS ENGINEERING

Software Requirements & Requirement properties

Prof. Phalachandra H. L
Department of Computer Science and Engineering

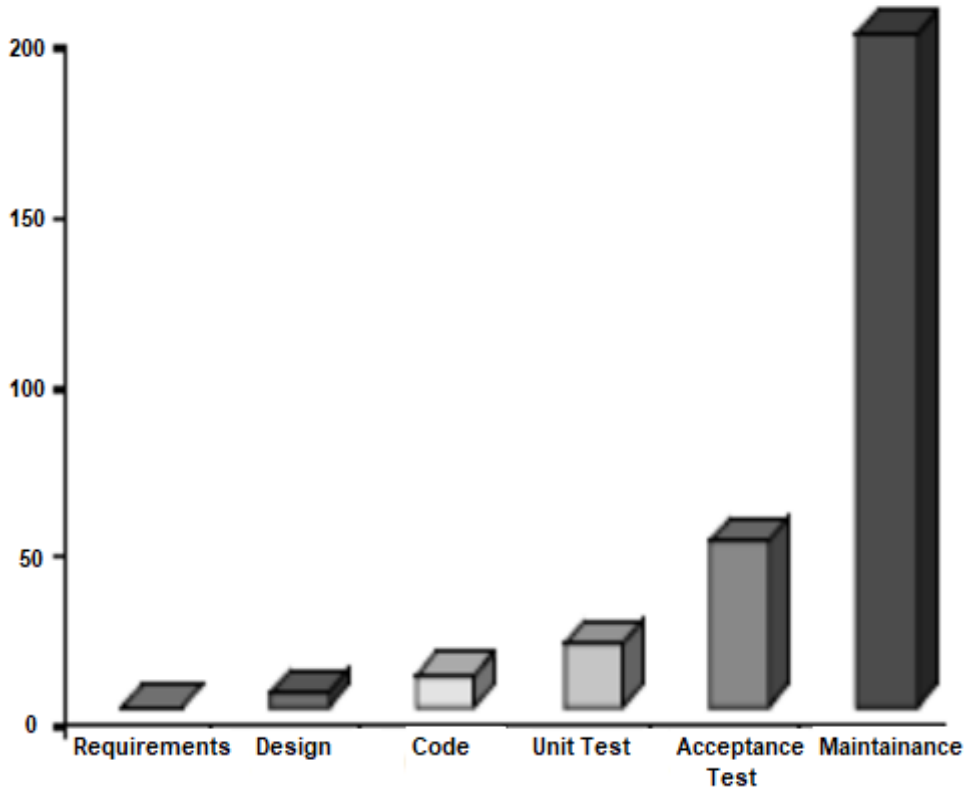
Generics

- Is usually the first step in any software intensive development lifecycle irrespective of model
  - Usually difficult, error prone and costly
  - Critical for successful development of all down stream activities
  - Errors introduced during requirements phase if not handled properly will propagate into the subsequent phases
  - Unnecessary, Late or invalid requirements can make the system cumbersome or slip
  - Requirement errors are expensive to fix


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

REQUIREMENTS ENGINEERING

Cost of repair as a function of time



Life Cycle Stage	Relative cost of Repair
Requirements	0.1 to 0.2
Design	0.5
Coding	1
Unit test	2
Acceptance test	5
Maintenance	20

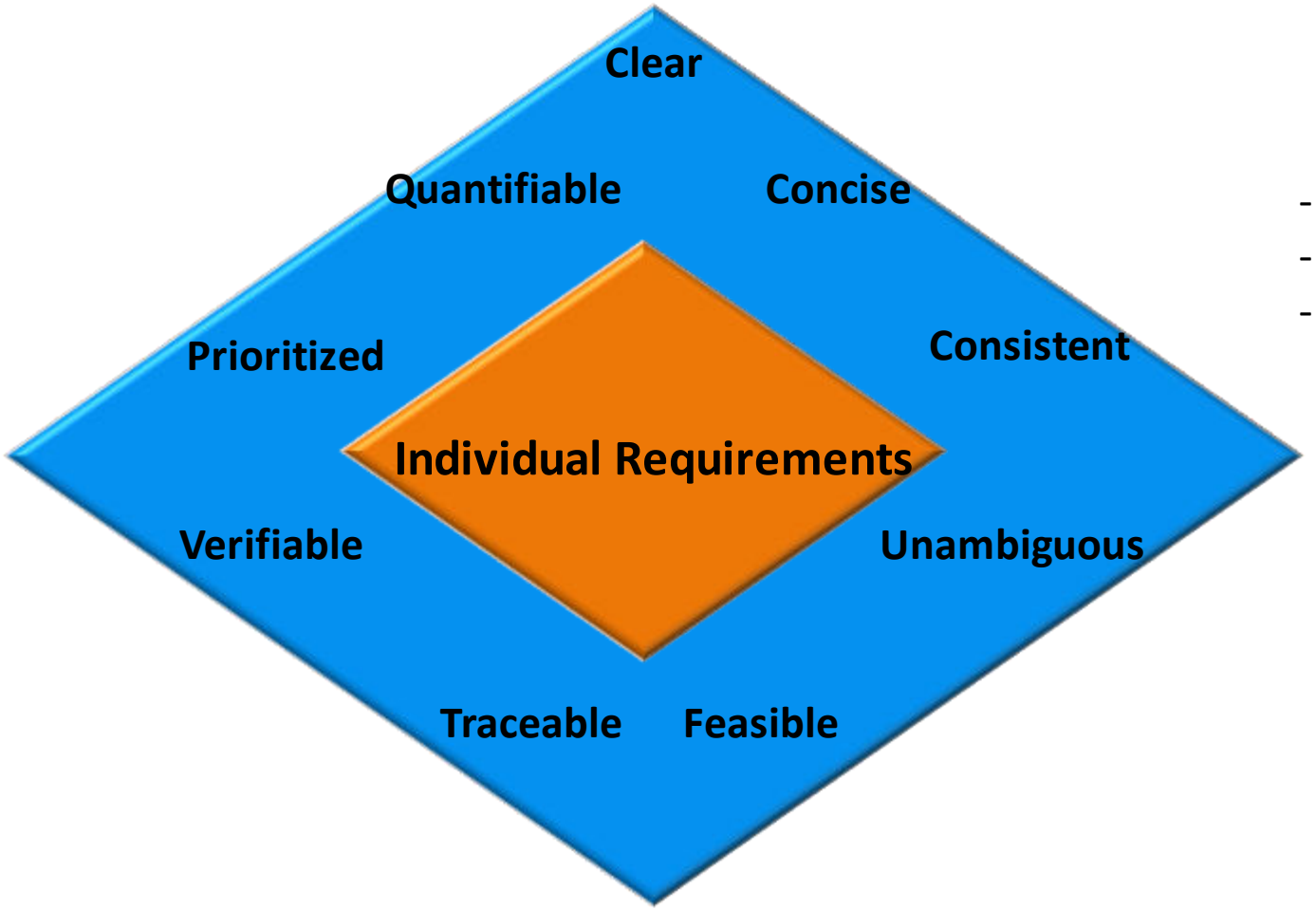
REQUIREMENTS ENGINEERING

Definition of Requirement

- Requirement is the Property which must be exhibited by software developed/adapted to solve a particular problem
- Requirement should specify the externally visible behavior of **what** and **not how**
- Requirements can be looked at as
  - Individual requirements
  - Set of requirements

REQUIREMENTS ENGINEERING

Properties of Requirement



Prioritized - requirements should be prioritized

Clear - Requirements should be written in precise, simple language that every reader can understand

Traceable - Linked backwards to stakeholder requirements and forwards to software design and code

Quantifiable - the requirement must have a clear, testable criterion and a cost-effective process to check it has been realized as requested

Concise - Requirements should be short and to the point

Consistent - Requirements should not be contradictory

Verifiable - Requirement must be verifiable by a cost-effective process

Unambiguous - Requirements should be stated in such a way that they can be interpreted in only one way

- Avoid combining Requirements

Examples

All screens must appear on the monitor quickly

or

When the user accesses any screen, it must appear on the monitor within 2 seconds (Clear – Concise - Unambiguous – Verifiable - Measurable)

The replacement control system shall be installed with no disruption to production

or

The replacement control system shall be installed causing no more than 2 days of production disruption

(Feasible)

Examples

The system must generate a batch end report and a discrepancy report when a batch is aborted

or

The system must generate a batch end report when a batch is completed or aborted

The system must generate a discrepancy report when a batch is aborted.

(Traceable)

The system must be user friendly

or

The user interface shall be menu driven. It shall provide dialog boxes, help screens radio buttons, dropdown list boxes, and spin buttons for user inputs

(Verifiable)

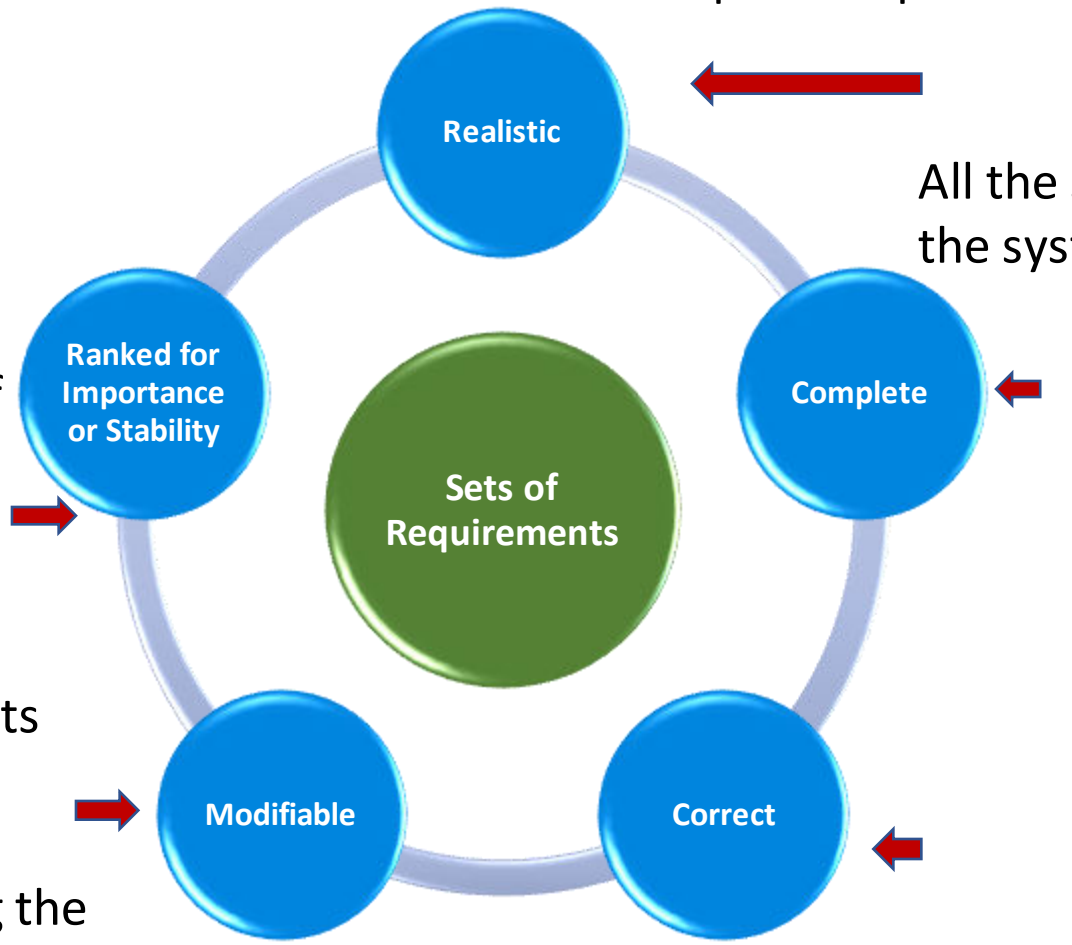
REQUIREMENTS ENGINEERING

Properties of a set of Requirement

When there are many requirements but limited time or budget, choices must be made about what to include or exclude. Factors such as changes in customer needs, improved developer understanding of the products, and changes in organizational policy will affect the stability of requirements.

Changes to the requirements should be able to be made easily, completely, and consistently while retaining the structure and style.

The goals that are represented by the requirements should be attainable within the time, resource, and cost constraints placed upon the project.



All the services that are needed from the system should be included

Every requirement should be one that the software will meet

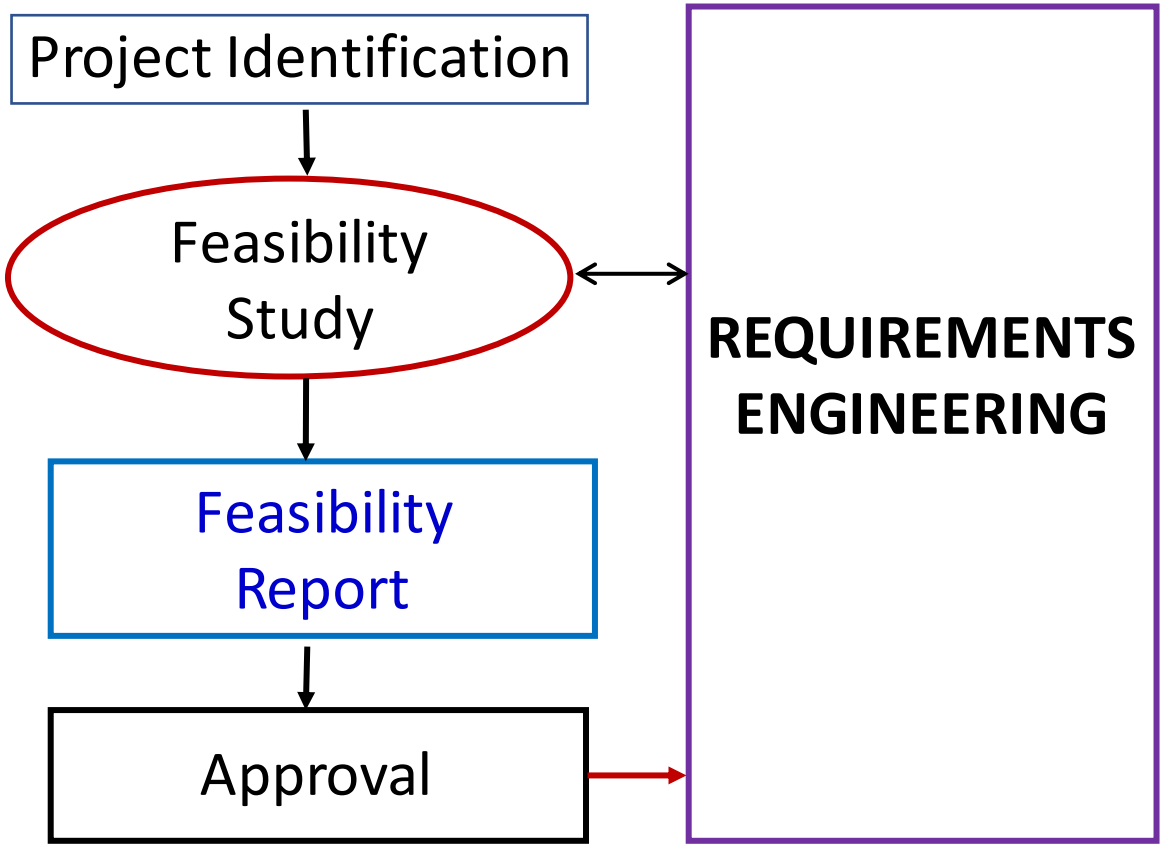
REQUIREMENTS ENGINEERING

Feasibility Study

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Feasibility Study

Mostly done before beginning a project, a short, low-cost study to identify.



REQUIREMENTS ENGINEERING

Activities in Feasibility Study

1. Client or the sponsor or the User who would have a stake in the project
2. What is the current solution to the problem
3. Who are the targeted customers and what is the future market place
4. Potential benefits : What would be gained by doing this project
5. Scope : A high level “what is included and what is not included” (This mostly includes High-level abstract description of the requirements)

REQUIREMENTS ENGINEERING

Activities in Feasibility Study

6. High level block Level understanding of the solution
7. Considerations to technology
8. Marketing strategy
9. Financial projection
10. Schedule and high level planning and budget requirements
11. Issues, Assumptions, Risks and Constraints
12. Alternatives and their consideration
13. Potential project organization

Will end with a go or No-go Decisions

REQUIREMENTS ENGINEERING

Requirements Engineering Process

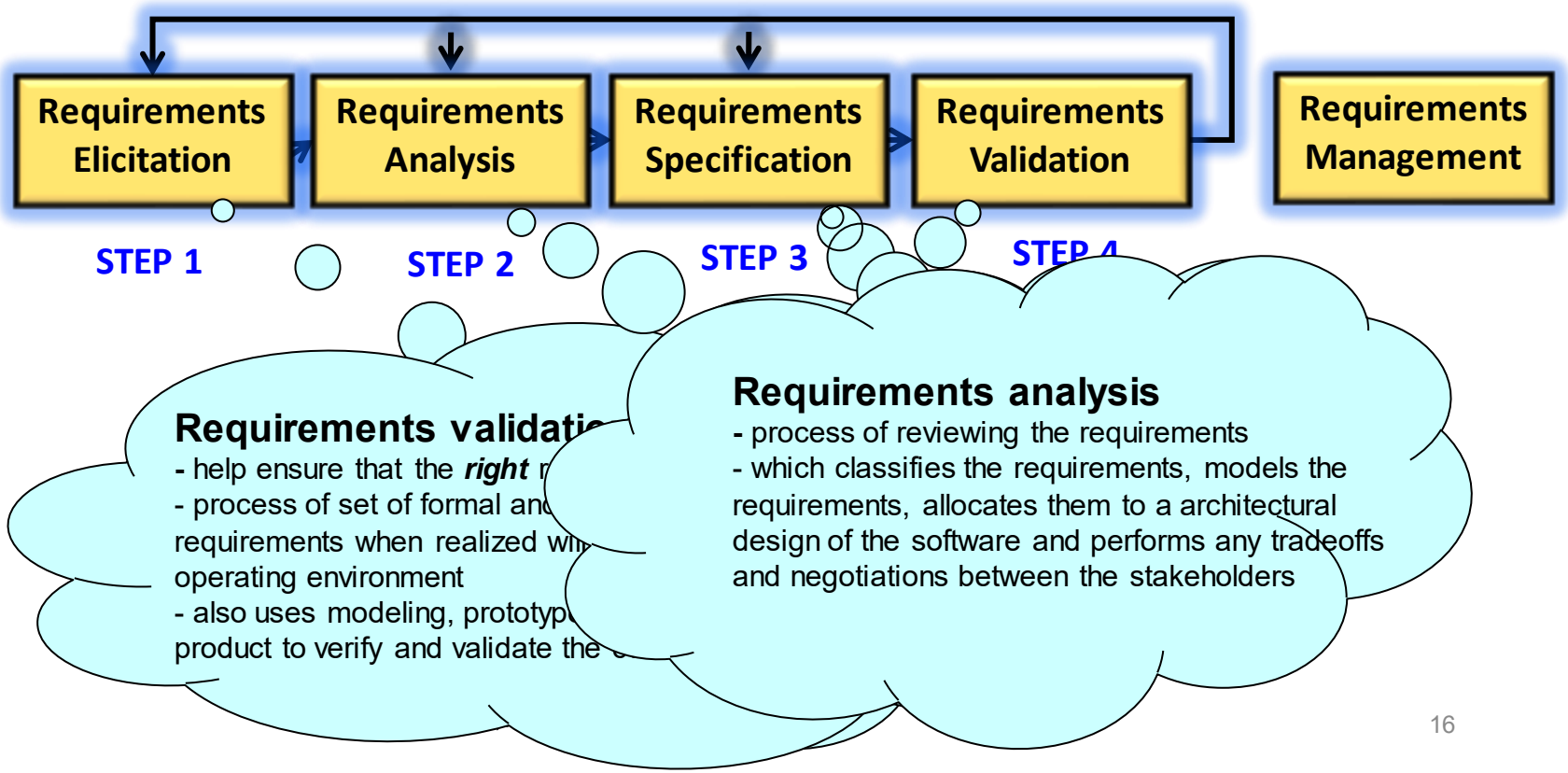
Prof. Phalachandra H. L
Department of Computer Science and Engineering

REQUIREMENTS ENGINEERING

Requirements Engineering process

There are “four + one” sets of activities that have been used, and have shown to produce specifications or requirements

Iterative Process



THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

REQUIREMENTS ENGINEERING

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

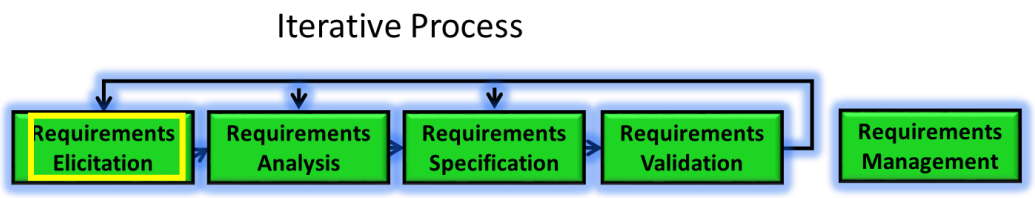
REQUIREMENTS ENGINEERING

Requirements Elicitation

Prof. Phalachandra H. L
Department of Computer Science and Engineering

REQUIREMENTS ENGINEERING

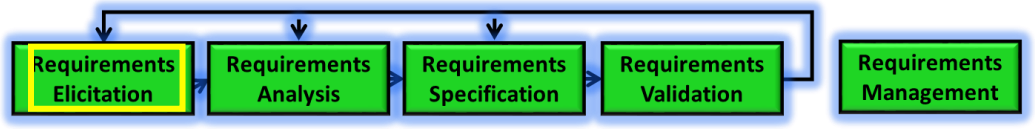
Requirements Elicitation process



- Is a process of working proactively with all stakeholders gathering their needs, articulating their problem, identify and negotiate potential conflicts thereby **establishing a clear scope and boundary for a project**
- It can also be described as a process of ensuring that the stakeholders have been identified and they have been given an **opportunity to explain their problem and needs** and describe what they would like the new system to do

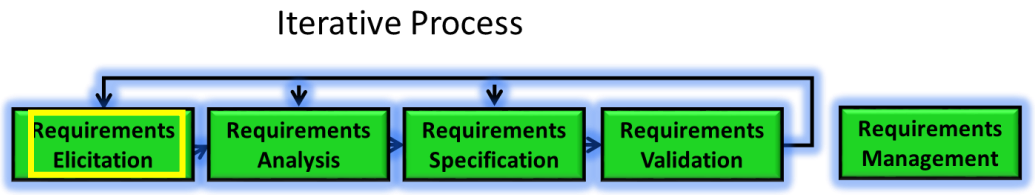
REQUIREMENTS ENGINEERING

Requirements Elicitation process



Involves

- Understanding the problem
- Understanding the domain
- Identifying clear objectives
- Understanding the needs
- Understanding constraints of the system stake holders
- Writing business objectives for the project



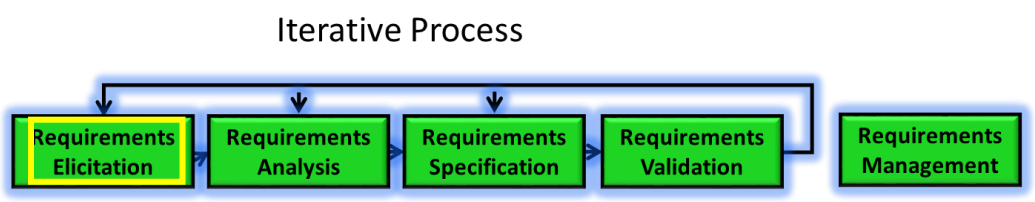
Elicitation Techniques

The approach depends on

- Nature of the system being developed
  - For example a UI intensive system needs a different approach compared to an embedded system as the navigation and behaviour is visible, so easy to comprehend
- Background and experience of stakeholders
  - If the stakeholder is a very hands-on person then prototyping and simulation would work
  - For non-computer savvy persons documents would be necessary

REQUIREMENTS ENGINEERING

Requirements Elicitation process



Elicitation Techniques

Active - where there are ongoing interaction between the stake holders and users. Some of the techniques are

- interviews
- facilitated meetings
- role-playing
- Prototypes
- Ethnography (study/collect the system in its normal working surroundings)
- Scenarios

Passive - infrequent interaction between the stake holders and users. E.g. use of

- use cases
- business process analysis and modelling
- workflows
- questionnaires
- checklists
- documentation

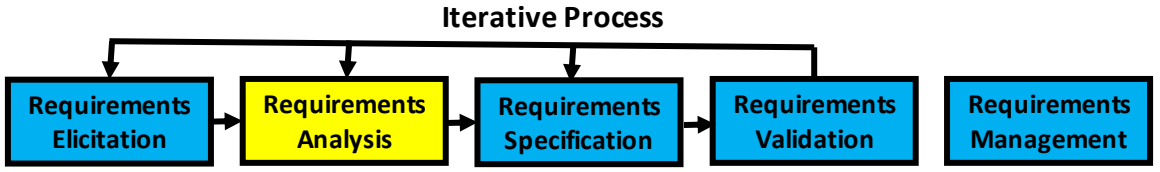
REQUIREMENTS ENGINEERING

Requirements Analysis

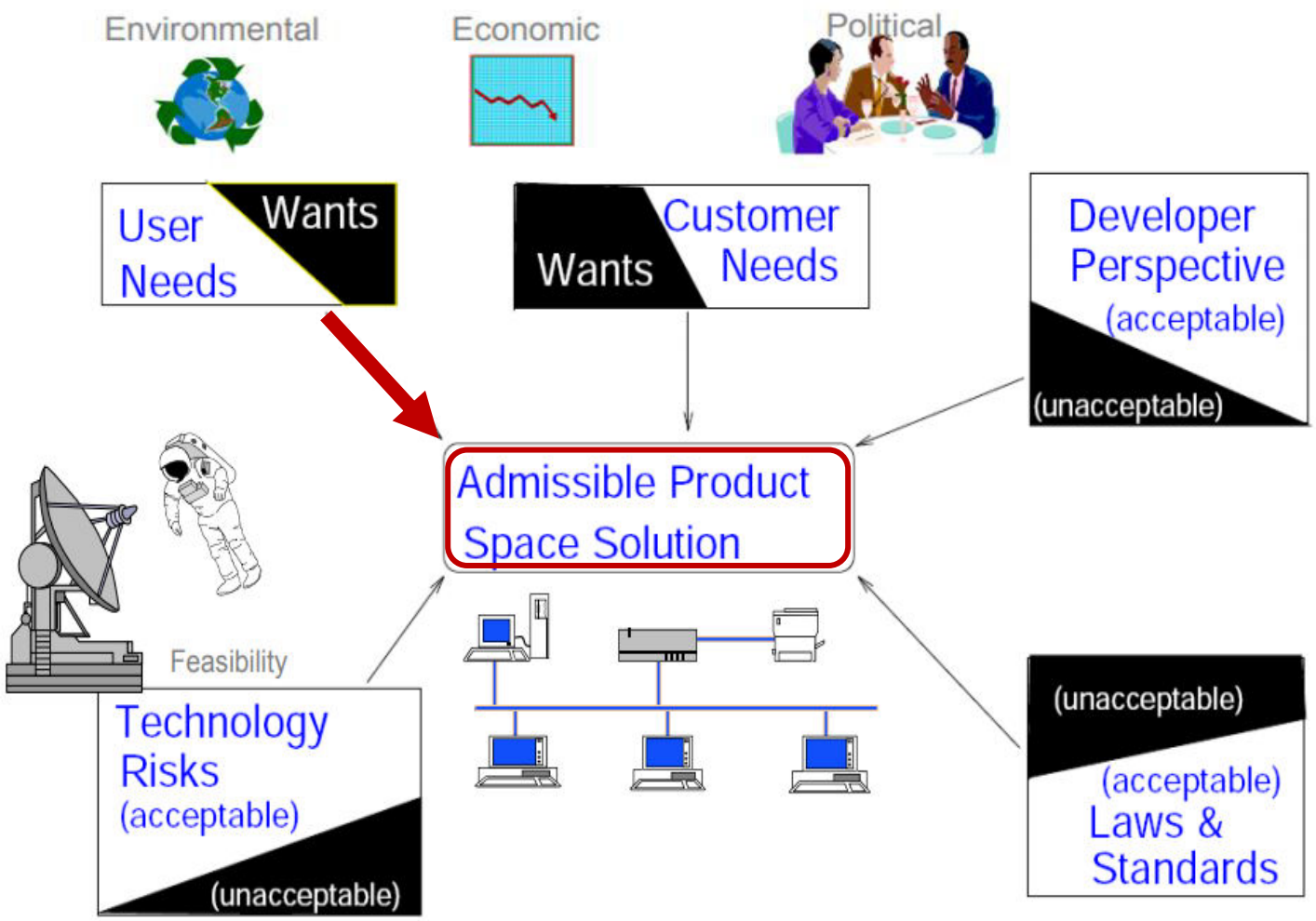
Prof. Phalachandra H. L
Department of Computer Science and Engineering

REQUIREMENTS ENGINEERING

Requirements Analysis process

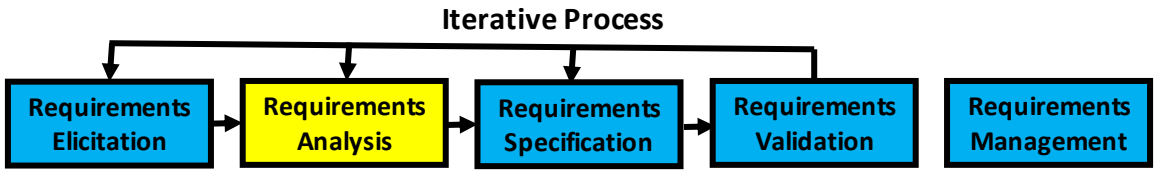


Requirements represent a compromise.



REQUIREMENTS ENGINEERING

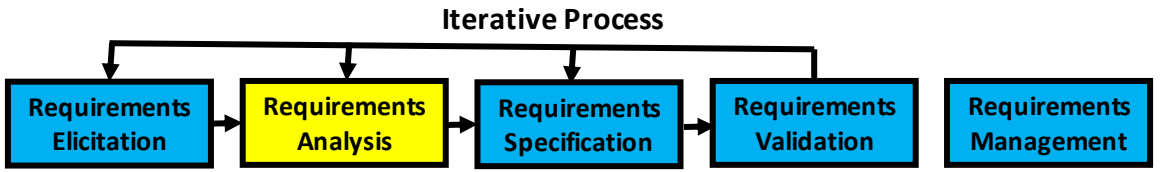
Requirements Analysis process



1. Understand the requirements in depth, both from a product and process perspective
2. Classify and Organize the requirements into coherent clusters
   - Functional, Non-Functional & Domain requirements
   - System and User Requirements
3. Model the requirements
4. Analyze the requirements (if necessary) using fish bone diagram

REQUIREMENTS ENGINEERING

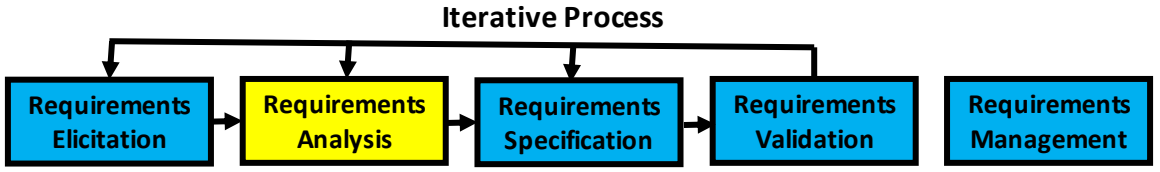
Requirements Analysis process



5. Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)
6. Negotiate Requirements
7. Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)
8. Identify risks if any
9. Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements

REQUIREMENTS ENGINEERING

Requirements Analysis process



1. **Understand the requirements in depth, both from a product and process perspective**

Requirement or Problem needs to be correctly internalized

A Storm coming in

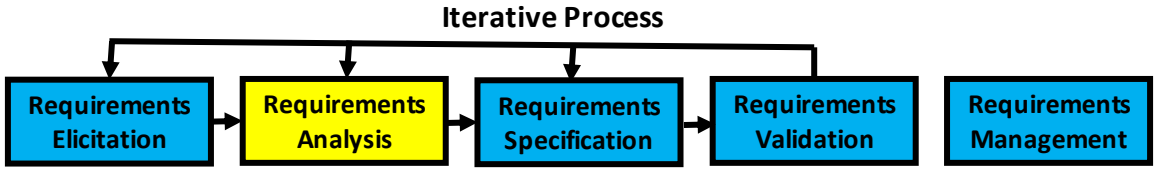
Could be a problem to a game

Could be a blessing to a farmer

1. **Understand the requirements in depth, both from a product and process perspective**
2. **Classify and Organize the requirements into coherent clusters**
3. **Model the requirements**
4. **Analyze the requirements (if necessary) using fish bone diagram**
5. **Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)**
6. **Negotiate Requirements**
7. **Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)**
8. **Identify risks if any**
9. **Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements**

REQUIREMENTS ENGINEERING

Requirements Analysis process



PES UNIVERSITY ONLINE

2. Classify and Organize the requirements into coherent clusters

Functional, Non-Functional & Domain requirements

Functional Requirements

Functionality or services, the system should provide with different inputs, and expression on how the system should behave in particular situations

- Could be written as high-level statements
- Can be verified
- This also indicates states what the system should not do

Example :

System shall assign a unique tracking number to each shipment

Non-Functional Requirements

Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards etc.

- Often applied to the system as a whole rather than individual features or services.
- Specify the criteria that can be used to judge the operation of the system

Example:

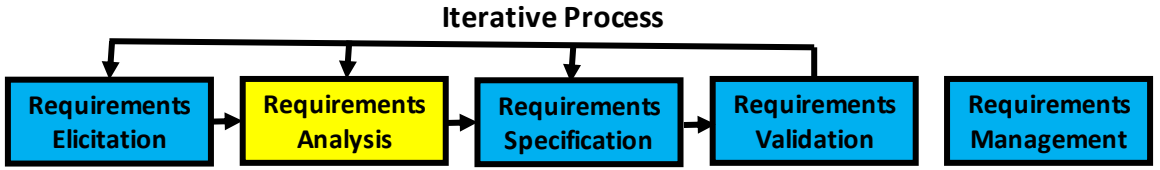
With 100 concurrent users a database record shall be fetched over the network in less than 3ms

Domain requirements Constraints on the system from the domain of operation

1. Understand the requirements in depth, both from a product and process perspective
2. Classify and Organize the requirements into coherent clusters
3. Model the requirements
4. Analyze the requirements (if necessary) using fish bone diagram
5. Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)
6. Negotiate Requirements
7. Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)
8. Identify risks if any
9. Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements

REQUIREMENTS ENGINEERING

Requirements Analysis process



PES UNIVERSITY ONLINE

2. Classify and Organize the requirements into coherent clusters (Cont.)

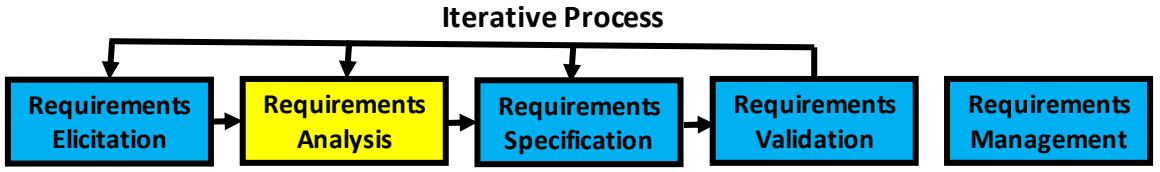
System and User requirements

User Requirements	System Requirements
Statements in natural language plus informal context diagrams system/sub-system and their interconnections and operational constraints. Written for/by customers.	A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor. Also called Software requirements / Functional specifications

1. Understand the requirements in depth, both from a product and process perspective
2. Classify and Organize the requirements into coherent clusters
3. Model the requirements
4. Analyze the requirements (if necessary) using fish bone diagram
5. Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)
6. Negotiate Requirements
7. Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)
8. Identify risks if any
9. Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements

REQUIREMENTS ENGINEERING

Requirements Analysis process



3. Model the requirements

A Model is a representation of a system in some form. A is a Model of B if A can be used to answer questions about B

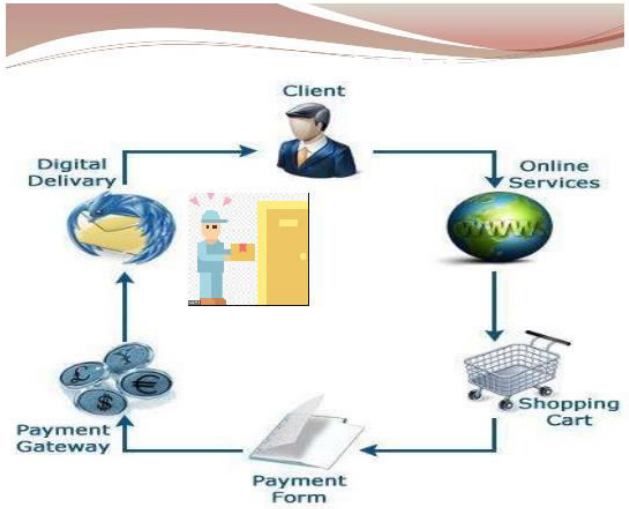
How would you represent a

Briefcase

Model of a Briefcase

Briefcase
- Capacity - Weight
+ Open() + Close()

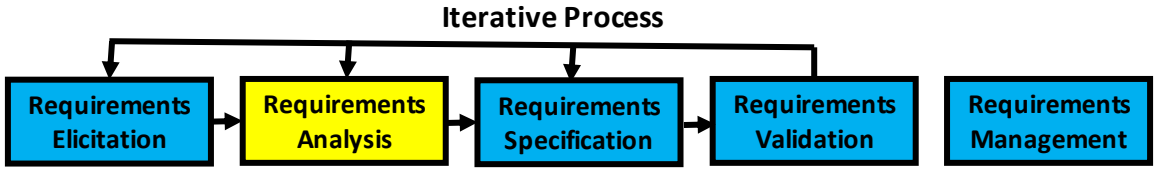
Online Shopping



Other Examples of Models

- Building Plan

1. Understand the requirements in depth, both from a product and process perspective
2. Classify and Organize the requirements into coherent clusters
3. Model the requirements
4. Analyze the requirements (if necessary) using fish bone diagram
5. Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)
6. Negotiate Requirements
7. Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)
8. Identify risks if any
9. Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements



3. **Modelling Systems (Cont.)**

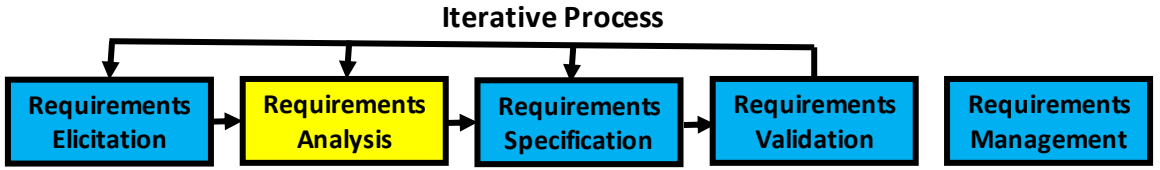
Goals for the models (or why models)

- Understanding (existing) System
  - Analyzing and Validating the requirements in terms of visible requirements within the problem
  - Make modifications
  - Fix, Upgrade, Integrate
- Communicating the requirements in terms of who, what and interpreting it in the same way
- Should typically provide value to all stakeholders
- Should help predict system behavior
  - "What if" analysis (financial models)
  - Test system
    - Performing tests when doing so on real systems are risky and expensive

      Example: Test limits or breaking point
- Should be Simple

REQUIREMENTS ENGINEERING

Requirements Analysis process



3. Modelling Systems (Cont.)

Structure Models

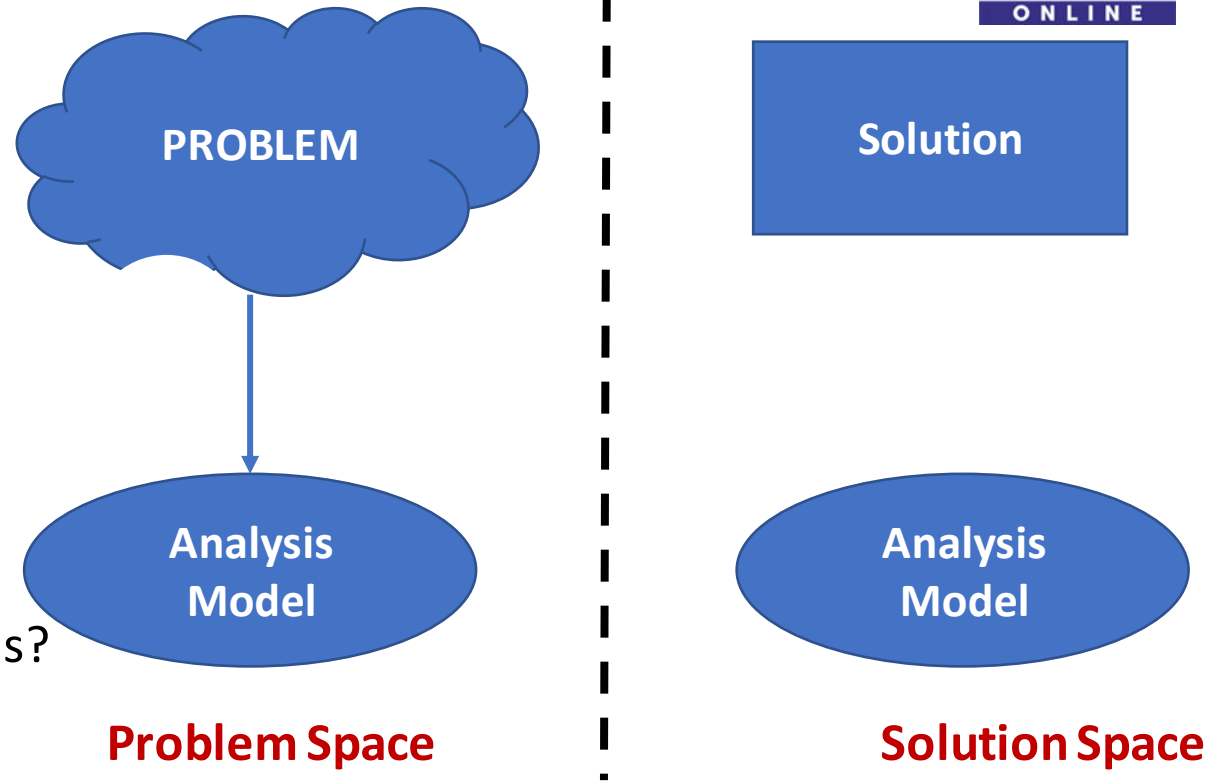
- Captures static aspects of system
- What entities exist in the system?
- How are they related?

  Example: Organization Chart

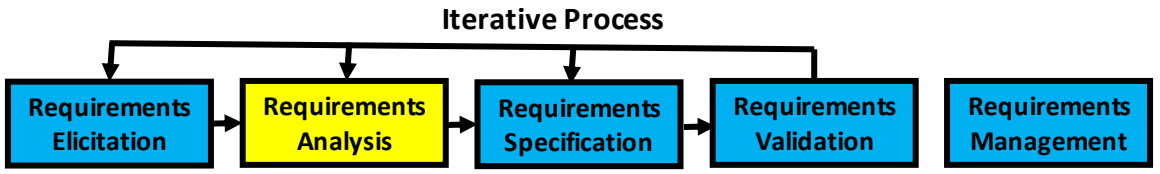
Behaviour Models

- Captures dynamic aspects of the system
- How do the entities interact in response to a stimulus?

  Example: Process

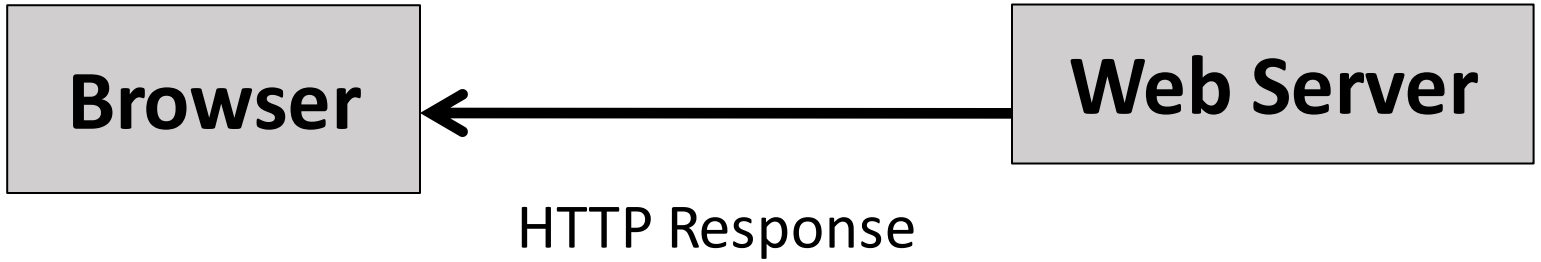


Models could **Informal** e.g. Prose or could **Simple Procedural** e.g. Flowcharts or Pseudo Code or could be **formal static models** like ER diagrams **or dynamic models.**

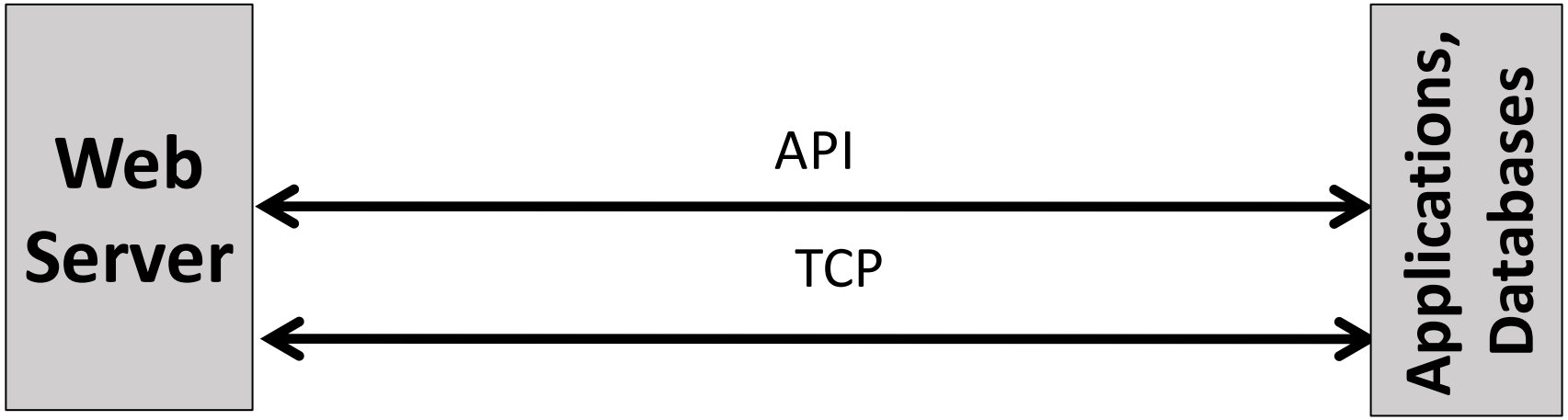


3. Modelling Systems (Cont.)

Some Simple Models



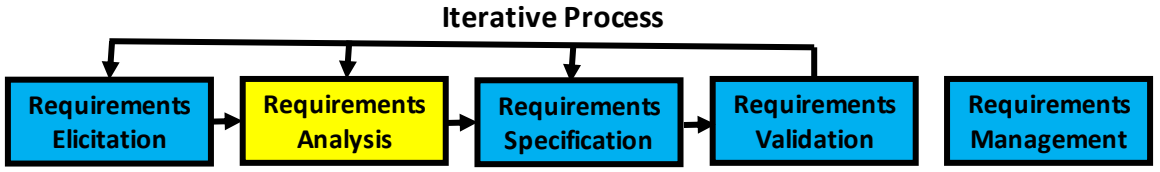
HTTP is an Application Layer Protocol (Defined in RFC 2616 of IETF)
Specifies *what* messages are exchanged between Client and Server (but not *how* they are transported)



They could be UML based models like Use Case models (we will discuss this in more detail next class)

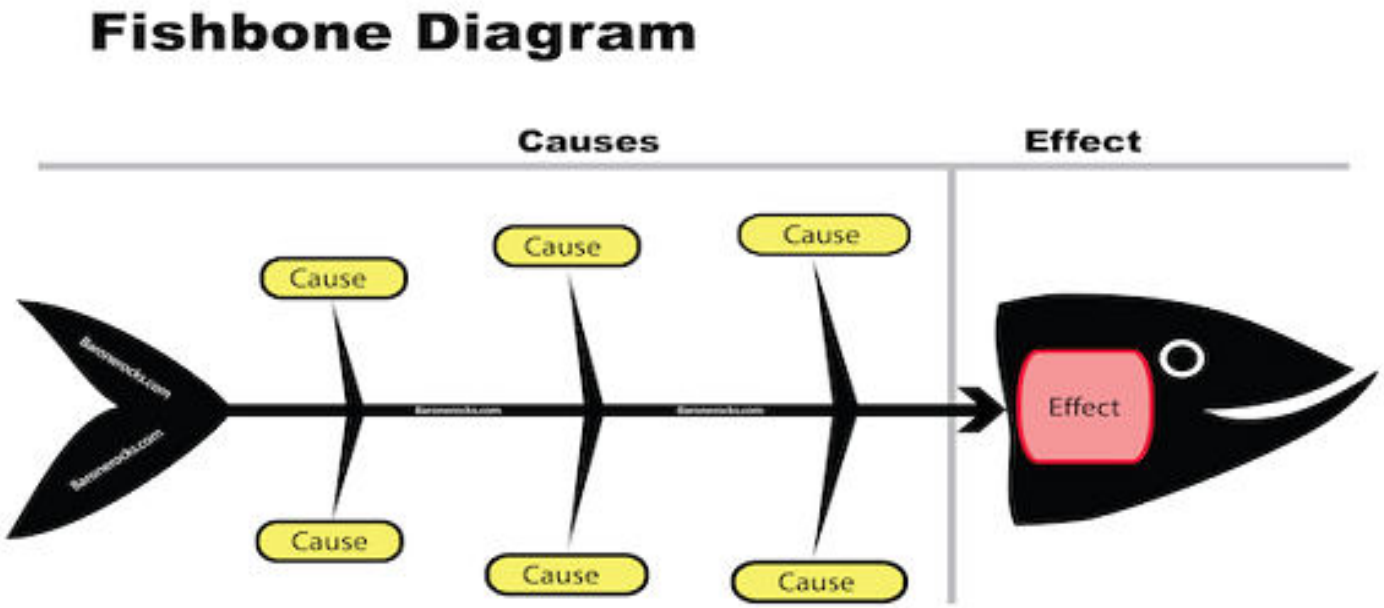
REQUIREMENTS ENGINEERING

Requirements Analysis process



4. **Analyze the requirements using fish bone diagram**

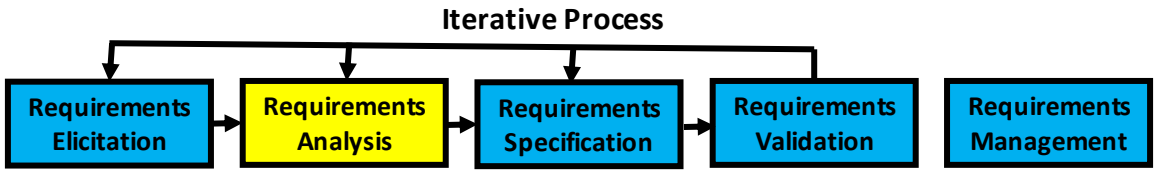
List out all the reasons/causes on why the requirement (effect) has come in and become relevant



1. **Understand the requirements in depth, both from a product and process perspective**
2. **Classify and Organize the requirements into coherent clusters**
3. **Model the requirements**
4. **Analyze the requirements (if necessary) using fish bone diagram**
5. **Recognize and resolve conflicts (e.g., functionality v. cost v. timeliness)**
6. **Negotiate Requirements**
7. **Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Wont have)**
8. **Identify risks if any**
9. **Decide on Build or Buy (Commercial Of The Shelf Solution) and refine requirements**

REQUIREMENTS ENGINEERING

Requirements Analysis process



5. Recognize and Resolve Conflicts
   - Functionality Vs Cost Vs timelines
6. Negotiate requirements
7. Prioritize the requirements (MoSCoW -Must have, Should have, Could have, Won't have)
   - Pareto Analysis (80-20 to focus on vital few to trivial many)
8. Identify risks if any
9. Decide on Build or Buy (Commercial Of The Shelf Solution- COTS) and refine requirements

THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

REQUIREMENTS ENGINEERING

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

REQUIREMENTS ENGINEERING

Modelling – UML and Use Case Models

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Modelling Recap

A Model is a representation of a system in some form. A is a Model of B if A can be used to answer questions about B

Couple of important goals of Modelling

- Providing an Understanding (existing) System
  - Analyzing and Validating the requirements in terms of visible requirements within the problem
- Communicating the requirements in terms of who, what and interpreting it in the same way

Discussed different kinds of Models

- Structural Models and Behavioral models

What is UML

- **Unified Modeling Language**
  - Language to express different types of models
  - Language defines
  - Syntax – Symbols and rules for using them
  - Semantics – What these symbols represent
- **UML can be used to**
  - Visualize (Graphical Notation)
  - Specify (Complete and Unambiguous)
  - Construct (Code Generation)
  - Document (Design, Architecture, etc.)

  the artifacts of software-intensive systems

REQUIREMENTS ENGINEERING

Conceptual model of UML

Things are abstractions in the model

Relationship ties together the things or abstractions

There are four different kinds of relationships

Diagrams : Graphical representation of set of elements, often rendered as a set of connected graph of vertices (things) and arcs (relationships)

It's a projection into the system

Use case is a description of set of sequences of actions that a system performs that yields an observable result of value to a particular actor in other words a functional requirement

Its used to structure the behavior. Its realized using collaboration. Depicted as an ellipse with solid lines

al elements. They are
he UML
ilding blocks can be put together
y through out UML

Active class
Component
Node
Use case
Collaboration
State machine
Realiza
Deployment

Use Case Diagram

Shows actors, use cases, and relationships

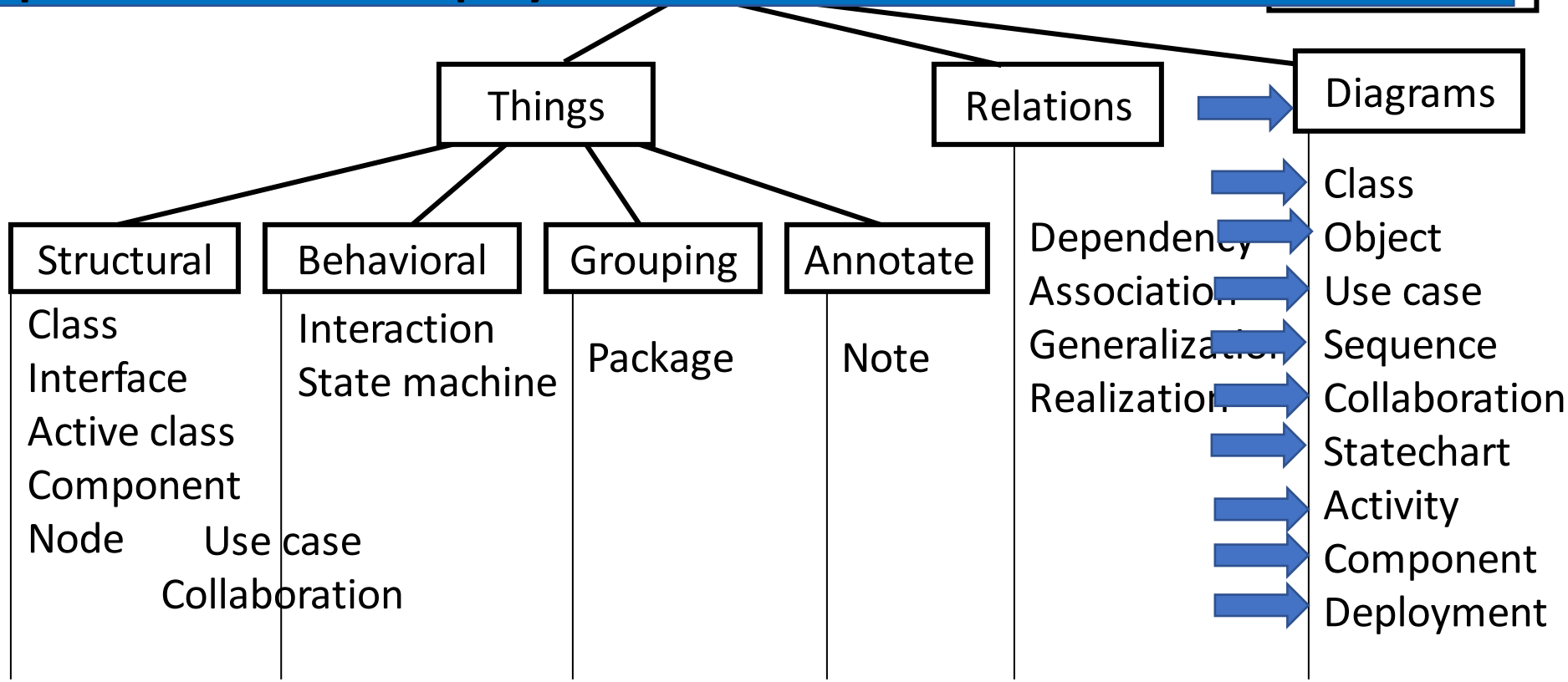
Conceptual model of UML

Diagrams

Component Diagram

Deployment Diagram

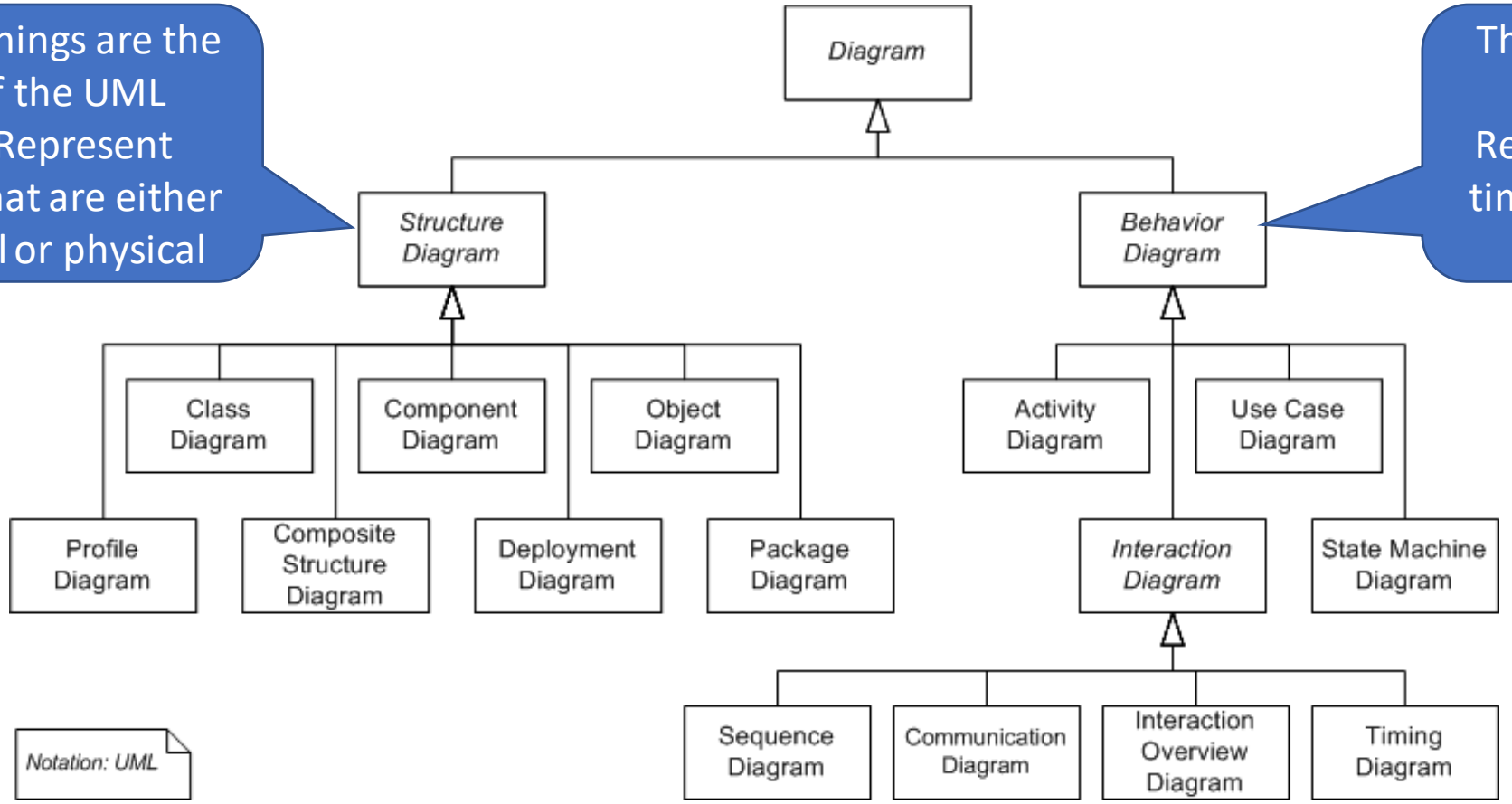
Shows the configuration of run-time processing nodes and the components that are deployed on them



UML Diagrams

Structural things are the nouns of the UML models. Represent elements that are either conceptual or physical

These are dynamic parts of the UML models. Represent behavior over time and space. Typically verbs of the model



UML Models and associated diagram types

Static or Structural diagrams:

What must be in the system

- Class
- Component
- Deployment
- Composite structure
- Object
- Package

Dynamic or Behavioral diagrams:

What must happen in the system

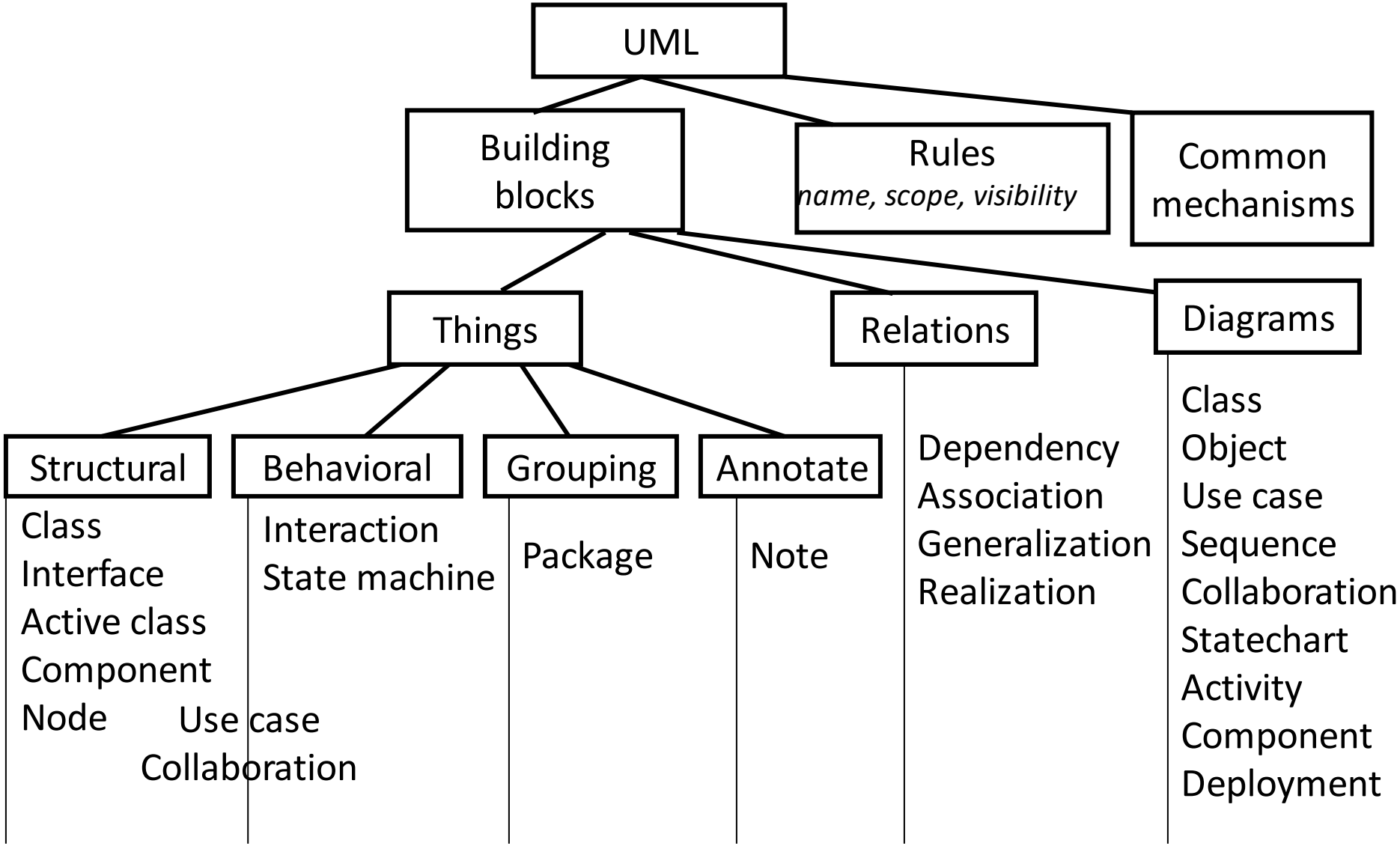
- Activity
- State machine
- Use case

Interaction diagram (kind of Dynamic diagram):

[emphasize the flow of control and data among the things in the system]

- Communication
- Sequence
- Timing

Conceptual model of UML (repeat .. for readability purpose)



UML Diagrams (repeat contents for the Animated Slide)

- Use Case Diagram
  - Shows actors, use cases, and relationships
- Class Diagram
  - Shows classes, interfaces, relationships
  - Most common – address static view
- Object Diagram
  - Shows objects (instances)
- Component Diagram
  - Shows deployable components, including interfaces, ports, and internal structure
- Activity Diagram
  - Shows processes, including flow of control and data

UML Diagrams… (continued) repeated contents for the Animated Slide

- State Diagram
  - Shows states, transitions, events, and activities depicting the dynamic view of internal object states
- Sequence Diagram
  - Shows interactions between objects as a time-ordered view
- Collaboration Diagram
  - Similar to sequence diagram, but a spatial view, using numbering to indicate time order
- Deployment Diagram
  - Shows the configuration of run-time processing nodes and the components that are deployed on them

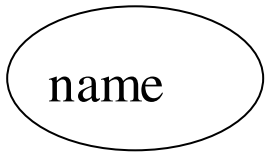
REQUIREMENTS ENGINEERING

Using Use Case Diagrams

- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- Use case diagrams contain use cases, actors, and their relationships.

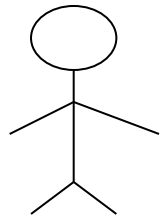
Use Case

- Use cases specify the desired behavior.
- A use case is
  - a description of a set of sequences of actions a system performs to yield an observable result of value to an actor.
  - includes variants
- Named as verb.
- Each sequence represent an interaction of actors with the system.



Actors

- An actor represents a set of roles that users of use case play when interacting with these use cases.
- Actors can be human or automated systems.
- Actor is someone interacting with use case (system function). Named by noun.
- Actors are entities which require help from the system to perform their task or are needed to execute the system's functions.
- Actors are not part of the system.
- An Actor triggers use case
- Actor has responsibility toward the system (inputs), and have expectations from the system (outputs).

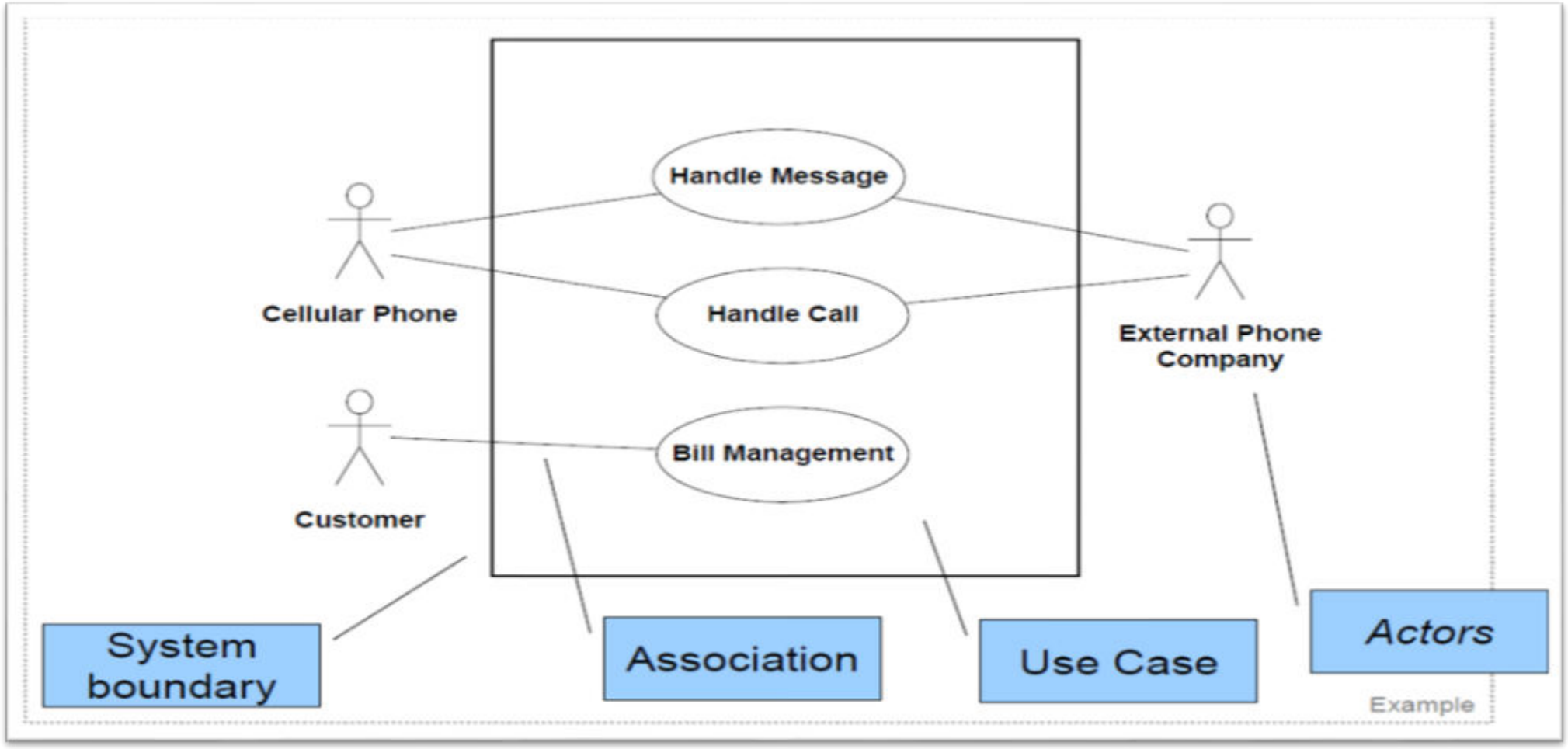


name

<< actor >>
X Authorization System

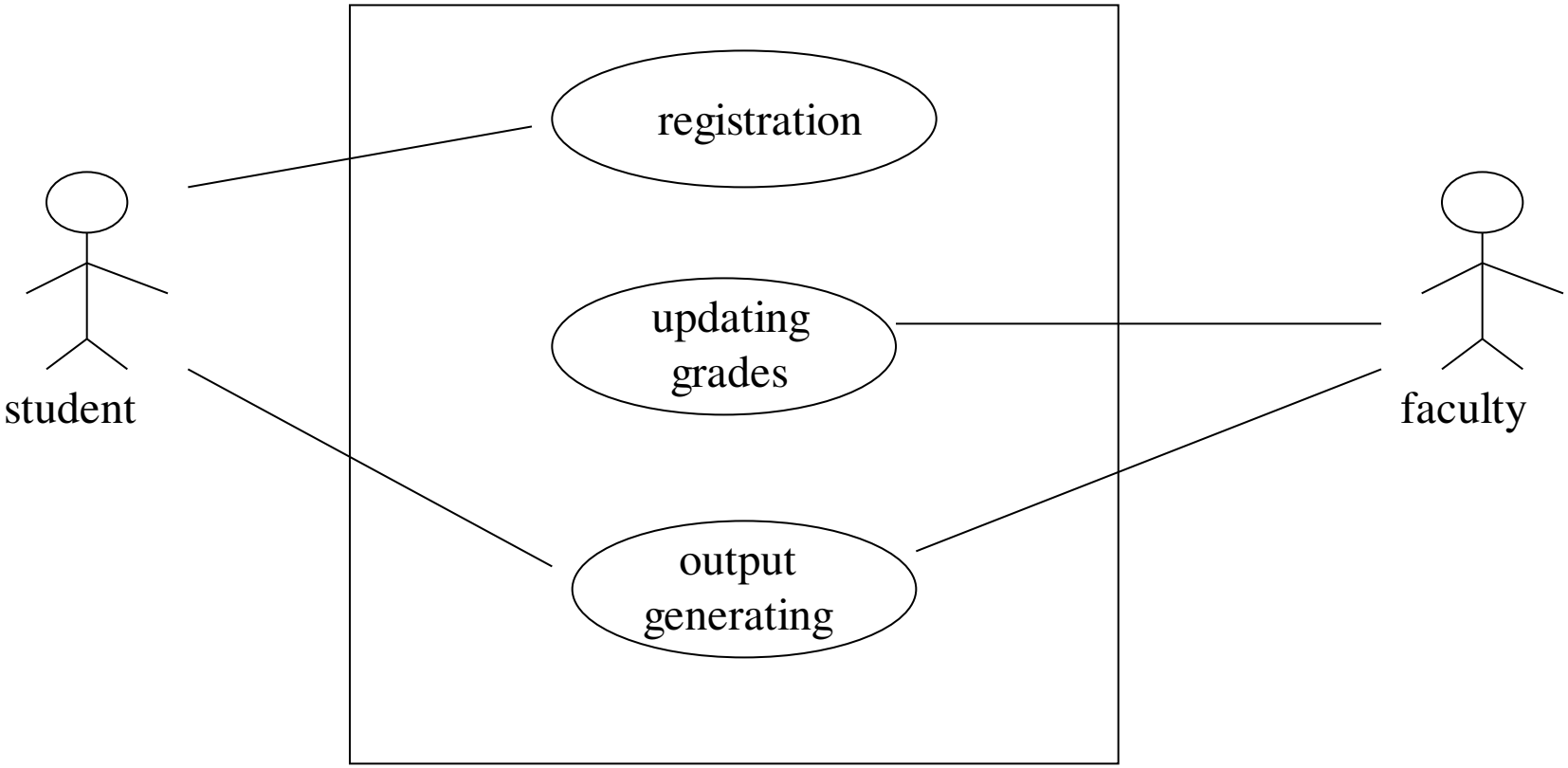
REQUIREMENTS ENGINEERING

Example of Use Case Diagram



REQUIREMENTS ENGINEERING

Example

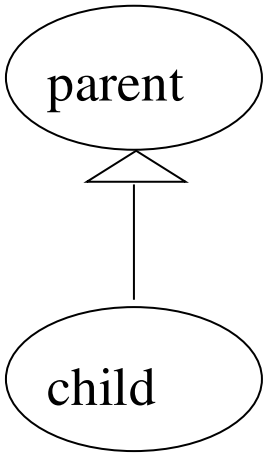


Relationships between Use Cases

1. Generalization - use cases that are specialized versions of other use cases.
2. Include - use cases that are included as parts of other use cases. Enable to factor common behavior.
3. Extend - use cases that extend the behavior of other core use cases. Enable to factor variants.

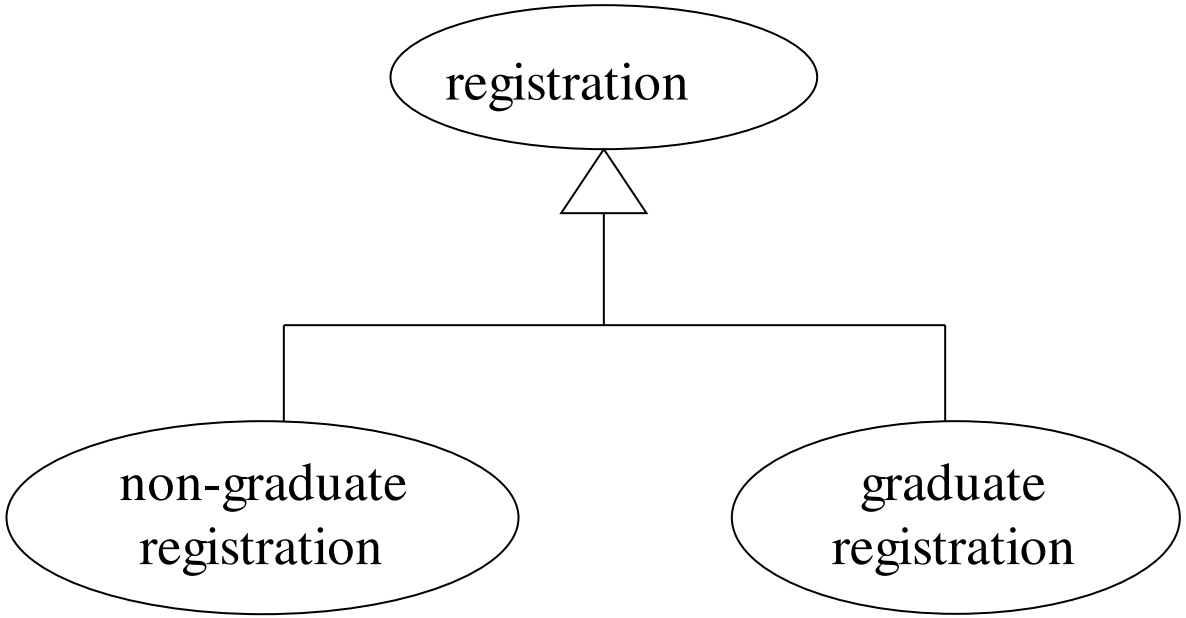
1. Generalization

- The child use case inherits the behavior and meaning of the parent use case.
- The child may add to or override the behavior of its parent.
- Share the same relationship to the actor



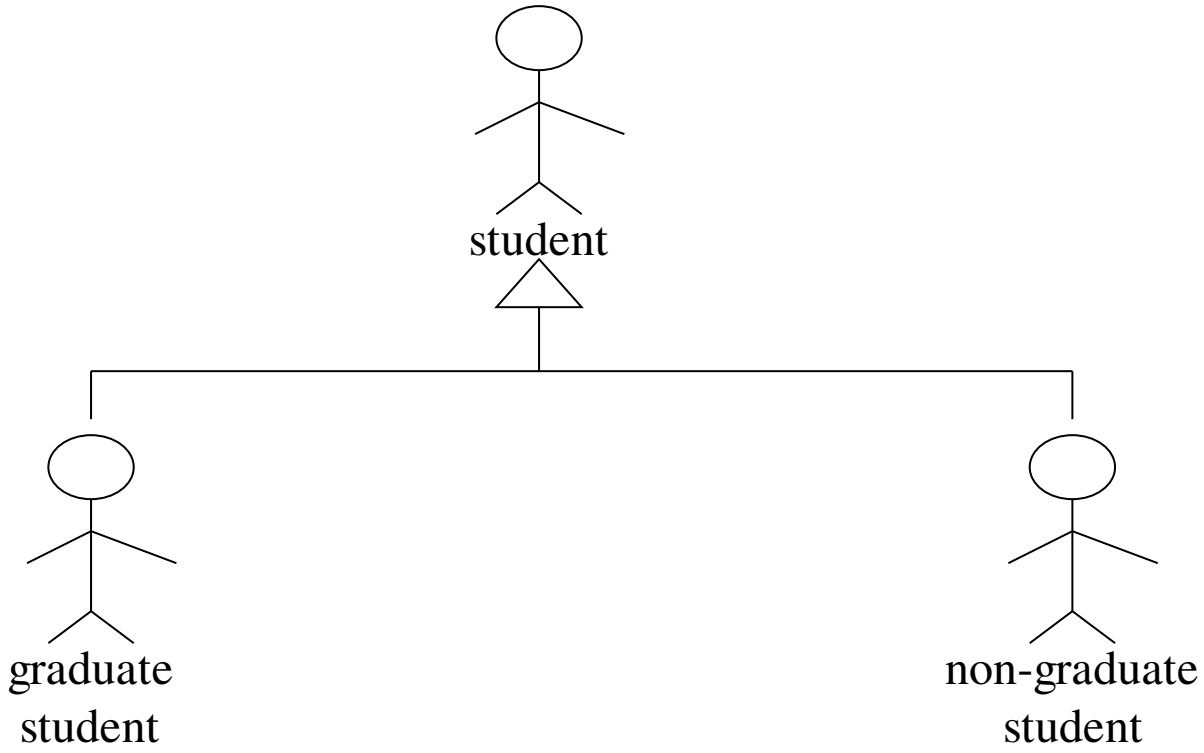
REQUIREMENTS ENGINEERING

More about Generalization



REQUIREMENTS ENGINEERING

Generalization as a Relationships between Actors

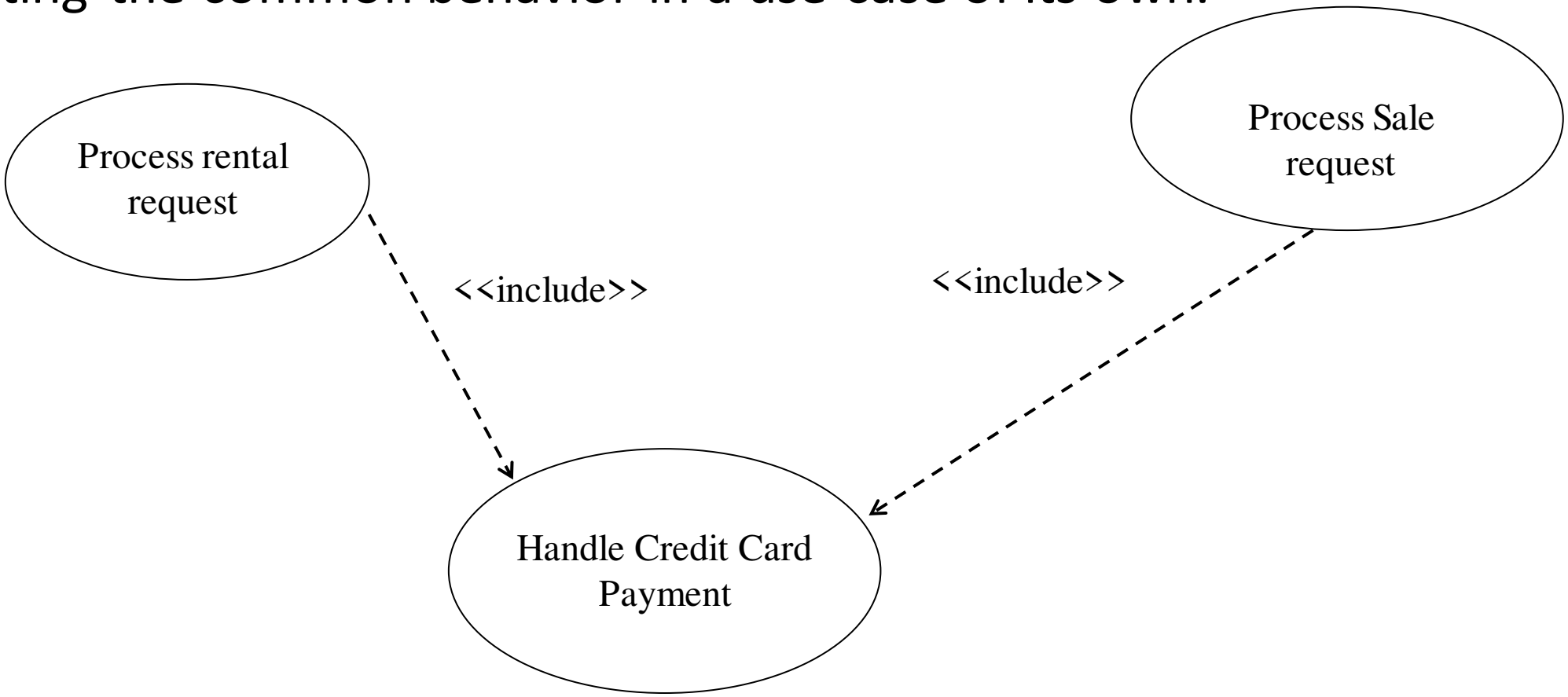


2. Include

- The base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- The included use case never stands alone. It only occurs as a part of some larger base that includes it.

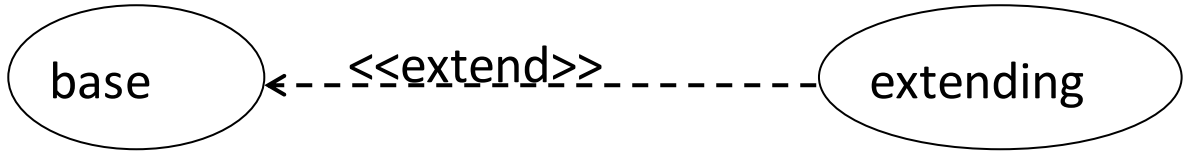
More about Include

- Enables to avoid describing the same flow of events several times by putting the common behavior in a use case of its own.



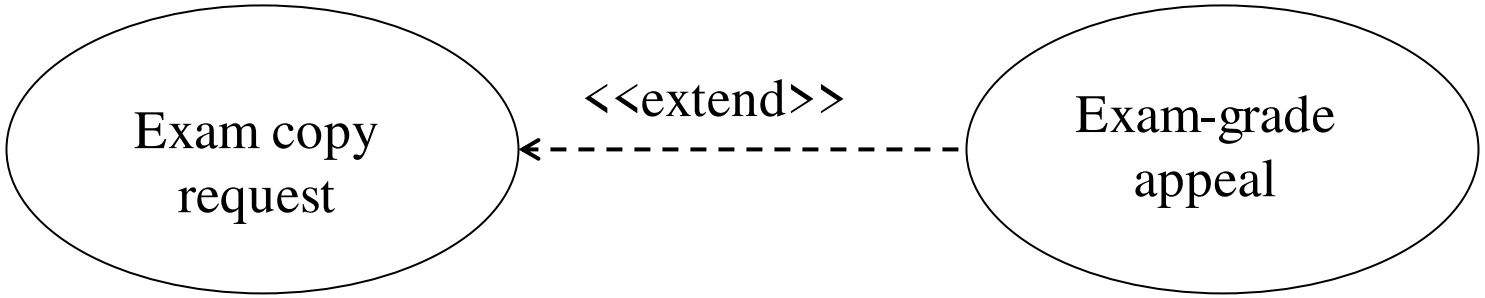
3. Extend

- The base use case implicitly incorporates the behavior of another use case at certain points called extension points.
- The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case.



More about Extend

- Enables to model optional behavior or branching under conditions.

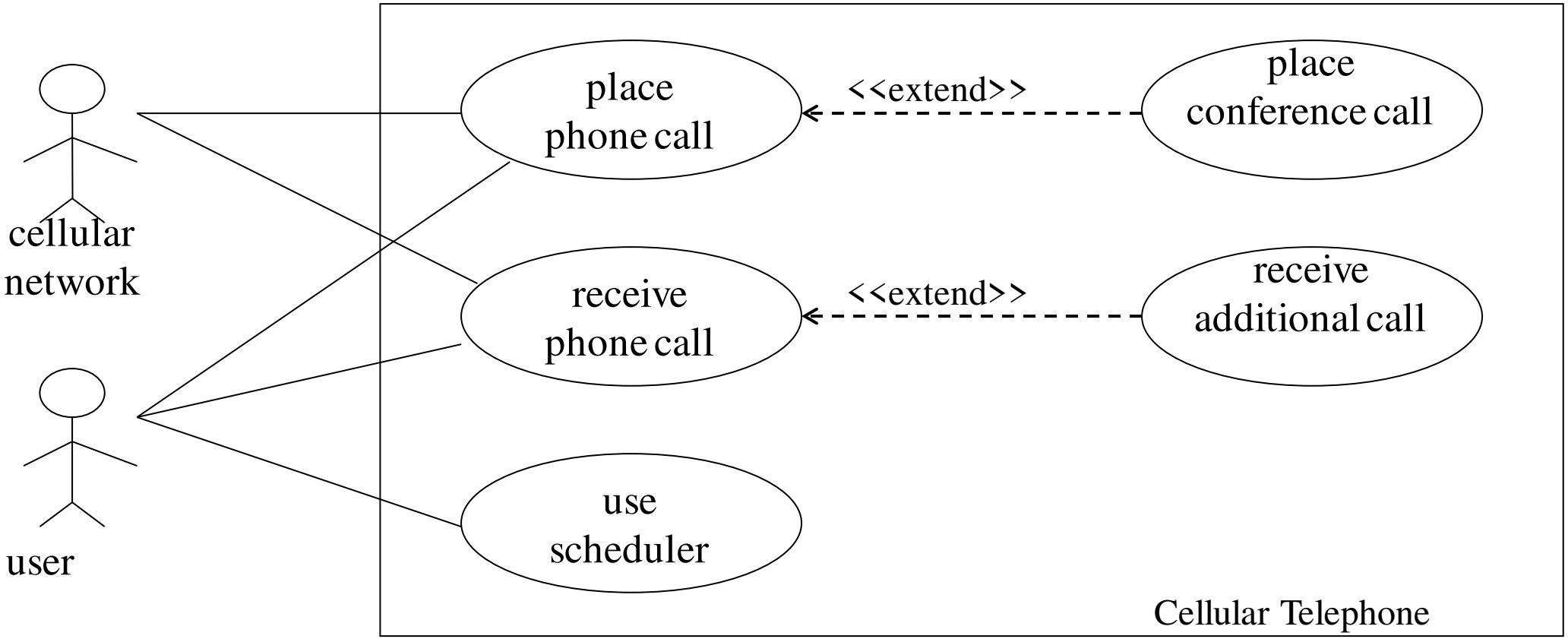


Relationships between Use Cases and Actors

- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

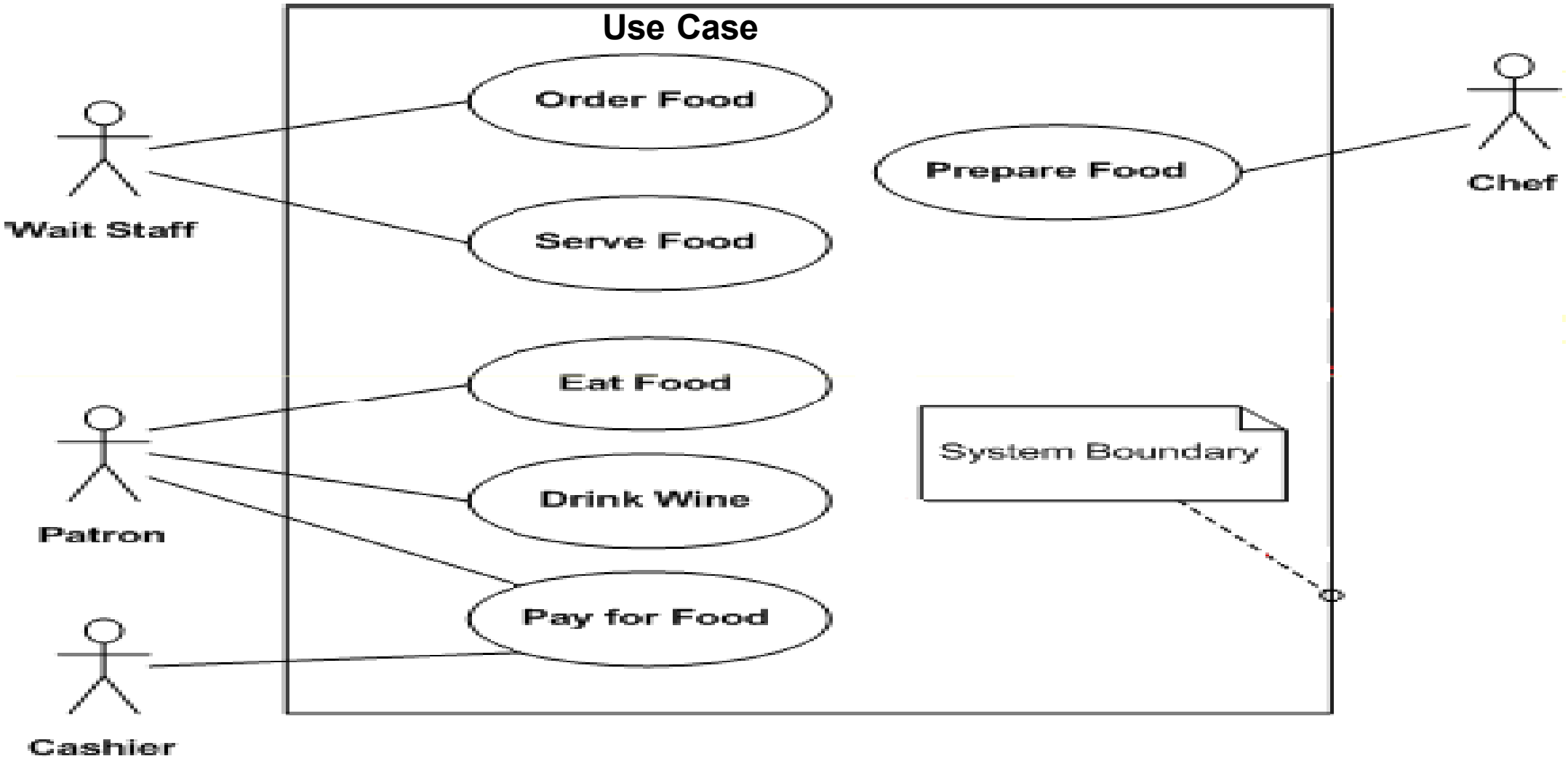
REQUIREMENTS ENGINEERING

Example



REQUIREMENTS ENGINEERING

Use case diagram of a restaurant



REQUIREMENTS ENGINEERING

Formal template for representing of a Use Case – Use case Specification

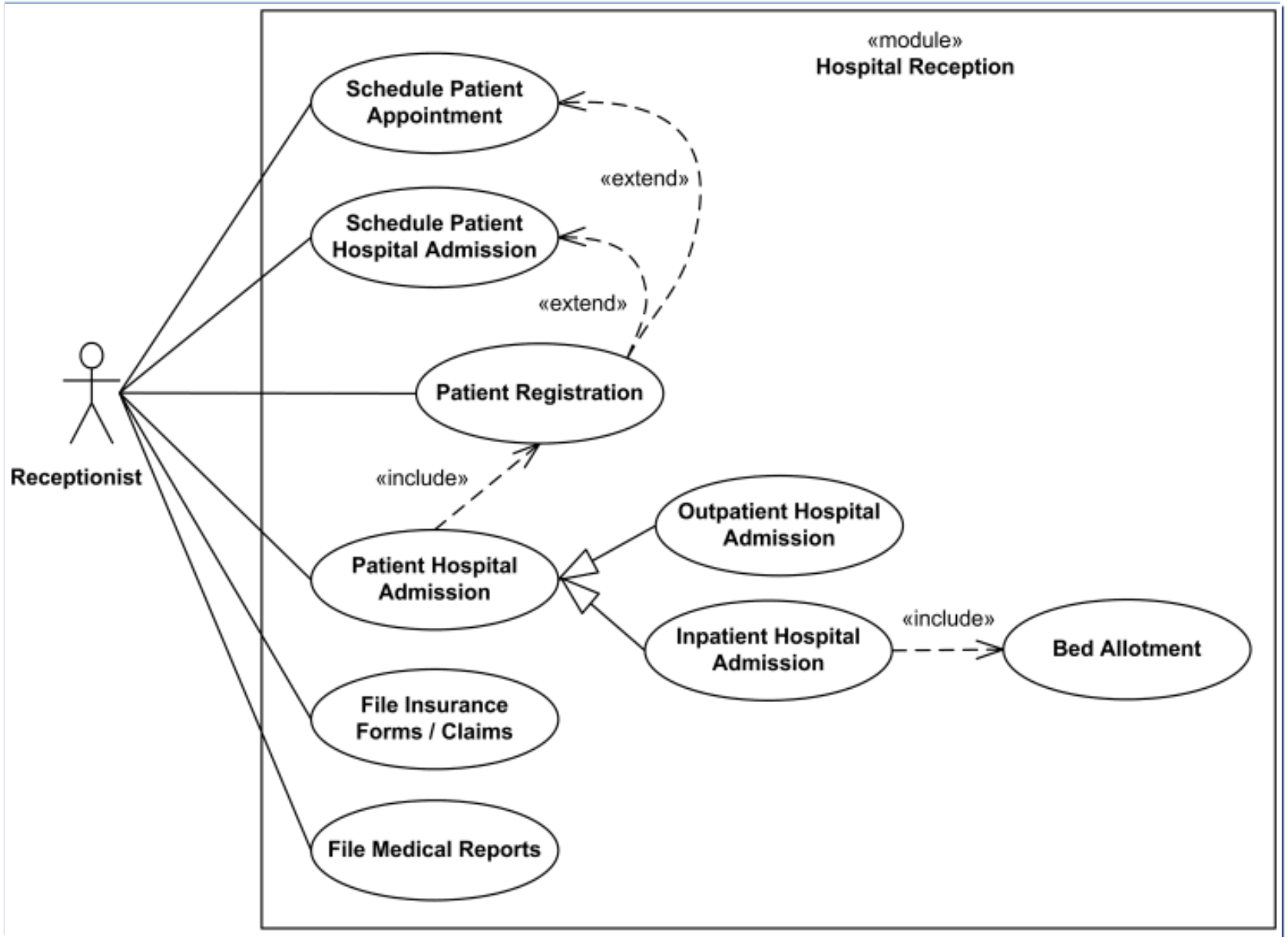
- Name (Must start with a verb)
- Summary
- Actors
- Pre-conditions
  - Conditions that must exist *before* the use case is executed
- Description
  - Textual description (may include steps to execute) and typically is the primary functionality
- Exceptions
  - These are paths which will need to handle exceptions which could be all to provide handling of things which are not provide you with a primary functionality including things like power failure
- Alternate Flows
  - Handles the other functionality paths for the summary these could be some in the exceptions too
- Post-conditions
  - Conditions that must exist *after* the use case is executed

REQUIREMENTS ENGINEERING

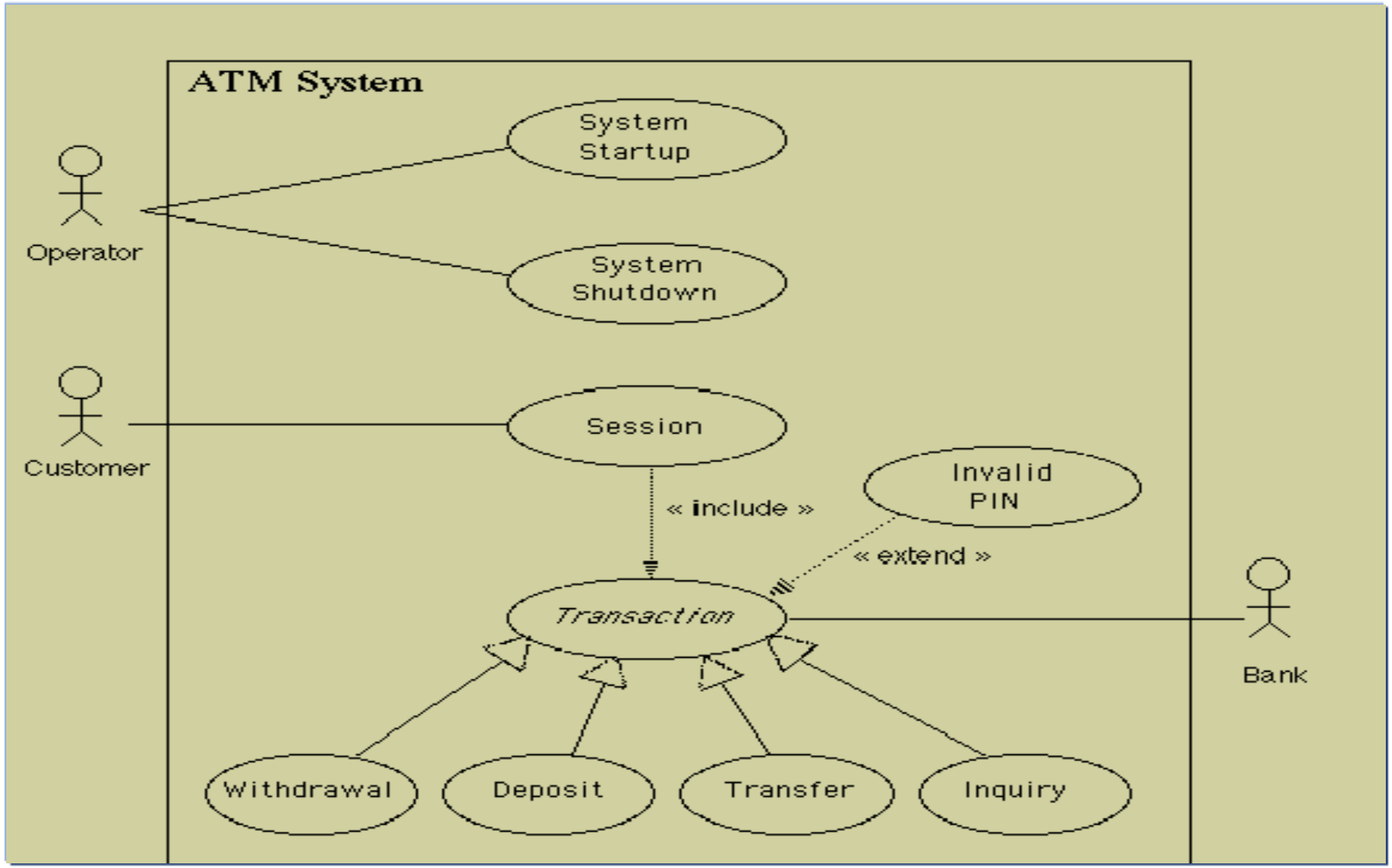
Example (Internet Banking)

- Name: Transfer Funds
- Summary/Overview : Transfer funds from one account to another
- Actor: Customer
- Pre-conditions
  - Source account must have sufficient funds
- Description
  - Customer identifies the accounts from which and to which funds have to be transferred, enters the amount to be transferred, and confirms the transaction
- Exceptions
  - Cancel, Insufficient funds, Cannot identify destination account
  - Needs to handle ..say power failure
- Alternate Flows
  - Handles all the exception paths
- Post-conditions
  - Funds transferred and account balances updated

REQUIREMENTS ENGINEERING



REQUIREMENTS ENGINEERING



REQUIREMENTS ENGINEERING

How to create Use Case diagram

- List main system functions (use cases) in a column
- Draw ovals around the function labels
- Draw system boundary
- Draw actors and connect them with use cases
- Specify include and extend relationships between use cases

REQUIREMENTS ENGINEERING

Use Case Description

Each use case may include all or part of the following:

- Title or Reference Name - meaningful name of the UC
- Author/Date - the author and creation date
- Modification/Date - last modification and its date
- Purpose - specifies the goal to be achieved
- Overview - short description of the processes
- Cross References - requirements references
- Actors - agents participating
- Pre Conditions - must be true to allow execution
- Post Conditions - will be set when completes normally
- Normal flow of events - regular flow of activities
- Alternative flow of events - other flow of activities
- Exceptional flow of events - unusual situations
- Implementation issues - foreseen implementation problems

Example- Money Withdraw

- Actors: Customer
- Pre Condition:
  - The ATM must be in a state ready to accept transactions
  - The ATM must have at least some cash on hand that it can dispense
  - The ATM must have enough paper to print a receipt for at least one transaction
- Post Condition:
  - The current amount of cash in the user account is the amount before the withdraw minus the withdraw amount
  - A receipt was printed on the withdraw amount
  - The withdraw transaction was audit in the System log file

Example- Money Withdraw (cont.)

- **Typical Course of events:**

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

Example- Money Withdraw (cont.)

- Alternative flow of events:
  - Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - Step 8: Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - Step 8: Customer exceeds its legal amount. Display an error message, and go to step 6.
- Exceptional flow of events:
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card

REQUIREMENTS ENGINEERING

Example- Money Withdraw (cont.)

- One method to identify use cases is actor-based:
  - Identify the actors related to a system or organization.
  - For each actor, identify the processes they initiate or participate in.
- A second method to identify use cases is event-based:
  - Identify the external events that a system must respond to.
  - Relate the events to actors and use cases.
- The following questions may be used to help identify the use cases for a system:
  - What are tasks of each actor ?
  - Will any actor create, store, change, remove, or read information in the system ?
  - What use cases will create, store, change, remove, or read this information ?
  - Will any actor need to inform the system about sudden, external changes ?
  - Does any actor need to be informed about certain occurrences in the system ?
  - Can all functional requirements be performed by the use cases ?

THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

REQUIREMENTS ENGINEERING

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

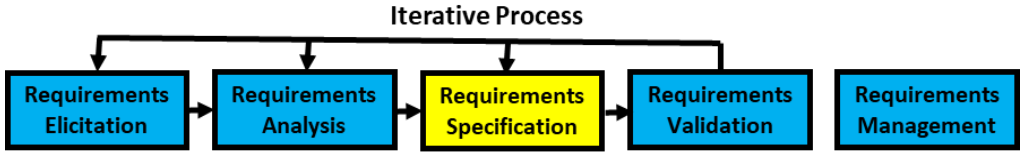
REQUIREMENTS ENGINEERING

Requirements Specification

Prof. Phalachandra H. L
Department of Computer Science and Engineering

REQUIREMENTS ENGINEERING

Requirements Specification



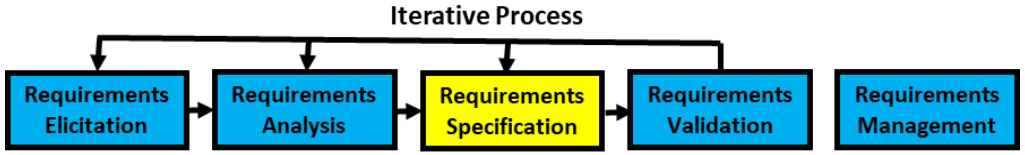
Once requirements are elicited .. analyzed .. we then ***specify those requirements***..

Requirements specification is the documentation of a set of requirements that is reviewed and approved by the customer and provides direction for the software construction activities in the next stage of the life cycle

The ***software requirements specification (SRS)*** document is the basis for customers and contractors/suppliers agreeing on what the product will and will not do. It describes both the functional and nonfunctional requirements

REQUIREMENTS ENGINEERING

Documentation Characteristics



Reasons for documentation:

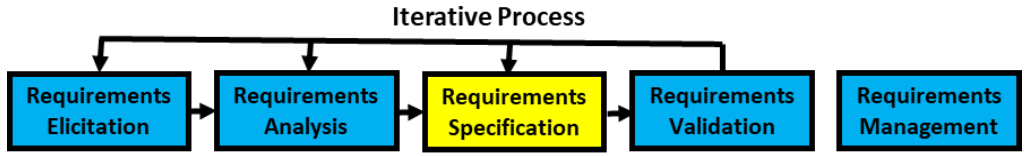
- Visibility (e.g. needed for project plan)
- Formalization leads to better clarity when expressed in writing
- User support (e.g., user manual)
- Team communication (e.g. interface specifications, global teams)
- Maintenance and evolution (e.g., requirements and easier control)

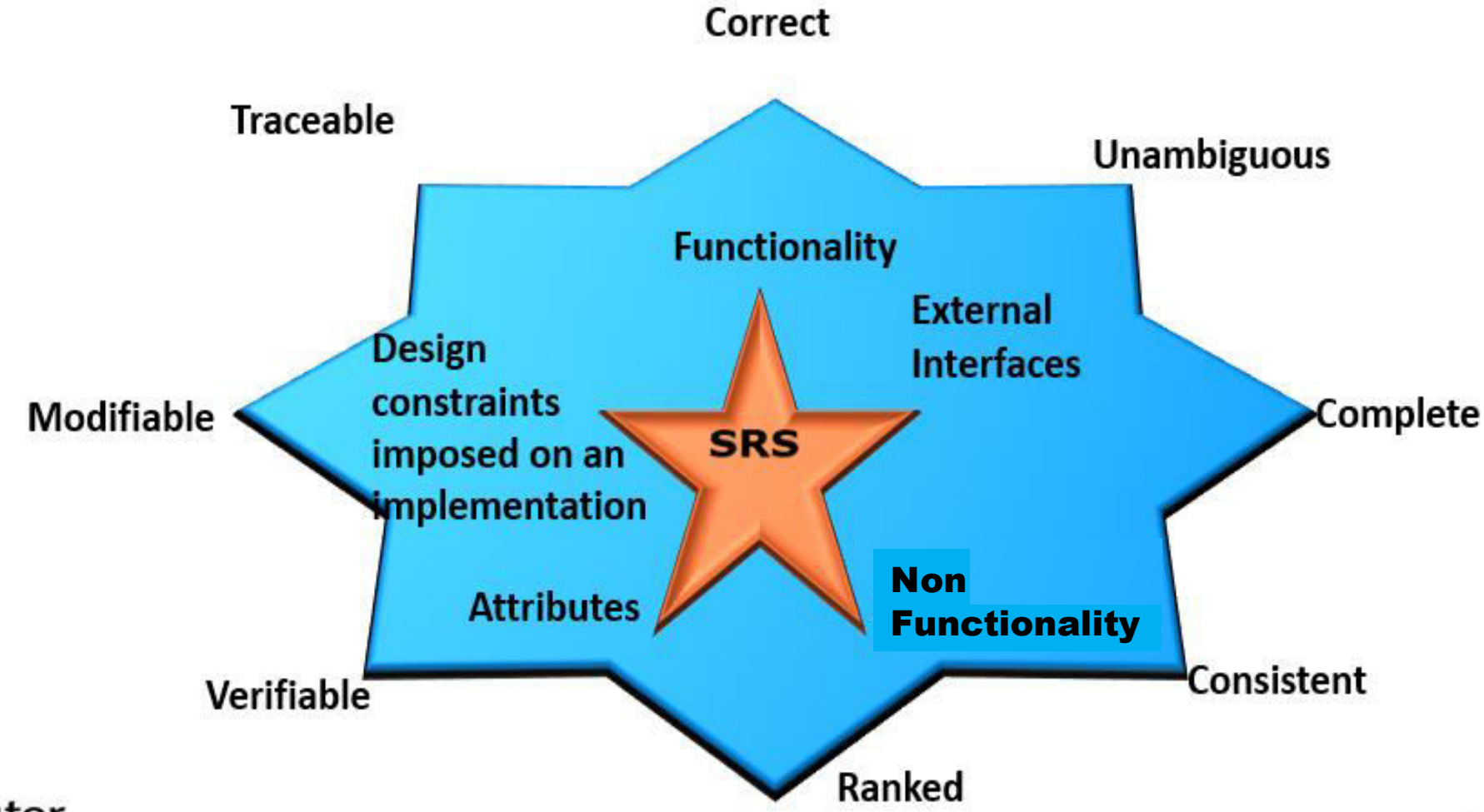
Characteristics of documentation:

- Accurate and kept current
- Appropriate for audience
- Maintained online (usually)
- Simple but professional in style and appearance

REQUIREMENTS ENGINEERING

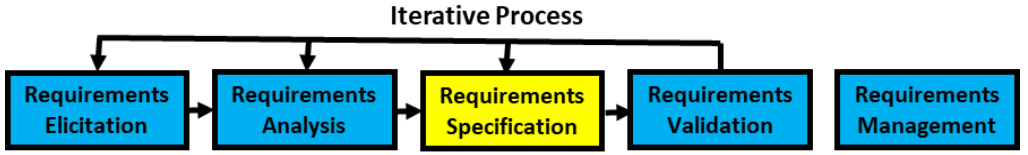
Requirements Specification





REQUIREMENTS ENGINEERING

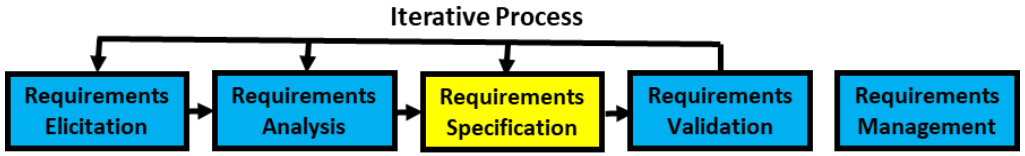
Software Requirements Specification (SRS)



- **Functionality:** What is the software supposed to do?
- **External interfaces.** How does the software interact with people, the system's hardware, other hardware, and other software?
- **Non Functionality.**
  This includes all of the Quality criteria's which drive the functionality.
  These could be
  - Performance - What is the speed, response time, recovery time, etc., of various software functions?
  - Other Quality Attributes like Availability, Portability, correctness, maintainability, etc.
- **Design constraints imposed on an implementation**
  - required standards in effect
  - implementation language
  - policies for database integrity
  - resource limits
  - security
  - operating environment(s) etc.?

REQUIREMENTS ENGINEERING

Software Requirements Specification (SRS)



ToC recommended by IEEE for SRS (IEEE Std. 830-1998)

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms, and abbreviations
- 1.4 References
- 1.5 Overview

2. Overall description

- 2.1 Product perspective
- 2.2 Product functions
- 2.3 User characteristics
- 2.4 Constraints
- 2.5 Assumptions and dependencies

3. Specific requirements

- 3.1 External Interface
- 3.2 Functional Requirements
- 3.3 Non-Functional Requirements
- 3.4 Design Constraints

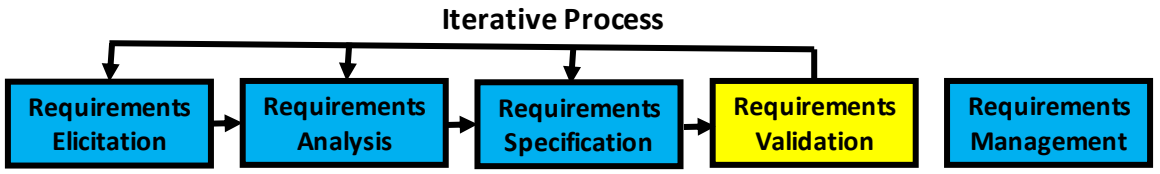
Appendixes

Index

Illustration

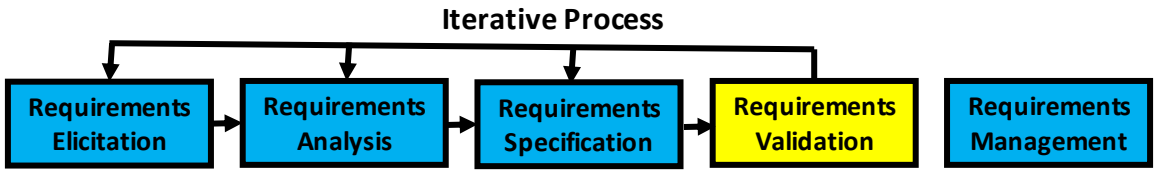
REQUIREMENTS ENGINEERING

Requirements Validation

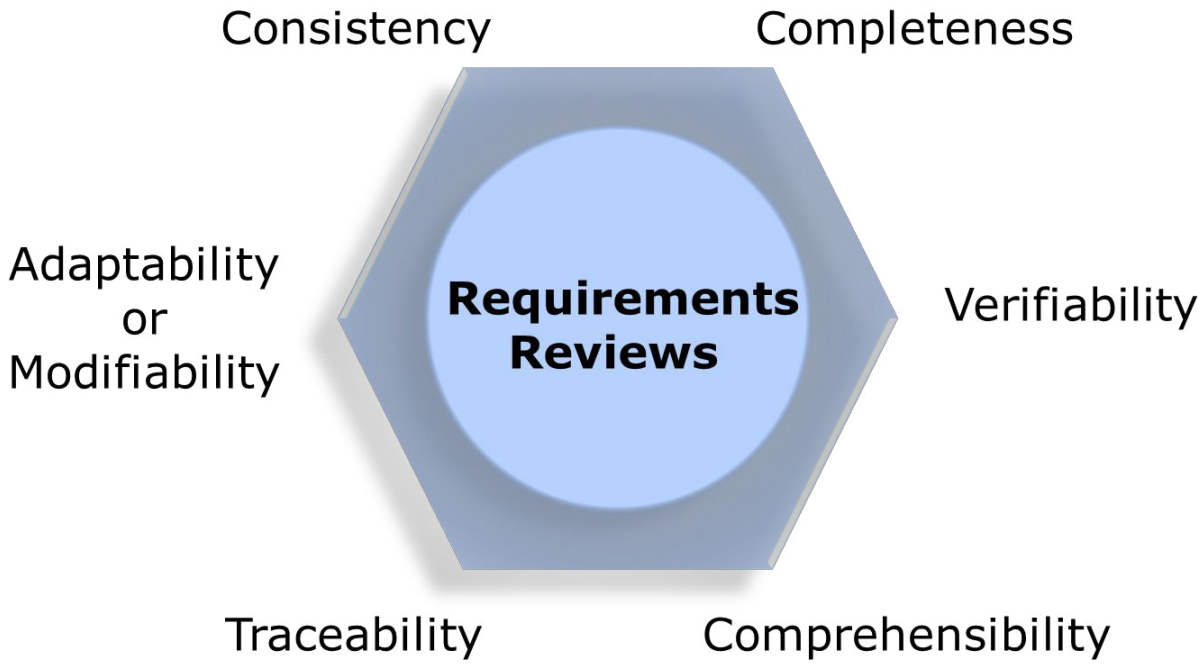


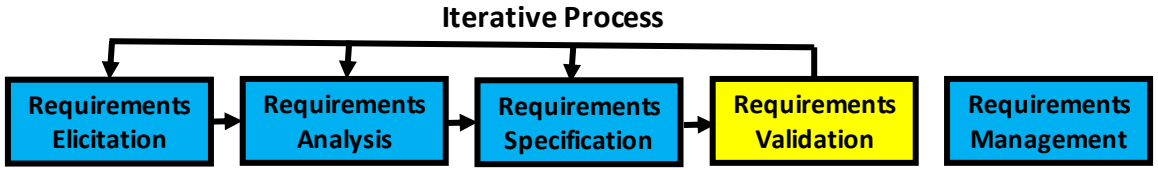
- The purpose of requirements is to help ensure that the requirements does what the customer wants
- This is an important phase because repairing requirement errors in downstream phases can be expensive
- Validation and Verification
  - Validation determines whether the software requirements if implemented, will solve the right problem and satisfy the intended user needs
  - Verification determines whether the requirements have been specified correctly

(Reviews are used for both validation and verification)



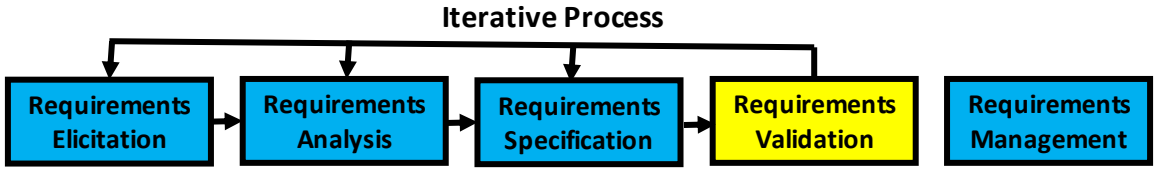
1. Requirement Reviews





2. Prototyping

- Prototype facilitates user involvement during requirements engineering phase and ensures engineers and users have the same interpretation of the requirements
- Prototyping is most beneficial in systems
  - that will have many user interactions
    - Example - design of online billing systems where the use of screen dialogs is extensive
- Systems with little or no user interaction may not benefit as much from prototyping
  - such as batch processing or systems that mostly do calculations



3. Model Validation

- Ensuring that the models represent all essential functional requirements These could be Use Case or other models
- Demonstrating that each model is consistent in itself

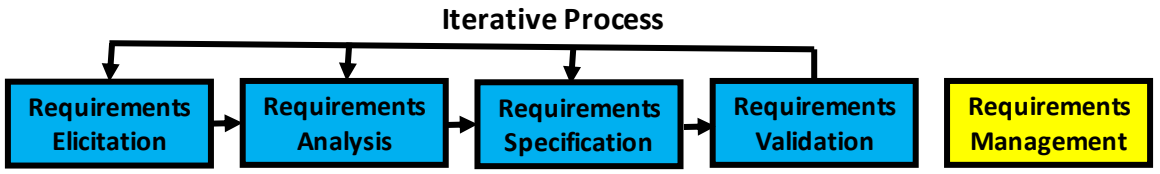
  Example : Flow diagrams, class models and static analysis, verify for existence of proper communication paths to facilitate data exchange
- Usage of the Fish Bone Analysis technique to validate if the requirements identified is addressing the reasons needing the solution to the problem which had led to the requirements

4. Acceptance Criteria

- Using Acceptance criteria's to see if there are requirements matching with that the Acceptance criteria

REQUIREMENTS ENGINEERING

Requirements Management



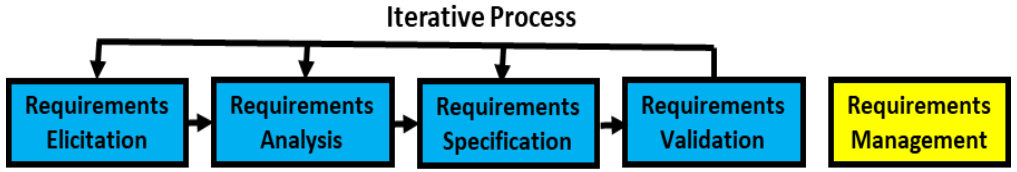
- Requirements specification is the baseline on which the future lifecycle phases will need to build upon
- Requirements can change due to
  - Better understanding of the problem
  - Customer internalizing the problem and solution
  - Evolving environment and technology landscape

Requirements management has two facets

- Ensuring that the requirements are all addressed in each phases of the lifecycle
- Ensuring that the changes in the requirements are handled appropriately

REQUIREMENTS ENGINEERING

Requirements Traceability Matrix - RTM

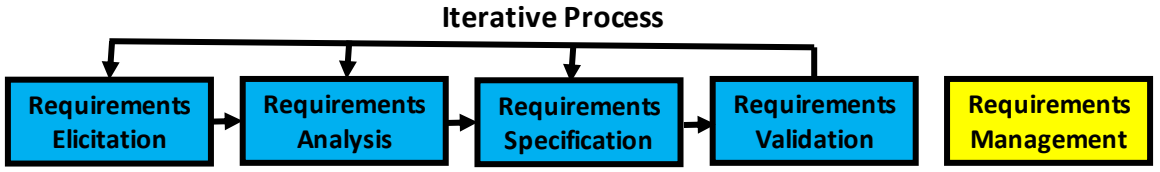


Req Id	Architectural Section	Design Section	File/ Implementation	Unit Test Id	Functional Test ID	System Test ID	Acceptance Test Id

- Requirements identification – This could be a number like FR-1/NF-1 or a unique use case reference UC-1, goal-hierarchy numbering along with version information etc.)
- Every phase of the SDLC progressively fills the RTM
- Requirements are traced across the SDLC using the requirement traceability matrix (RTM)
  - Forward Tracing
  - Backward Tracing

REQUIREMENTS ENGINEERING

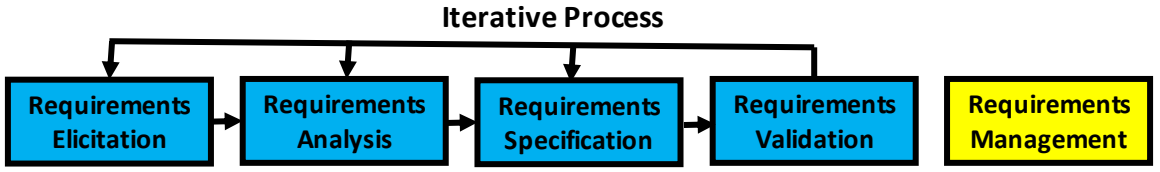
Requirements Change Management



- Reiterating, number of things can change in the environment during the development … leading to change in Requirements
- Change in the requirements have impacts on plans, work products etc.
- Uncontrolled changes can have a huge adverse impact on project in terms of cost, schedule, quality and expectations
- Changes have to be allowed, but in a controlled manner or in other words has to be managed
- In the perspective of managing the changes, change requests go through a formal change management process as in the subsequent slides

REQUIREMENTS ENGINEERING

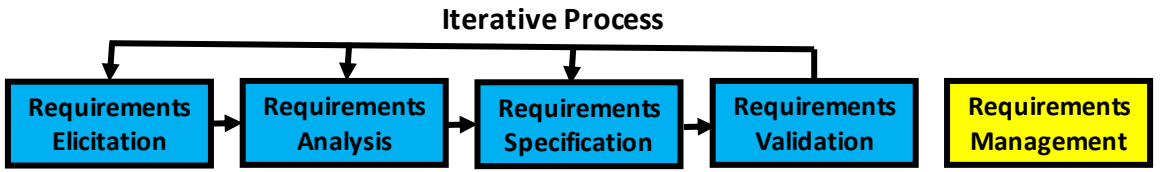
Requirements Change Process



- Log the request for change and assign a change request Identifier
- Log
  - Who is requesting for the change
  - Why is the request coming in
  - What is being requested to change
- Perform impact analysis on the work products and items
- Estimate impact on scope, effort and schedule (may be quality)
- Review impact with stakeholders
- Solicit Formal Approval as part of the approval process
- Rework the work products/items
- Log the following post changes
  - When was it changed
  - Who all made changes
  - Who reviewed the changes
  - Who tested the changes
  - Which release stream is the change going to be part of

REQUIREMENTS ENGINEERING

Requirements Change Management



Change Control

- Impact of change at specific phases
- Responding to change requests

Traceability

- Links between user requirement, architecture design, implementation, testing of all system components

Version Control

- Record approved changes
- Communicate changes
- New document versions

Status Tracking

- Pending
- Approved
- Rejected
- Deferred
- Validated
- Completed
- Cancelled

THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu

SOFTWARE ENGINEERING

SOFTWARE PROJECT MANAGEMENT

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet and supplemented by my experience. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

SOFTWARE PROJECT MANAGEMENT

Software Project Management Fundamentals

Prof. Phalachandra H. L
Department of Computer Science and Engineering

Software Project Management Fundamentals

- Software Engineering is very relevant for large projects
- These large projects could involve several people (10s to 100s to 1000s) working together for a prolonged length of time (several months to years)
- These projects need to be planned (which is significant % of the management activity) and controlled
- This needs co-ordination for ensuring Integration/ interoperability/ & working together

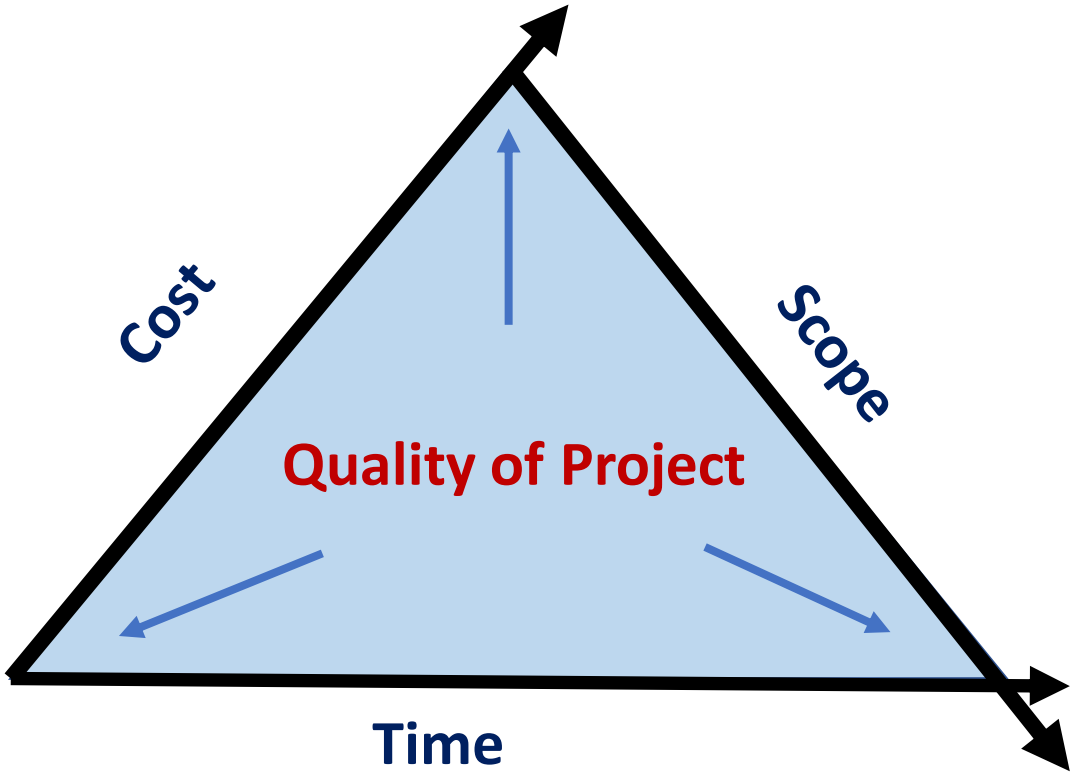
SOFTWARE PROJECT MANAGEMENT

Software Projects and their characteristics

Software projects are temporary individual or collaborative enterprises that are carefully planned to achieve a particular aim/or to create a unique product or service. These projects typically have certain characteristics

- They are made up of certain unique activities which do not repeat under similar circumstances
- They are goal specific
- Typically contain a sequence of activities to deliver the end product
- Time bound- Every project has a specified start and end date
- Technically inter-related activities

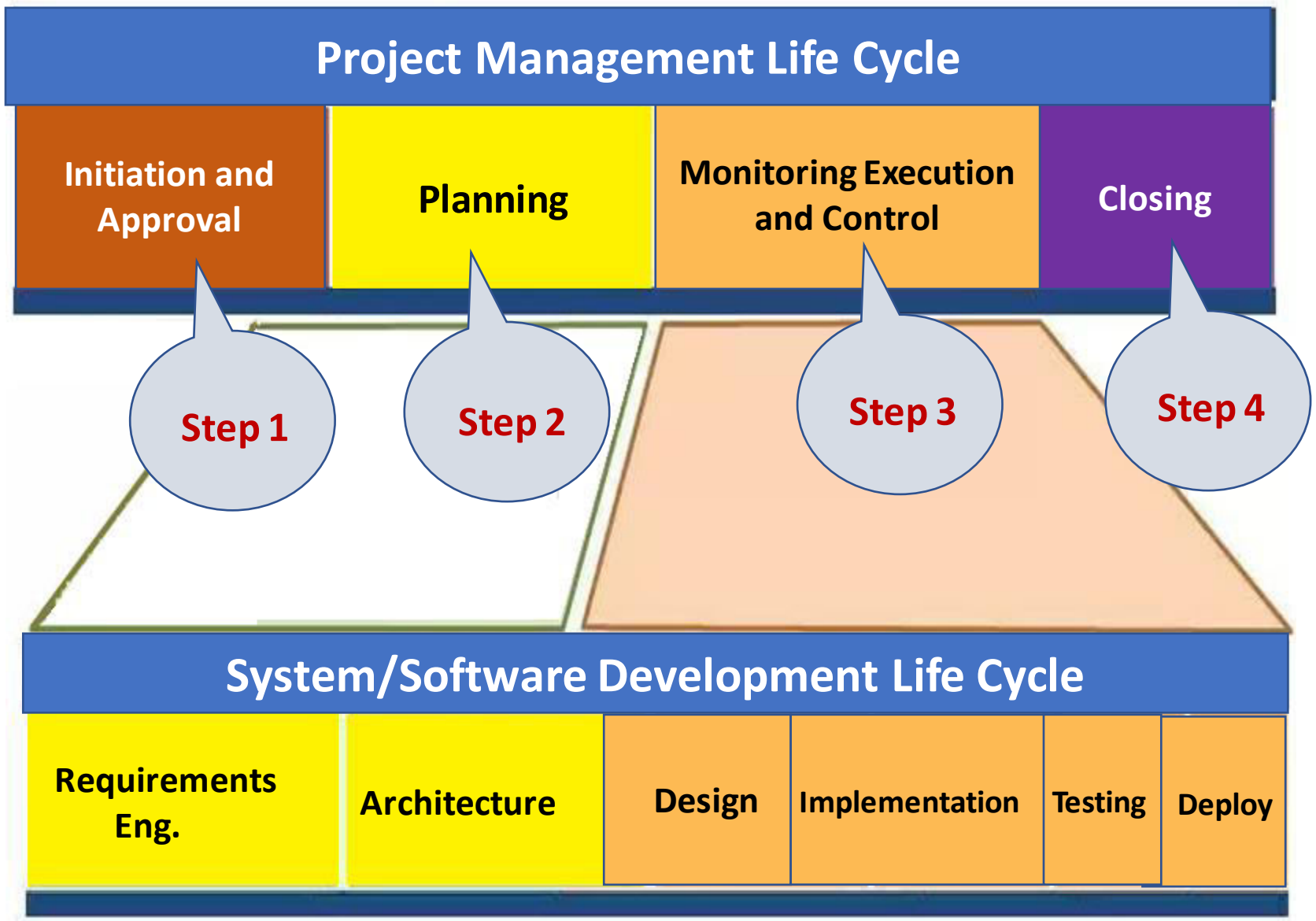
Projects are constrained by Scope, Time (Schedule), Cost (of all Resources)



Project Management will need to maintain this triangle in Equilibrium

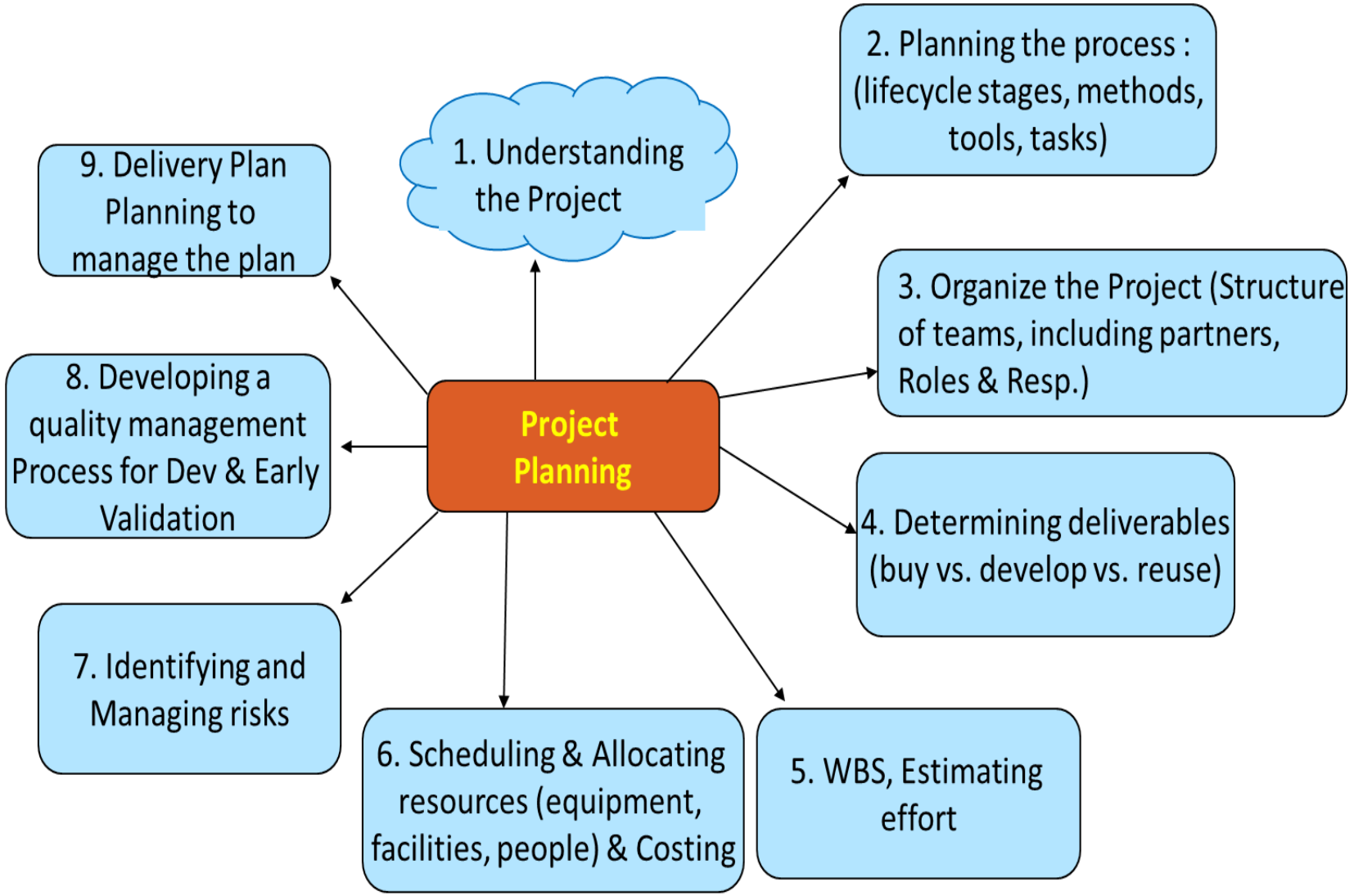
SOFTWARE PROJECT MANAGEMENT

Software Project Management Lifecycle



SOFTWARE PROJECT MANAGEMENT

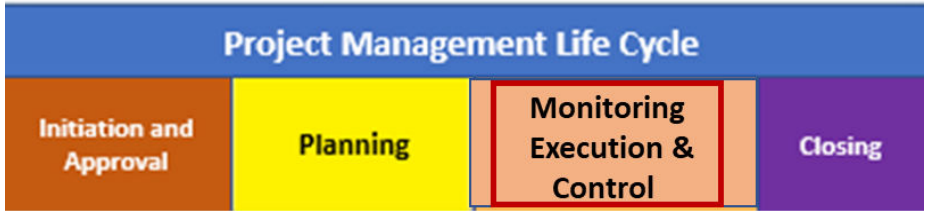
Project planning process



More details towards planning will be discussed in the next session

SOFTWARE PROJECT MANAGEMENT

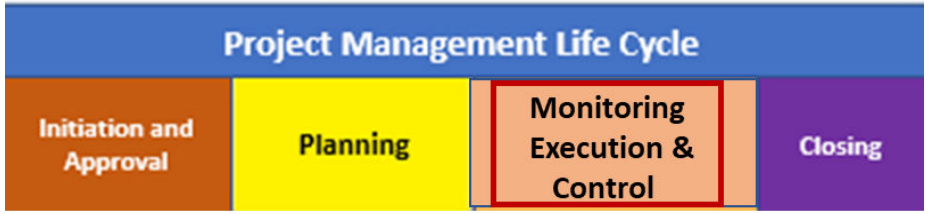
Software Project Monitoring and Control



- **Project Monitoring and control** of the project begins once the project plan is created.
- The **Monitoring** and **Controlling** process is continuously performed and encompasses all the tasks and uses all the measures and metrics necessary to ensure that the **project** is on track e.g. within scope, on-time, within budget and proceeds with minimal risk.
- Project monitoring involves using Quantitative data which is continuously collected all along the project
- Project control activities involves making decisions or adjustments in dimensions like time, cost, organizational structure, Scope etc. for the project under execution

SOFTWARE PROJECT MANAGEMENT

Software Project Monitoring and Control



Project monitoring and control dimensions could be :

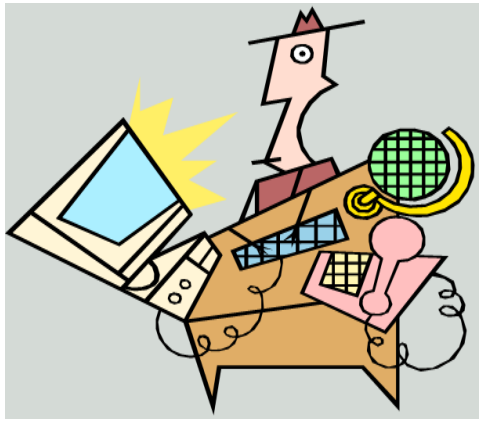
1. Time, both the in terms of effort (number of man-months) and the schedule

- **Measuring progress is hard ("we spent half the money, so we must be halfway")**
- **Development models serve to manage time**
- **More people ⇒ less time?**

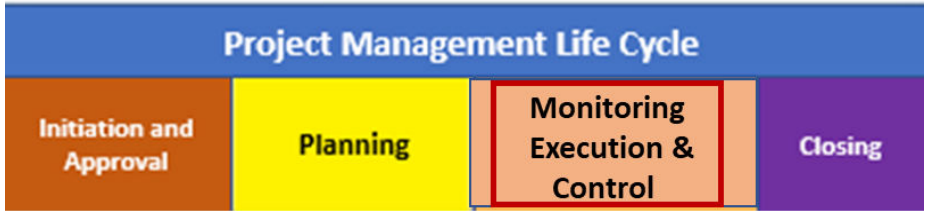
  Brooks' law: adding people to a late project makes it later

Project monitoring and control dimensions (Cont.)

2. Information (availability, propagation)
   - communication including documentation



- **Documentation**
  - **Technical documentation**
  - **Current state of projects**
  - **Changes agreed upon**
  - **…**
- **Agile projects: less attention to explicit documentation, more on tacit knowledge held by people**



Project monitoring and control dimensions (Cont.)

3. Organization and its structure, roles & responsibilities (people and team aspects)

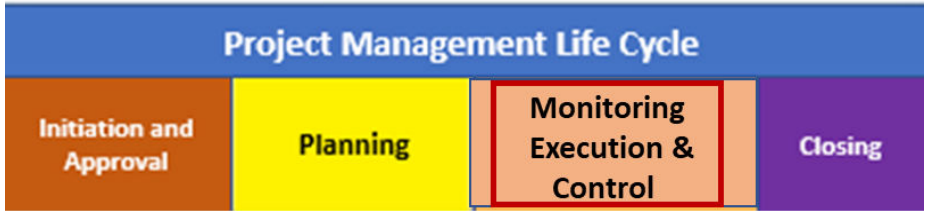
- **Building a team**
- **Reorganizing structures**
- **Clarifying, managing expectations and reorienting roles and responsibilities**
- **Coordination of work**

Project monitoring and control dimensions (Cont.)

4. Quality is not an add-on feature; it has to be built in.

(what are leading indicators and what are lagging indicators)

- **Quality has to be designed in**
- **Quality is not an afterthought**
- **Quality requirements often conflict with each other**
- **Requires frequent interaction with stakeholders**



Project monitoring and control dimensions (Cont.)

5. Cost, Infrastructure & Personnel (Capital and Expense)

- **Which factors influence cost?**
- **What influences productivity?**
- **Relation between cost and schedule**

- Project retrospective also needs to happen, using the data collected more like postmortem of this project to operate better in the next project execution
- Capability maturity models which will be discussed later helps in this perspective

SOFTWARE PROJECT MANAGEMENT

Project Closure

The Project Closure Phase will formally close the project and then report its overall level of success to the sponsor.

Steps Involved

- Hand over deliverables to customer & obtain project or UAT (User Acceptance Testing) sign-off from client to confirm that the team has met the project objectives & the agreed requirements
- Complete documentation and pass it on to Business and cancel supplier contracts
- Release staff and equipment, and inform stakeholders of closure.
- Post Mortem - Post Implementation Review is completed to determine the projects success and identify the lessons learned.
- Of-course Celebrate Success

THANK YOU

Prof. Phalachandra H.L.
Department of Computer Science and Engineering
phalachandra@pes.edu