PES UNIVERSITY ONLINE

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R
Department of Computer Science and Engineering

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials sourced online

Software Engineering

Evaluation Guidelines

Dr. Jayashree R

Department of Computer Science and Engineering

Software Engineering

Context of Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Take a good look at these pictures

What are the expectations from these?

- Machine which flies
- Supports going to a specified place
- Carries people and goods
- Reliably functions

- Market System supporting buying and selling of stocks/bonds/ securities
- Supports buyers and sellers to be local/remote for all transactions
- Regulation, Data Integrity, Security

What are the expectations from these?

- Supports a market place featured around a web site
- Will have number of upgrades and revisions

- System to play a video game for entertainment and competition
- Supports individuals and groups
- Runs on dedicated or general purpose systems
- Supports various levels of skills and complexity

What else do you think they have in common?

LOTS OF LINES OF CODE

In 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

We also have to deal with '*Diminishing Value*' or '*Obsolescence*'

What do you think is the impact of a 90 minute outage at Amazon?

Loss of 2.8 Million $ revenue

Loss of customers

Everything in an environment like Amazon must be functioning in a user friendly way with the expectation that there will be no errors.

All these points have to be factored in!

There is hence a need for –

Interacting with customers and stakeholders **on what is needed**

Understanding **who and how** this is going to be used

Experts in multiple domains

Good planning

Team work

Ability to scale and support

This will require a lot of engineers. Lots of engineers implies many teams and team work!

A quick calculation

Let us assume a software engineer can code 1 LOC a minute.

Then,

1 LOC/minute * 60 minutes/hour = 60 LOC/hour
Say for 40 hours/week,
60*40 = 2400 LOCs/week/SE

Which implies,

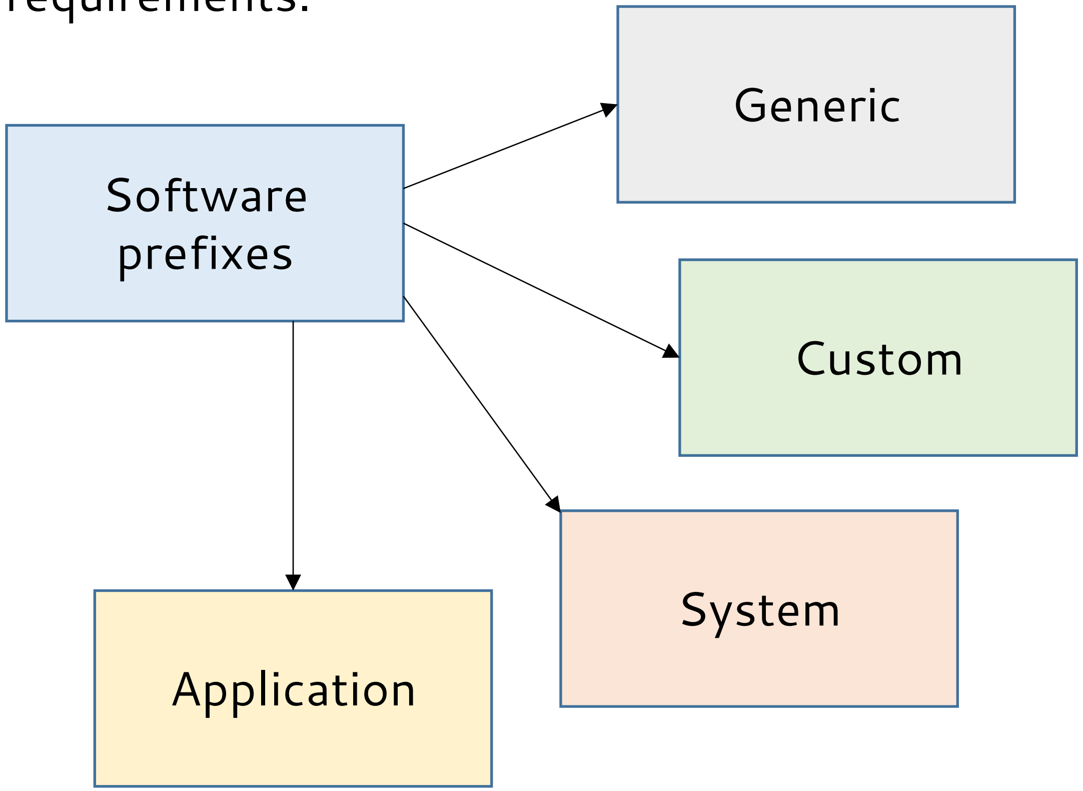
2400 LOC/week * (˜)50 = 1,20,000 LOCs/year/SE

Engineers will thus be working for long periods of time!

Let us look at some definitions

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.



Let us look at some definitions

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.

Software Engineering is the systematic, disciplined, quantifiable approach towards the development, operation, and maintenance of software products and thus supports managing of complexity.

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.

Is Computer Science the same as Software Engineering?

Think about

Consider a bridge collapse.

Is this issue a scientist’s problem or an engineer’s problem?

Hint

Scientist build something to learn something new

whereas

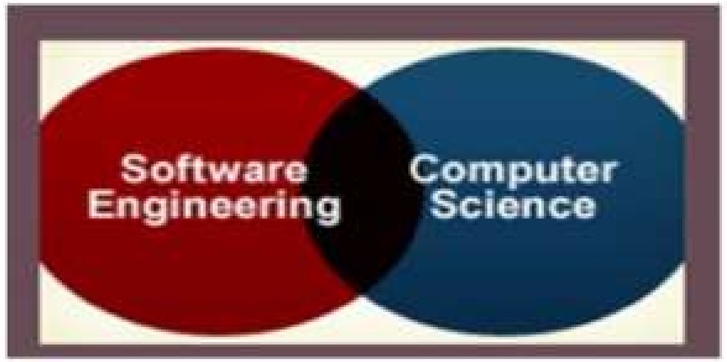
Engineer learns things to design and build quality products

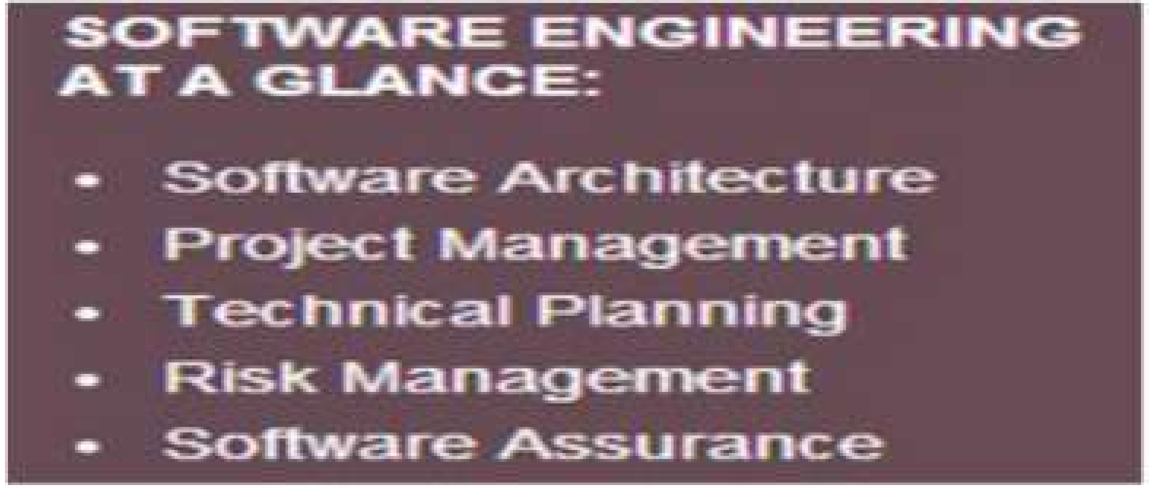
Scientists want to achieve scientific breakthroughs

whereas

Engineers want to avoid engineering failures

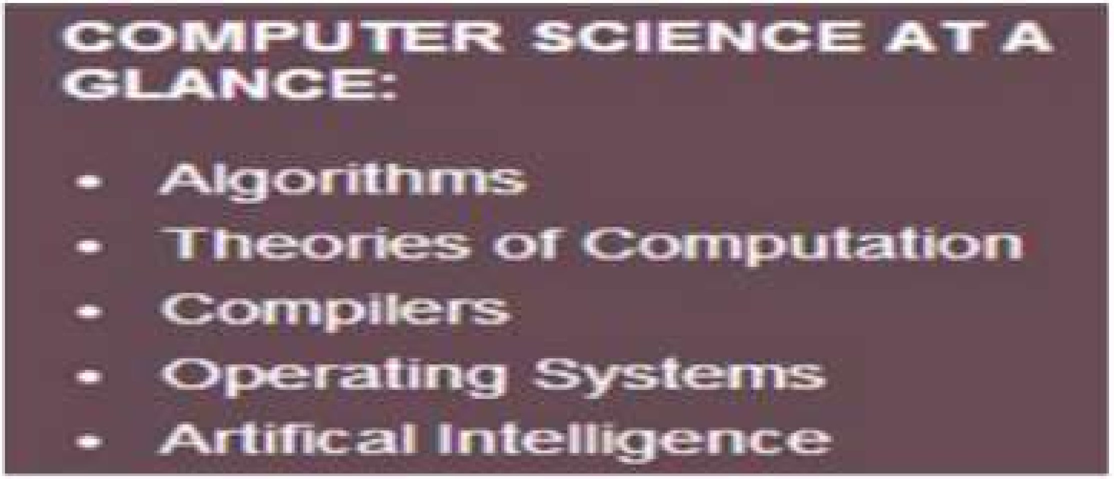
Differences between Computer Science and Software Engineering





SOFTWARE ENGINEERING AT A GLANCE:

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



COMPUTER SCIENCE AT A GLANCE:

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

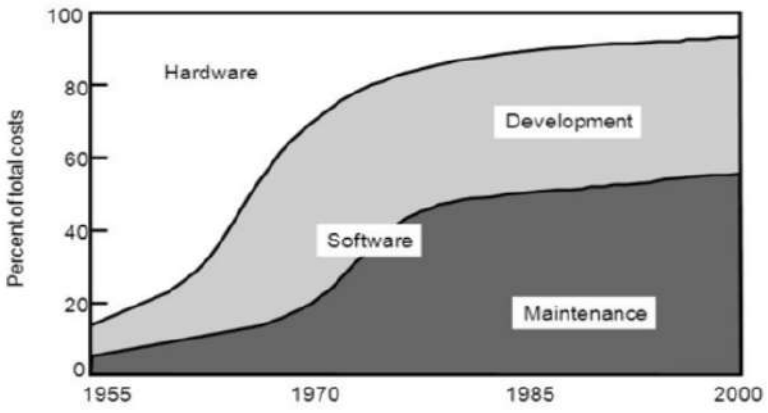
Fundamental Drivers of Software Engineering

Industrial Strength Software

- Needs to be operational
- Capable of being moved
- Needs to be maintainable
- Should have elaborate documentation
- Absence or minimal number of bugs
- Impactful to the business

Software is expensive

- Software labor is expensive
- Each line of code can cost between $5 – $35
- Maintenance and rework



Can influence the life or death of a person

- Example – Therac 25 (Radiation therapy)
- One software bug caused over exposure to radiation
- Death of 6 people

Fundamental Drivers of Software Engineering

Heterogeneity

- Systems should work as distributed systems

Security and trust

- Trust the software

Diversity

- Different types of software systems

Scale

- Scale easily with size

Business and Social changes

- Ability to change existing software and develop new software
- Organizations are becoming global

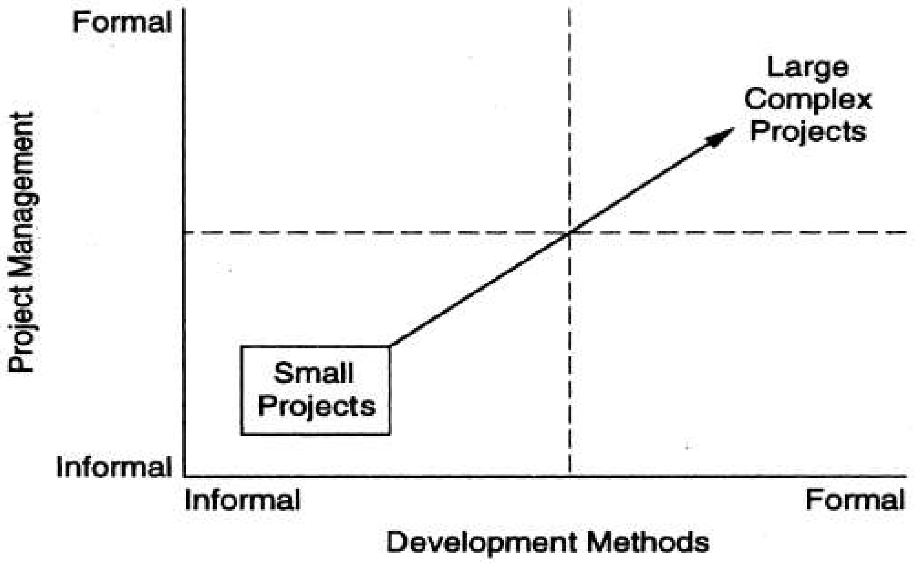
Fundamental Drivers of Software Engineering

Quality and productivity

Consistency and repeatability

Late and unreliable

- Quality as FLURPS + Portability + Efficiency/Maintainabilit y



- Example – Ariane Flight 501

Case Study – ARIANE Flight 501

Late and Unreliable

Typically 35% of the computer based projects are runaway

esa
ariane
©2013 ESA CNES ARIANESPACE / Optique Vidéo du CSG - P BAUDON

Watch the launch here!

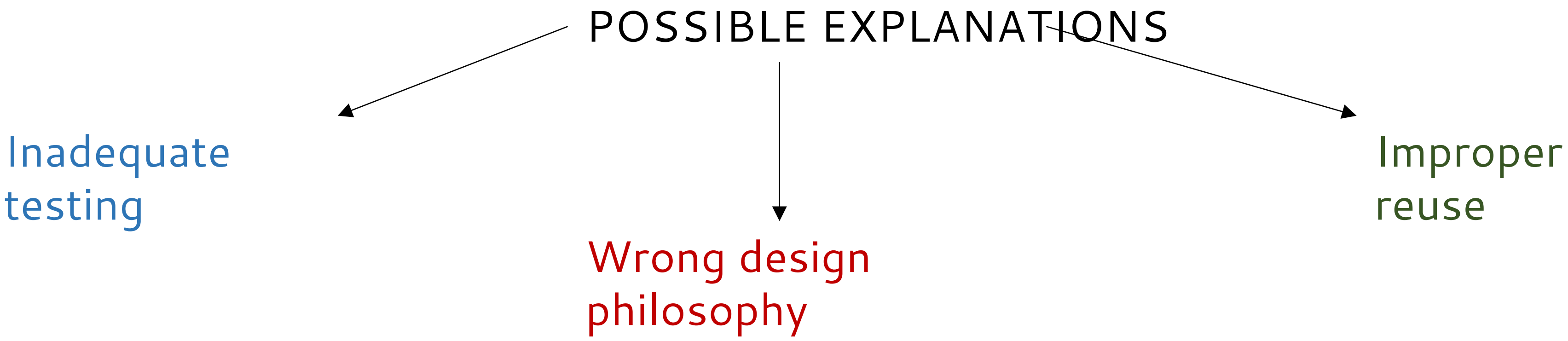
https://www.youtube.com/watch?v=wGeZEUjUKvc

Case Study – 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

Due to

- 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


POSSIBLE EXPLANATIONS

Inadequate testing

Wrong design philosophy

Improper reuse

Summary of why Software Engineering is required

Development of big programs &
Mastering complexity of big programs

Supporting large teams and team work

Efficient development of evolving software

Ensuring software process supports users effectively
&
Right choices and decisions are made

Ensuring visibility and continuity

Can you think of steps of a house

Hints

- Need for a house
- Initial research (sites, architecture, materials)
- Financial analysis
- Hiring a brokerage, architect & contractor
- Meetings & elevation plans
- Changes made to plans
- Different stages of construction (foundation, molding etc.)
- Plumbing and electrical fittings
- Woodwork & interior décor
- House warming ceremony
- Shifting & moving

Software Lifecycle

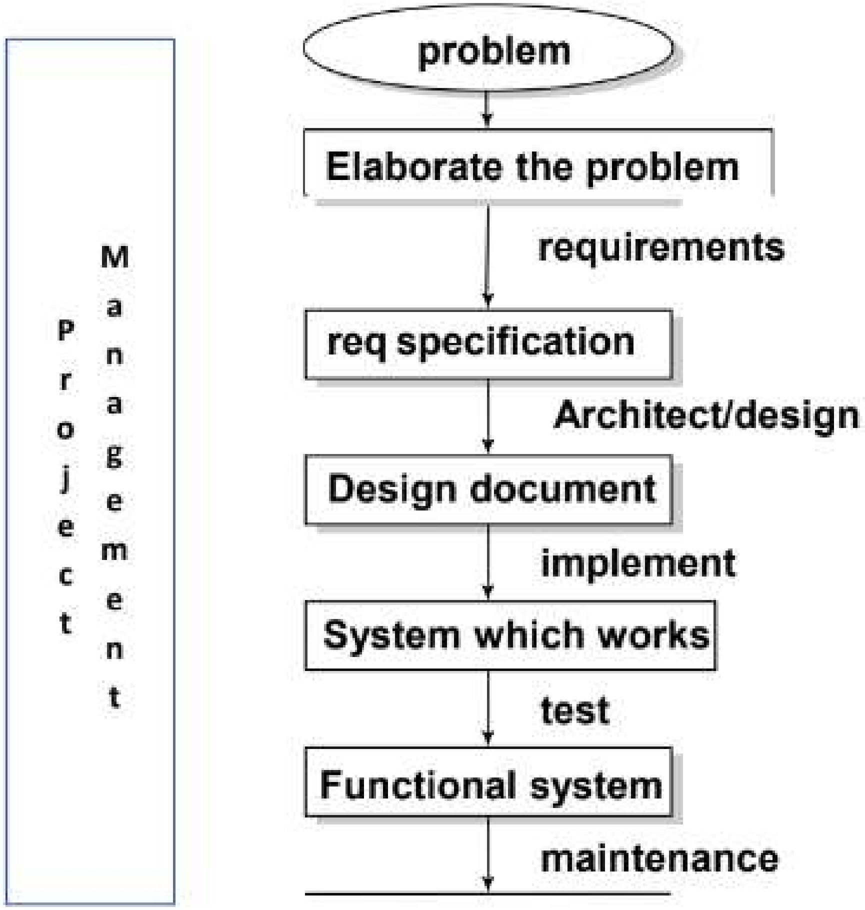
Software Process (is also called Lifecycle or process model or lifecycle model)

- 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.



Software Lifecycle

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: Time schedule

Software Process

Structure allows us to examine, understand, control and improve the activities in the process.

Other relevant processes such as configuration management process, change management process are a part of software development process.

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering
jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

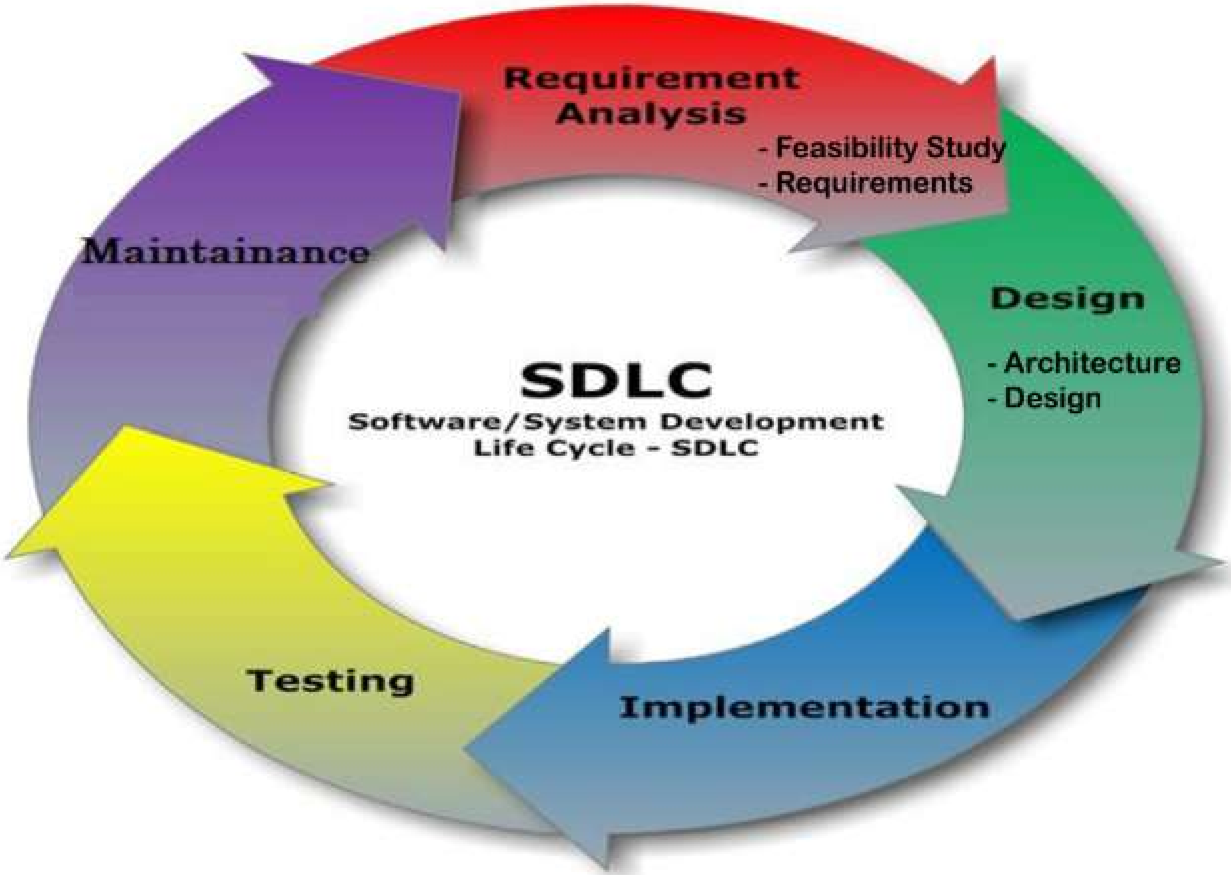
Software Engineering

SDLC, PDLC, PMLC, SMLC, Product Lifecycle

Dr. Jayashree R

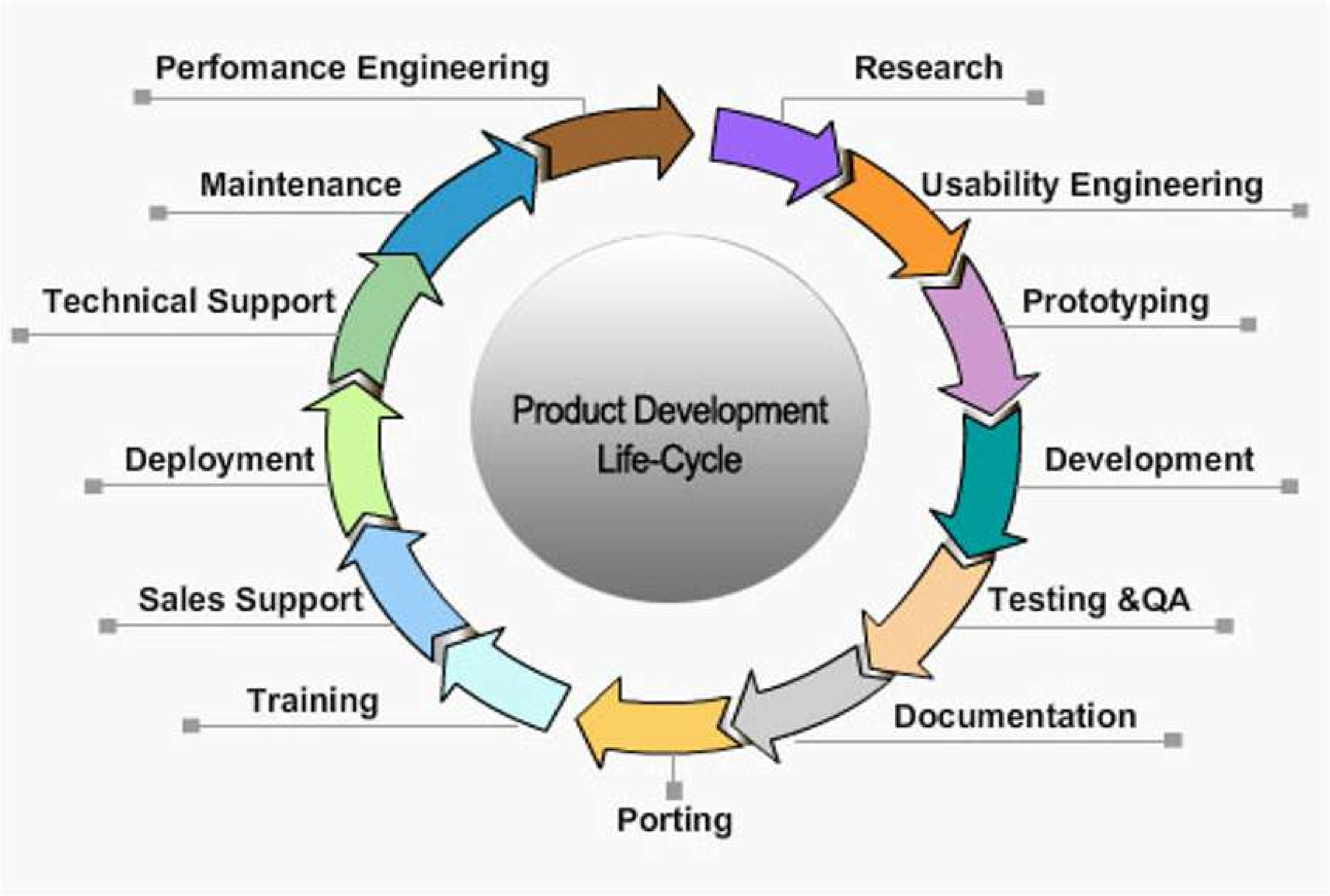
Department of Computer Science and Engineering

Software Development Lifecycle – SDLC



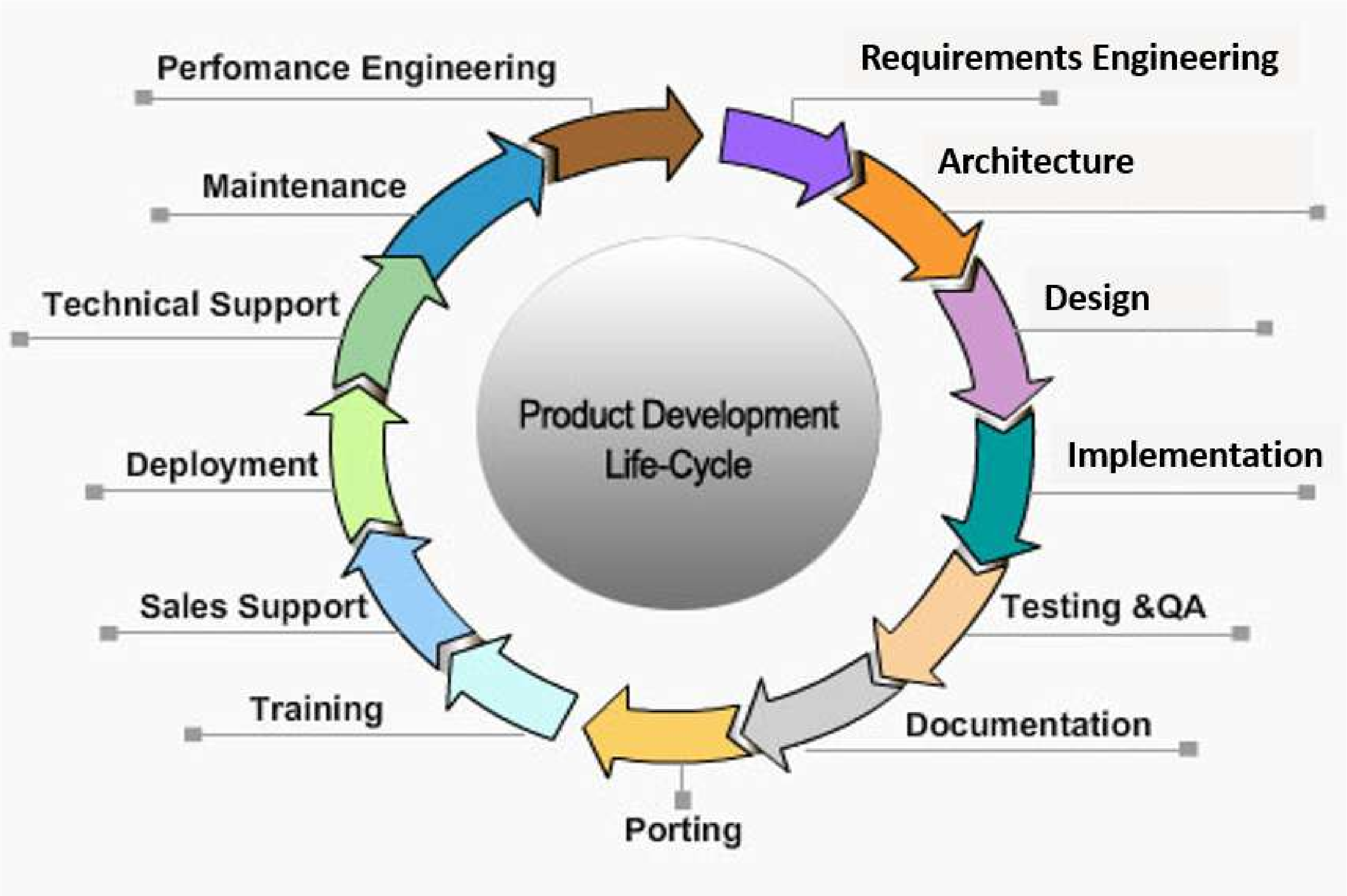
Product Development Lifecycle – PDLC

Variation 1



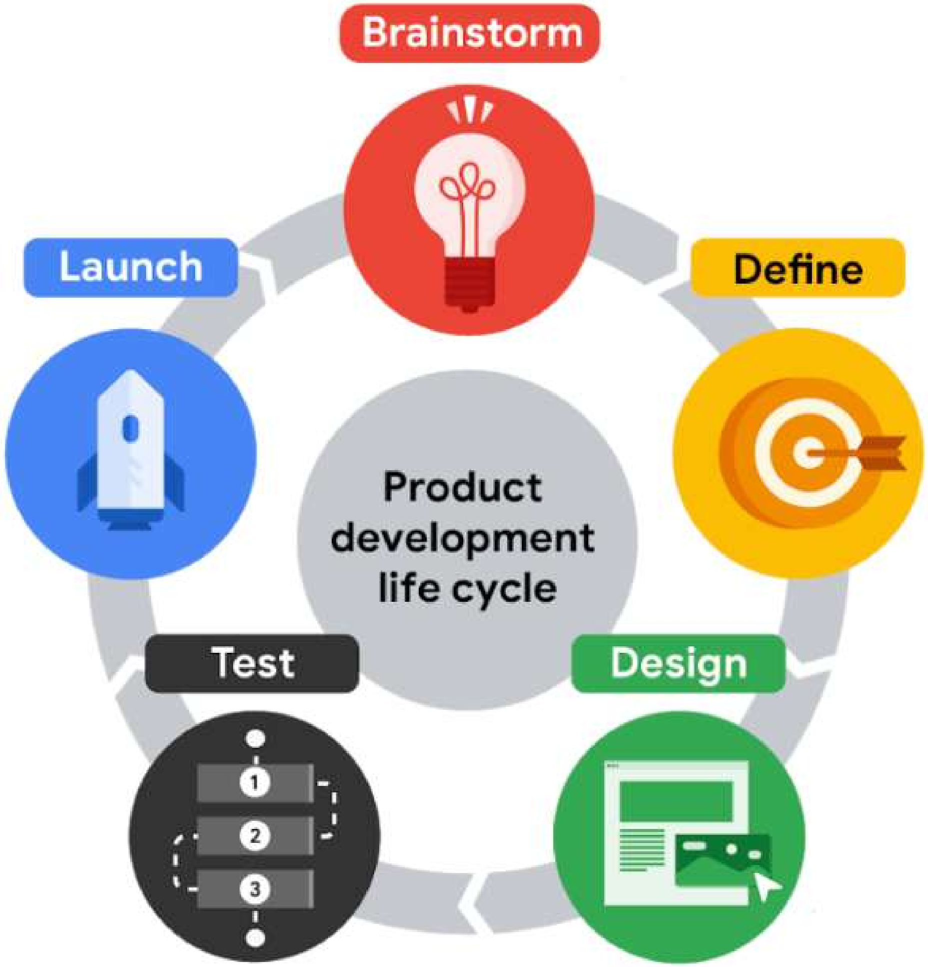
Product Development Lifecycle – PDLC

Variation 2



Product Development Lifecycle – PDLC

Variation 3



Brainstorm stage is when the team starts thinking of an idea for a product.

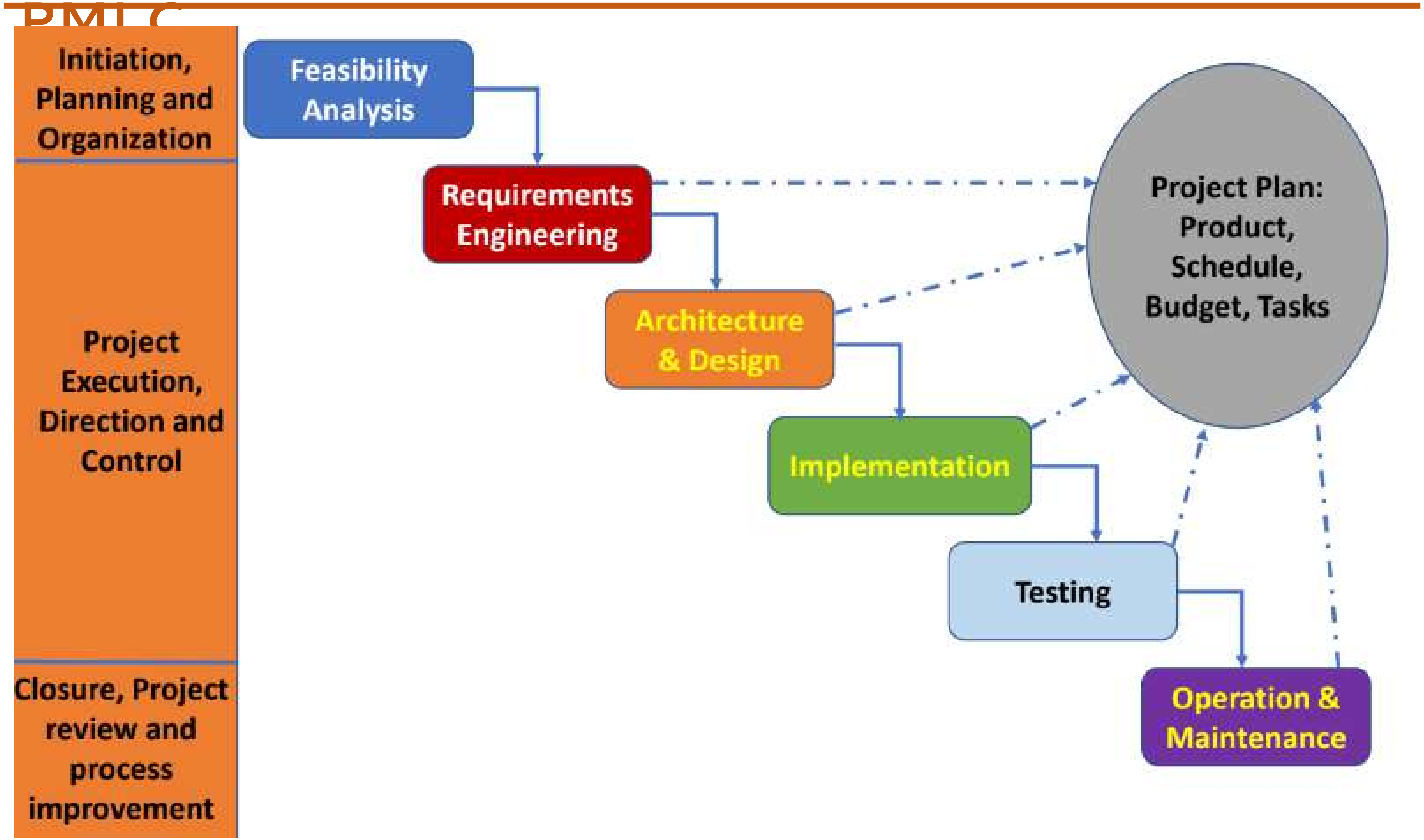
In the **Define** stage the goal is to figure out the specifications for the product by answering questions like: Who is the product for? What will the product do? And, what features need to be included for the product to be successful?

In the **Design** stage you start by drawing wireframes, which are outlines or sketches of the product, then move on to creating prototypes, which are early models of a product that convey its functionality.

The **Test** stage means writing the code and finalizing the overall structure of the product.

The **launch** stage is when the product is released into the world.

Project Management in System Development Lifecycle – PMLC



Initiation, Planning and Organization

Project Execution, Direction and Control

Closure, Project review and process improvement

Feasibility Analysis

Requirements Engineering

Architecture & Design

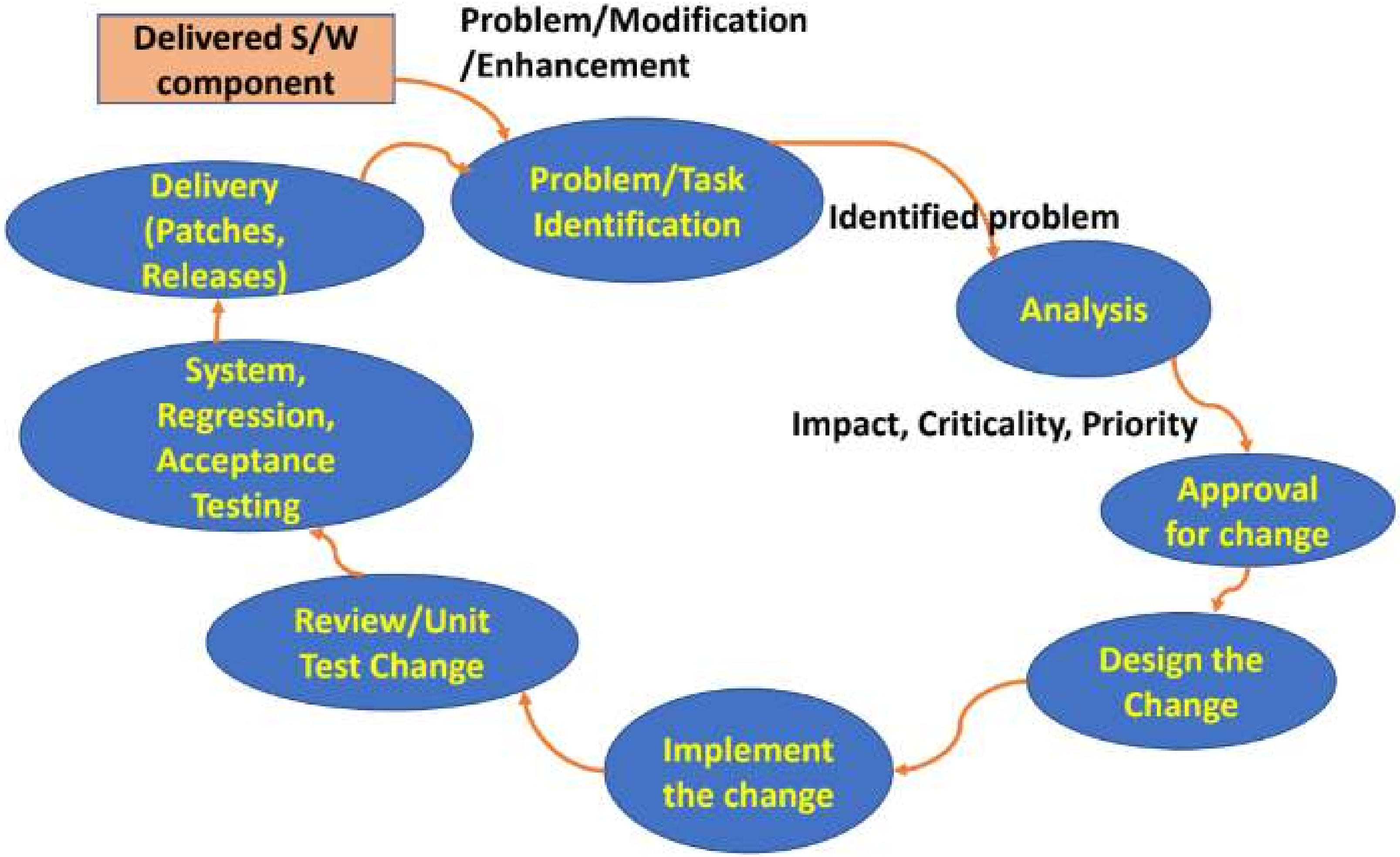
Implementation

Testing

Operation & Maintenance

Project Plan: Product, Schedule, Budget, Tasks

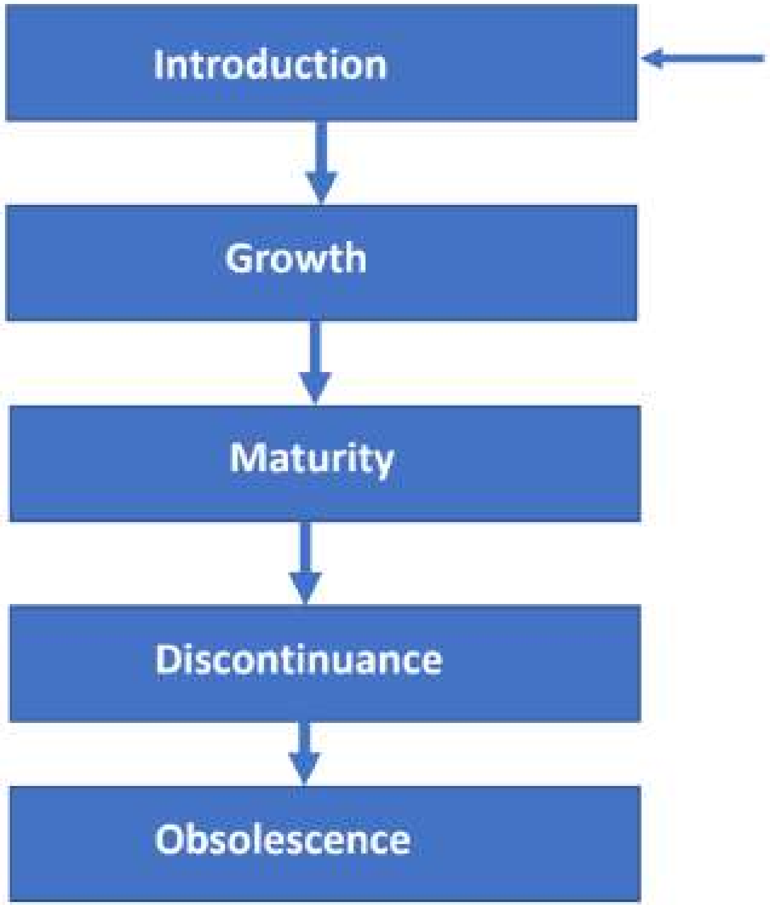
Software Maintenance Lifecycle – SMLC


Delivered S/W component
Problem/Modification /Enhancement
Problem/Task Identification
Identified problem
Analysis
Impact, Criticality, Priority
Approval for change
Design the Change
Implement the change
Review/Unit Test Change
System, Regression, Acceptance Testing
Delivery (Patches, Releases)

Product Lifecycle

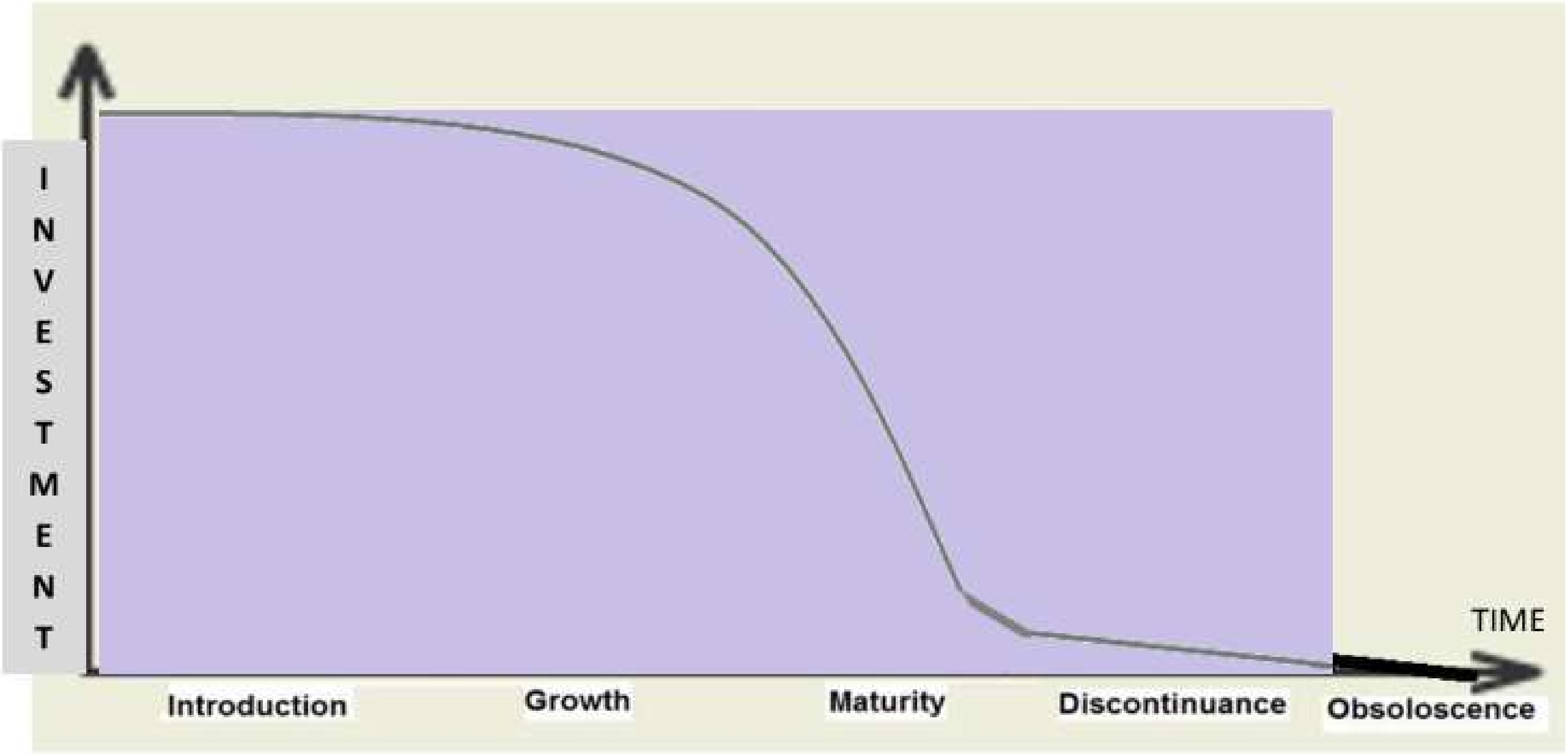
Attributes to Consider

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



Product built using a SDLC is initially released into market

Product Lifecycle Characteristics



THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Apoorva B S (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

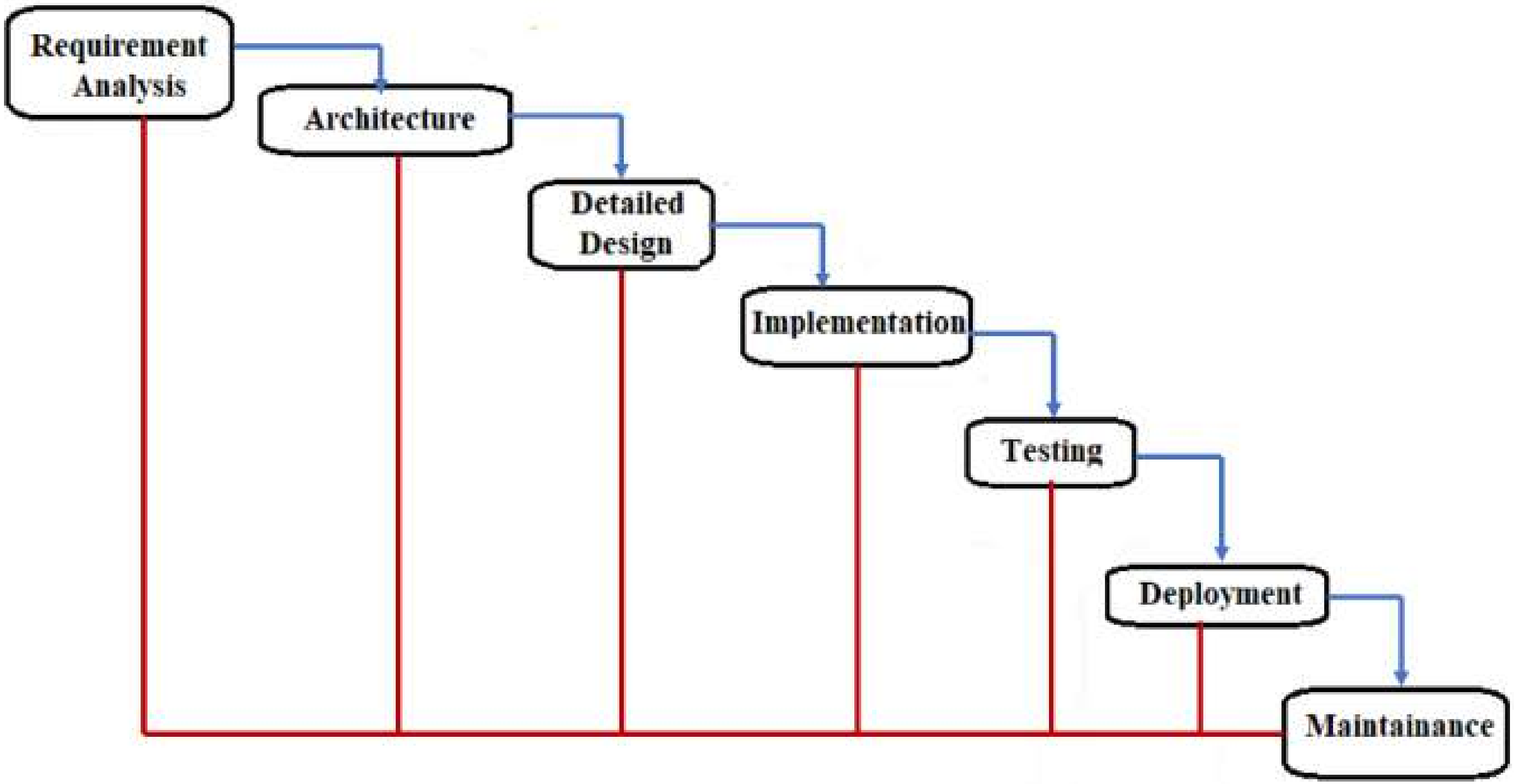
Software Engineering

Legacy SDLCs – Waterfall, V, Prototype, Incremental & Iterative Models

Dr. Jayashree R

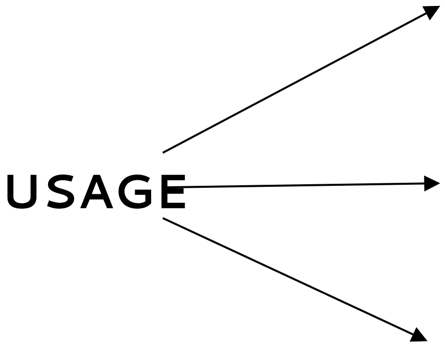
Department of Computer Science and Engineering

Waterfall model



Waterfall model – Advantages, Disadvantages & Usage

ADVANTAGES	DISADVANTAGES
Simple	Assumes requirements are frozen
Clear identified phases	Difficult to change & sequential
Easy to manage due to rigidity	Poor model for long projects
Each phase – specific deliverables + reviews	Big Bang approach
Easy to departmentalize and control	High risk + Uncertainty

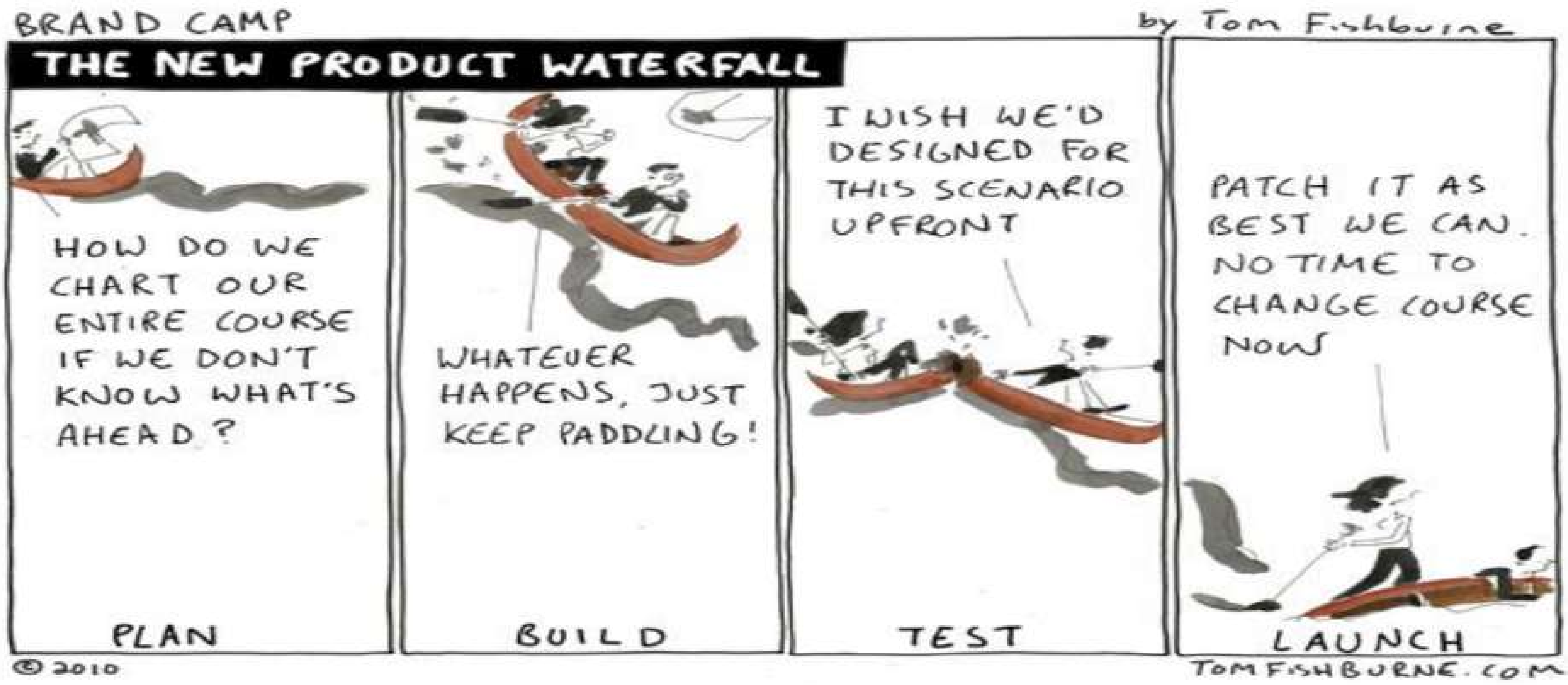

USAGE

Pure form: Short projects where requirements are well known

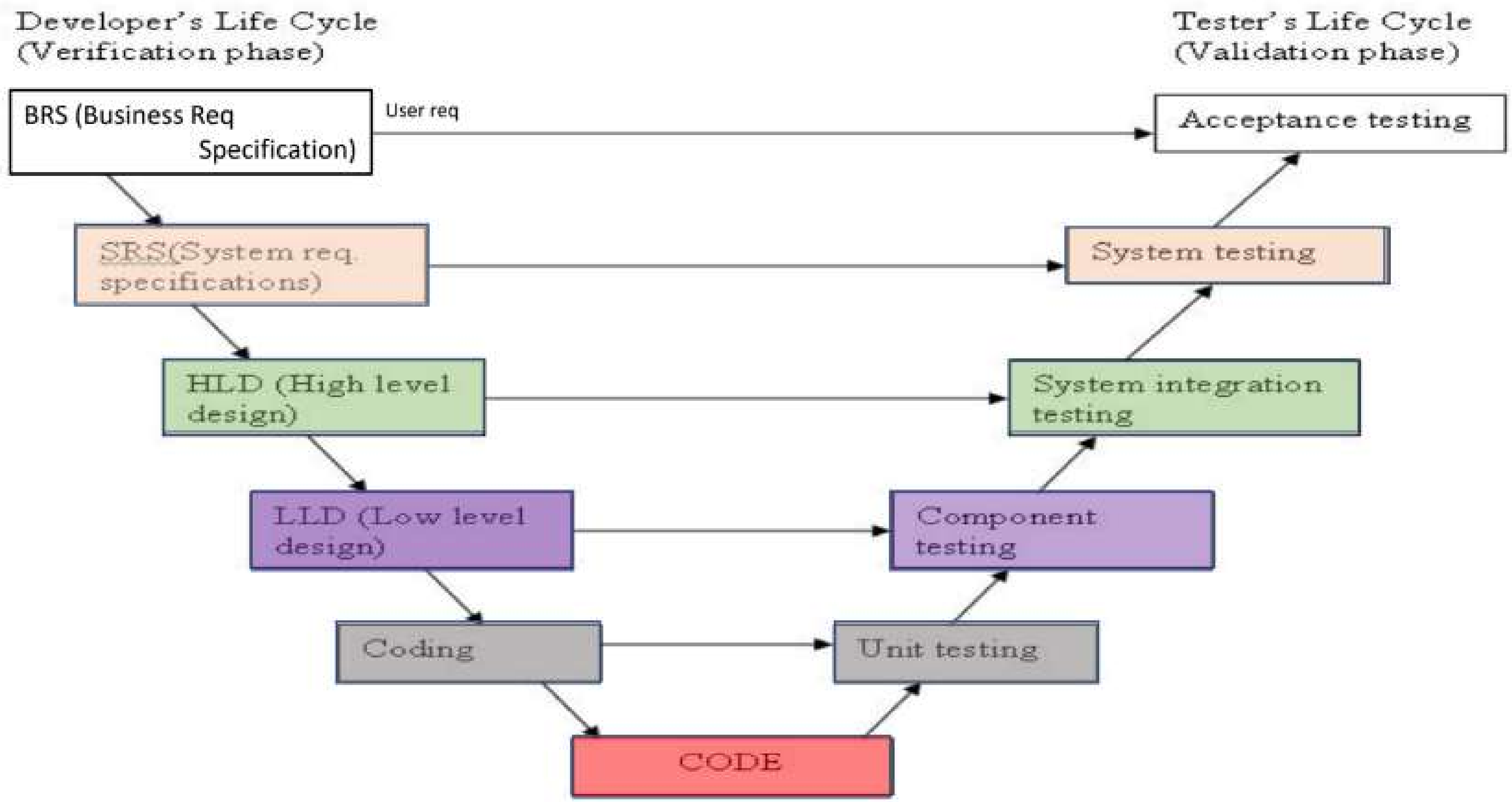
Product definition is stable & technology is understood

Variant form: High level in long projects

Waterfall model


BRAND CAMP
by Tom Fishburne
THE NEW PRODUCT WATERFALL
HOW DO WE CHART OUR ENTIRE COURSE IF WE DON'T KNOW WHAT'S AHEAD?
PLAN
WHATEVER HAPPENS, JUST KEEP PADDLING!
BUILD
I WISH WE'D DESIGNED FOR THIS SCENARIO UPFRONT
TEST
PATCH IT AS BEST WE CAN. NO TIME TO CHANGE COURSE NOW
LAUNCH
© 2010
TOMFISHBURNE.COM

V model

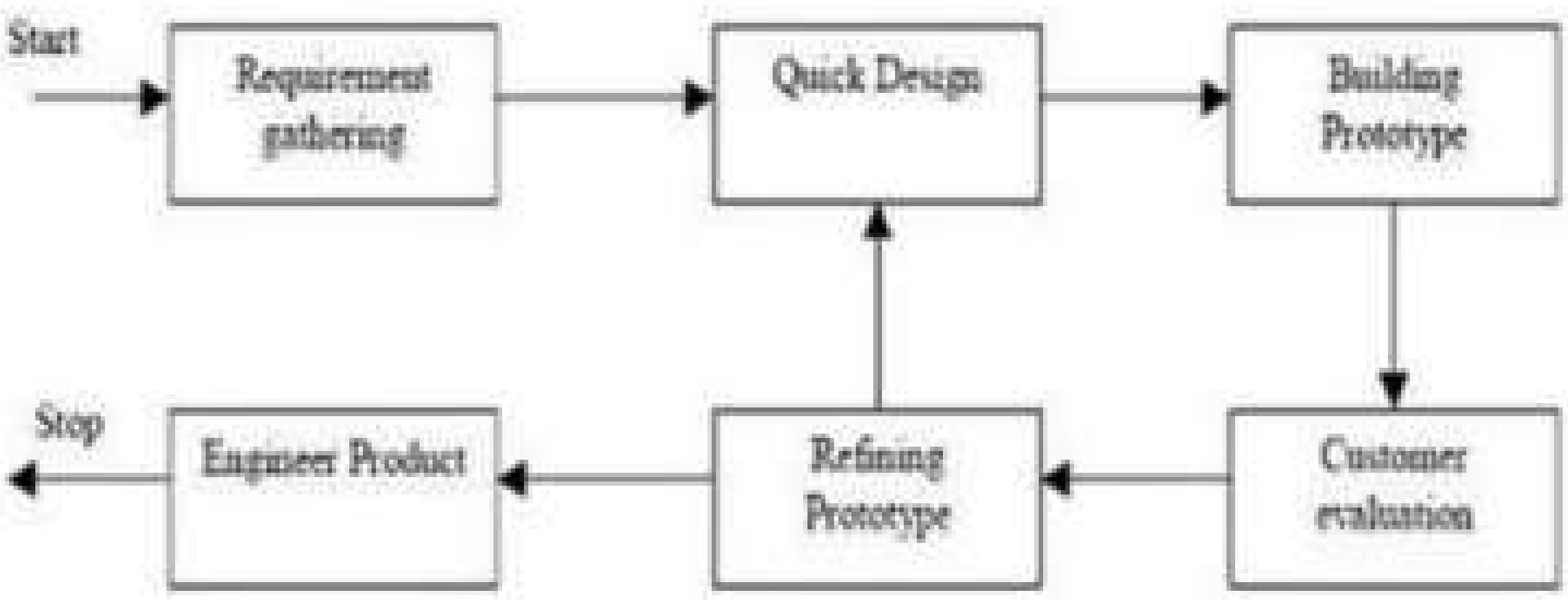

Developer's Life Cycle
(Verification phase)
Tester's Life Cycle
(Validation phase)
BRS (Business Req Specification)
User req
Acceptance testing
SRS(System req. specifications)
System testing
HLD (High level design)
System integration testing
LLD (Low level design)
Component testing
Coding
Unit testing
CODE

V model – Advantages, Disadvantages & Usage

ADVANTAGES	DISADVANTAGES
Similar to Waterfall model	Similar to Waterfall model
Test development activities can happen before formal testing cycle	No early prototypes of software
Higher probability of success + Increased effectiveness of usage of resources	Change in process => change in test documentation

USAGE: Similar to Waterfall model

Prototype model



- Cheap
- Entire system prototype is built to understand the requirements
- Types: Throw-away and Evolutionary

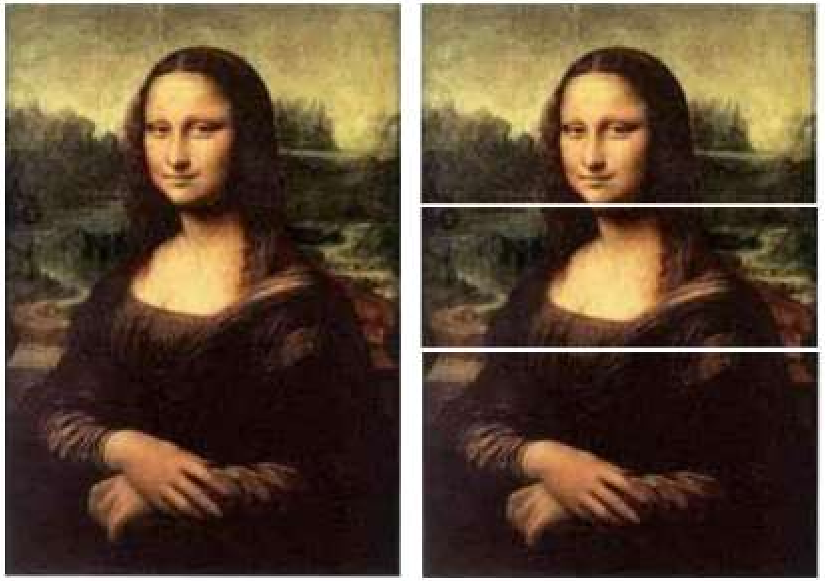
Prototype model – Advantages, Disadvantages & Usage

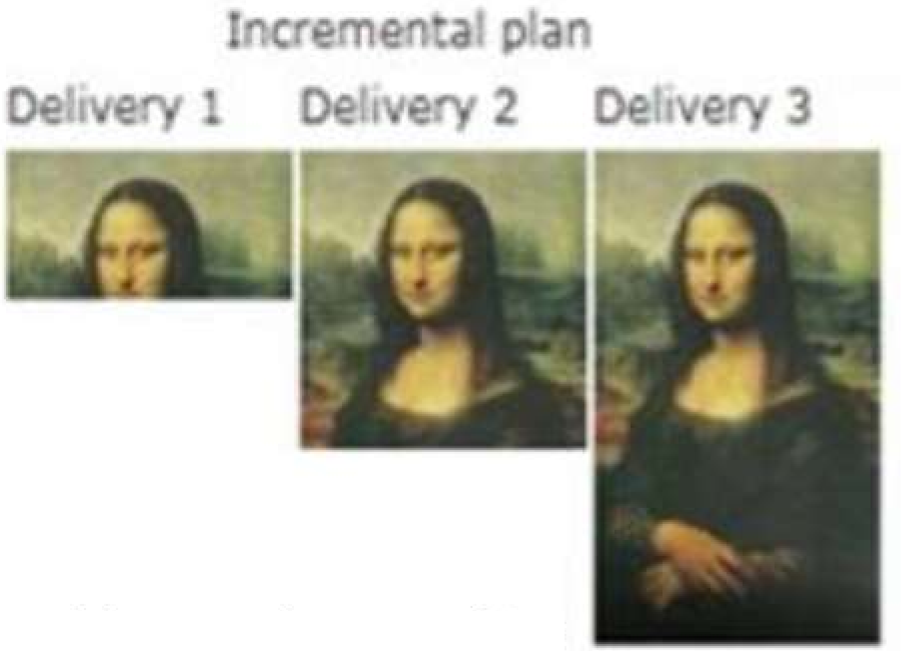
ADVANTAGES	DISADVANTAGES
Active involvement of users	May increase complexity of system as scope of system may expand beyond original plans
Better risk mitigation, Reduced time and cost, Resulting system is full featured, More stable system	Performance of resulting system may not be optimal

USAGE:

- When requirements are not clear
- Users are actively involved

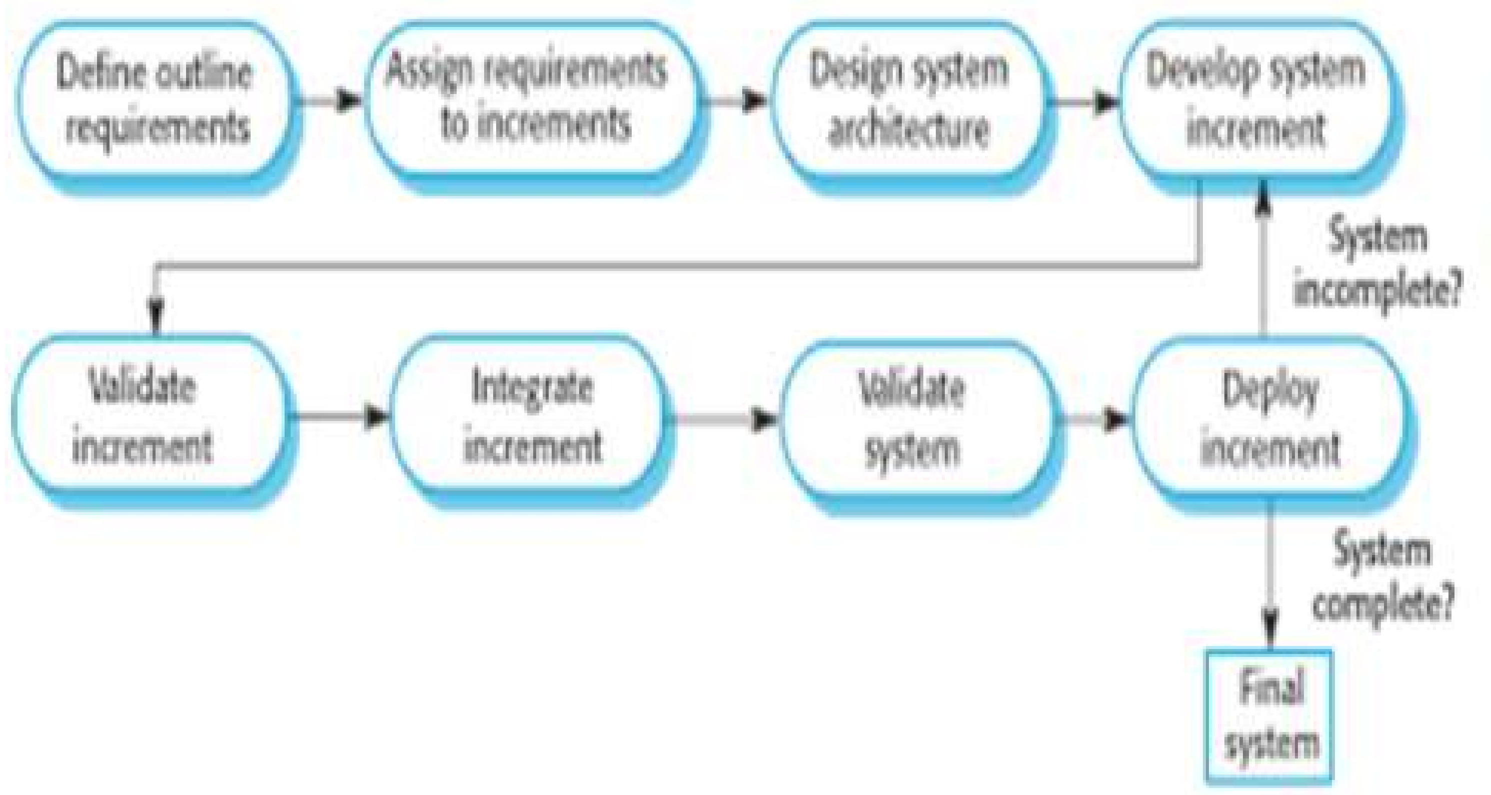
Incremental model





- Requirements are partitioned
- Working software in first module
- Each subsequent release adds functionality to previous module
- Continuous integration is done until entire system is achieved

Incremental model



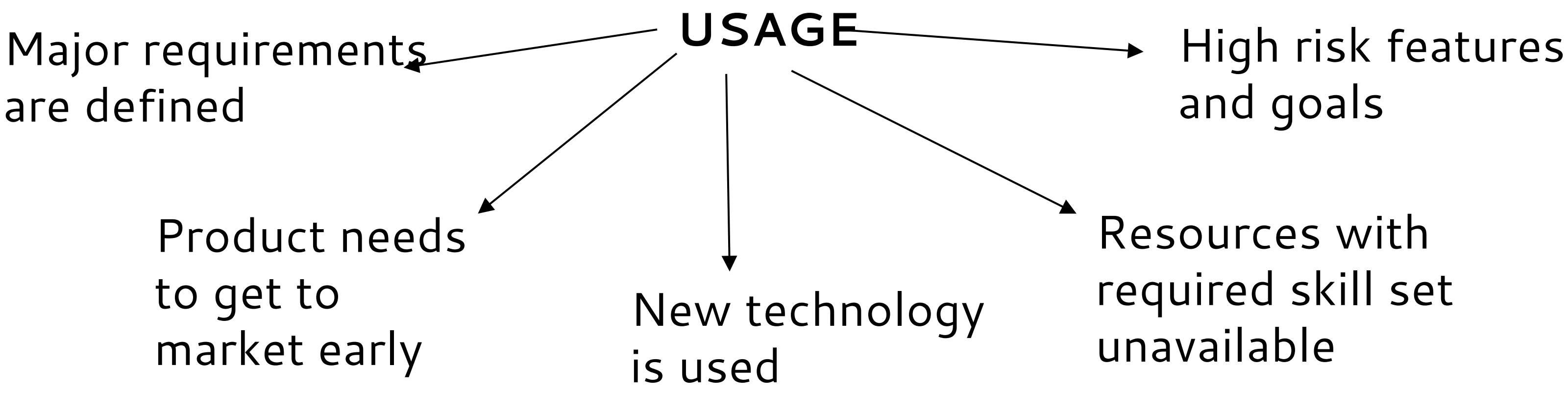
- Partitioned requirements can have a development lifecycle
- Models like waterfall can be used for each partition

FUN FACT

Leveraged prototype for additional features would be a form of incremental model

Incremental model – Advantages, Disadvantages & Usage

ADVANTAGES	DISADVANTAGES
Customer value and more flexible	Needs good planning and design
Easier to test and debug	Needs clear and complete definition of whole system
Easier to manage risk	Total cost is higher than waterfall
Continuous increments rather than monolithic	Hard to identify common functionalities across increments
Reduces over functionality	Management visibility is reduced



USAGE

Major requirements are defined

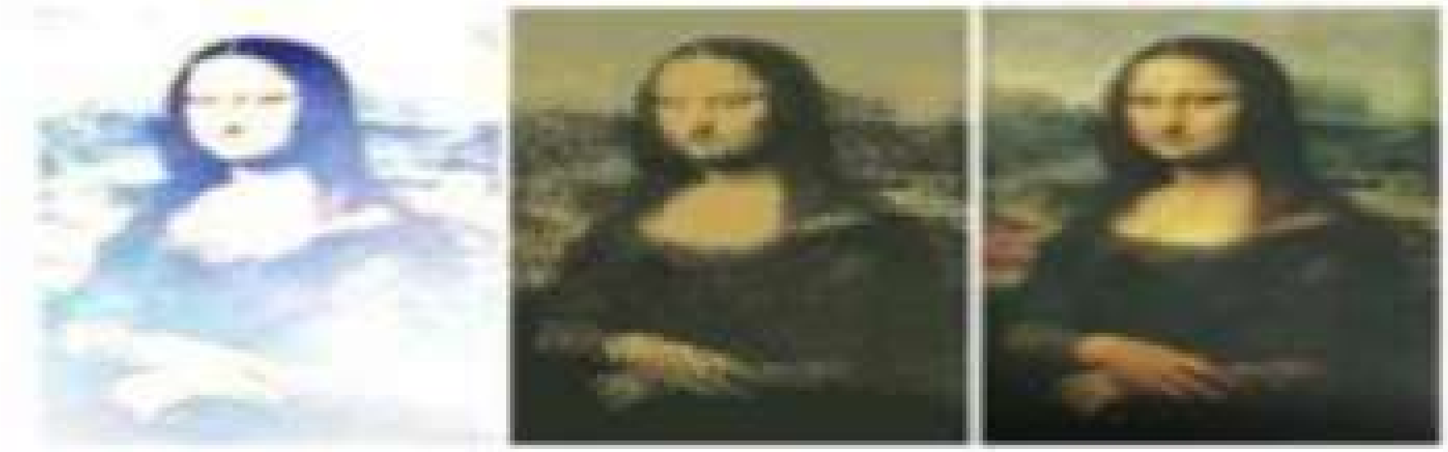
High risk features and goals

Product needs to get to market early

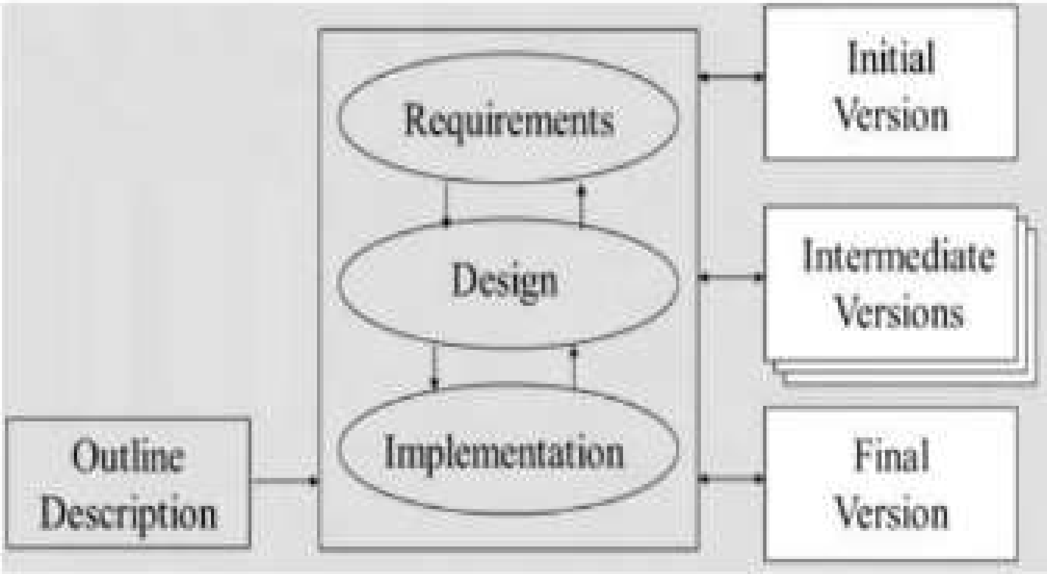
New technology is used

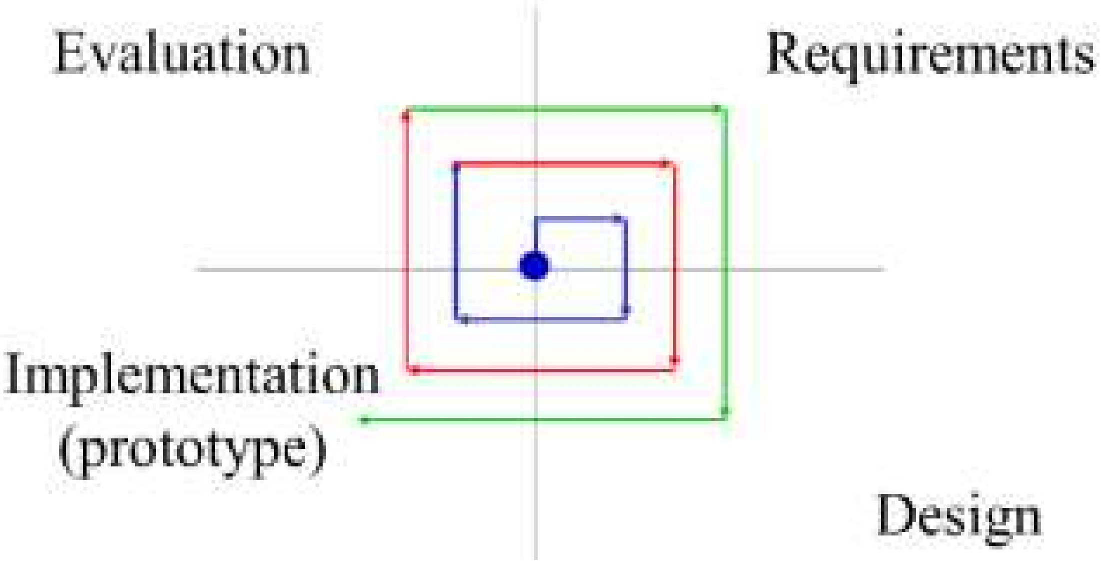
Resources with required skill set unavailable

Iterative model (Evolutionary)



- Initial implementation starts from a skeleton of product
- This is followed by refinement through user feedback & evolution
- Built with dummy modules
- Rapid prototyping
- Successive refinement





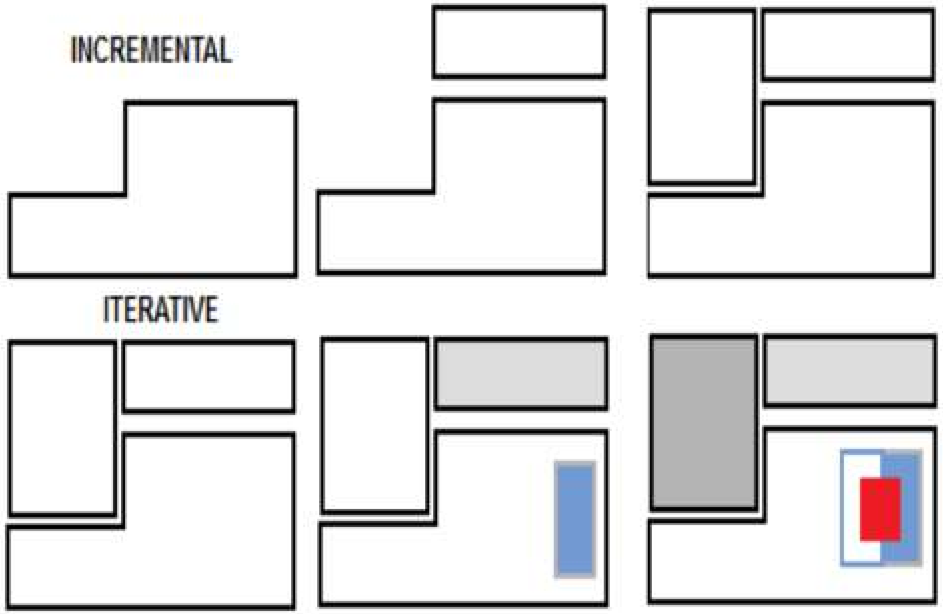
Iterative model – Advantages, Disadvantages & Usage

ADVANTAGES	DISADVANTAGES
Help identify requirement & solution visualization	Each phase is rigid with overlaps
Support risk mitigation, rework is reduced, incremental investment, feature creep, increased customer engagement	Costly system architecture may arise

USAGE: Large projects which may get extended

Comparison: Iterative model vs Incremental model

ITERATIVE MODEL	INCREMENTAL MODEL
Revisit and refine every thing	No need to go back and change delivered things
Focus on details of things	Focus on things not implemented yet
Leverage on learnings	Does not leverage on experience or knowledge



THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering
jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R
Department of Computer Science and Engineering

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

Software Engineering

Agile Philosophy

Dr. Jayashree R

Department of Computer Science and Engineering

Limitations of most legacy models

Can you think of some limitations of legacy

Predictive software development methods

Upfront planning

Do not facilitate periodic customer interaction

Suited for very large complex projects

Regulatory perspectives

Suited for global or distributed organizations

Product lifecycle and its eco system

People and skill perspective

Suitable for projects with clear definition

Suitable when things are not changing fast

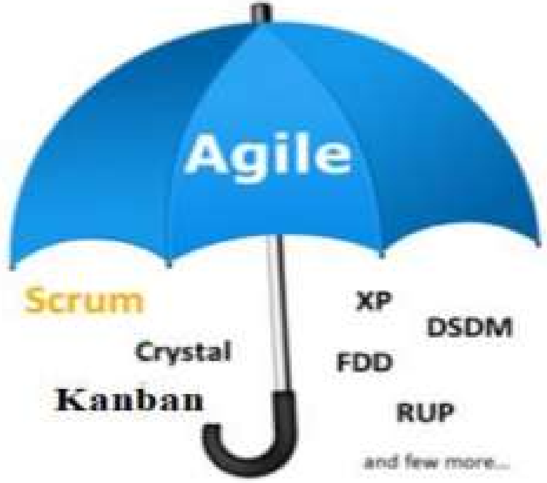
Agile Philosophy

Agile is an umbrella term used to describe a variety of methods.

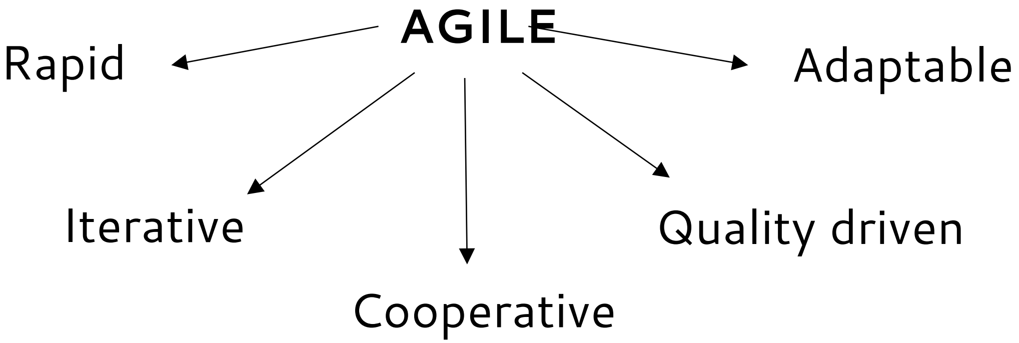
These methods encourage

Continual realignment of development goals with needs and expectations of the customer

Reducing massive planning overhead to allow fast reactions to change



Agile is not a process. It is a set of values or a philosophy.



AGILE

Rapid

Adaptable

Iterative

Cooperative

Quality driven

Agile Manifesto

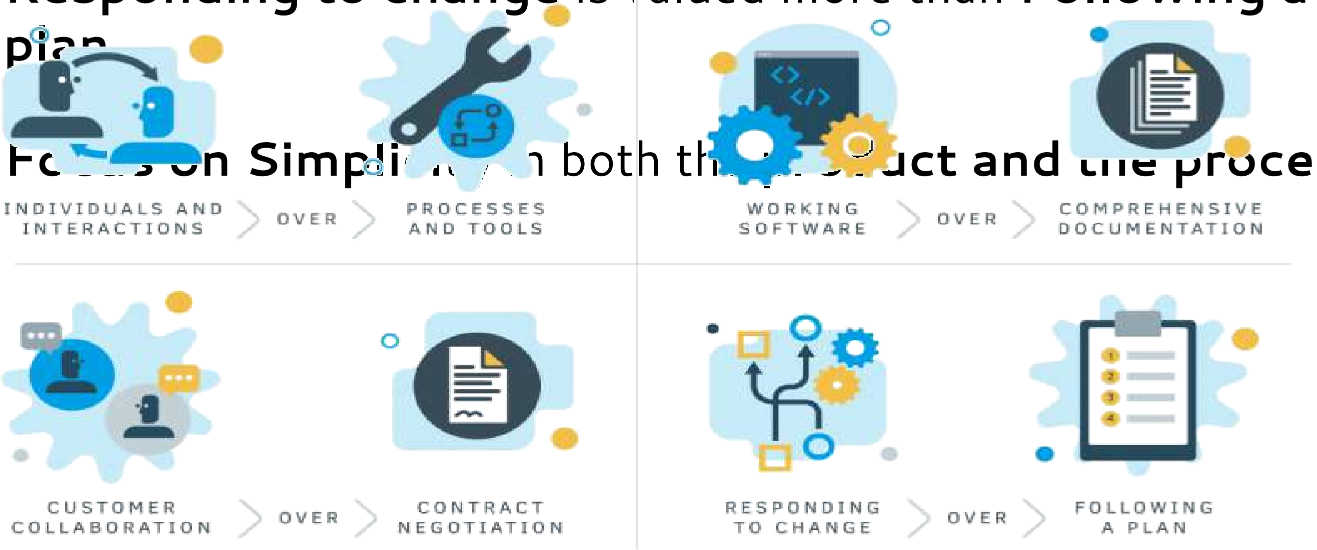
- **Individuals and interactions** are valued more than **Processes and tools**

- **Working software** is valued more than **Comprehensive documentation**

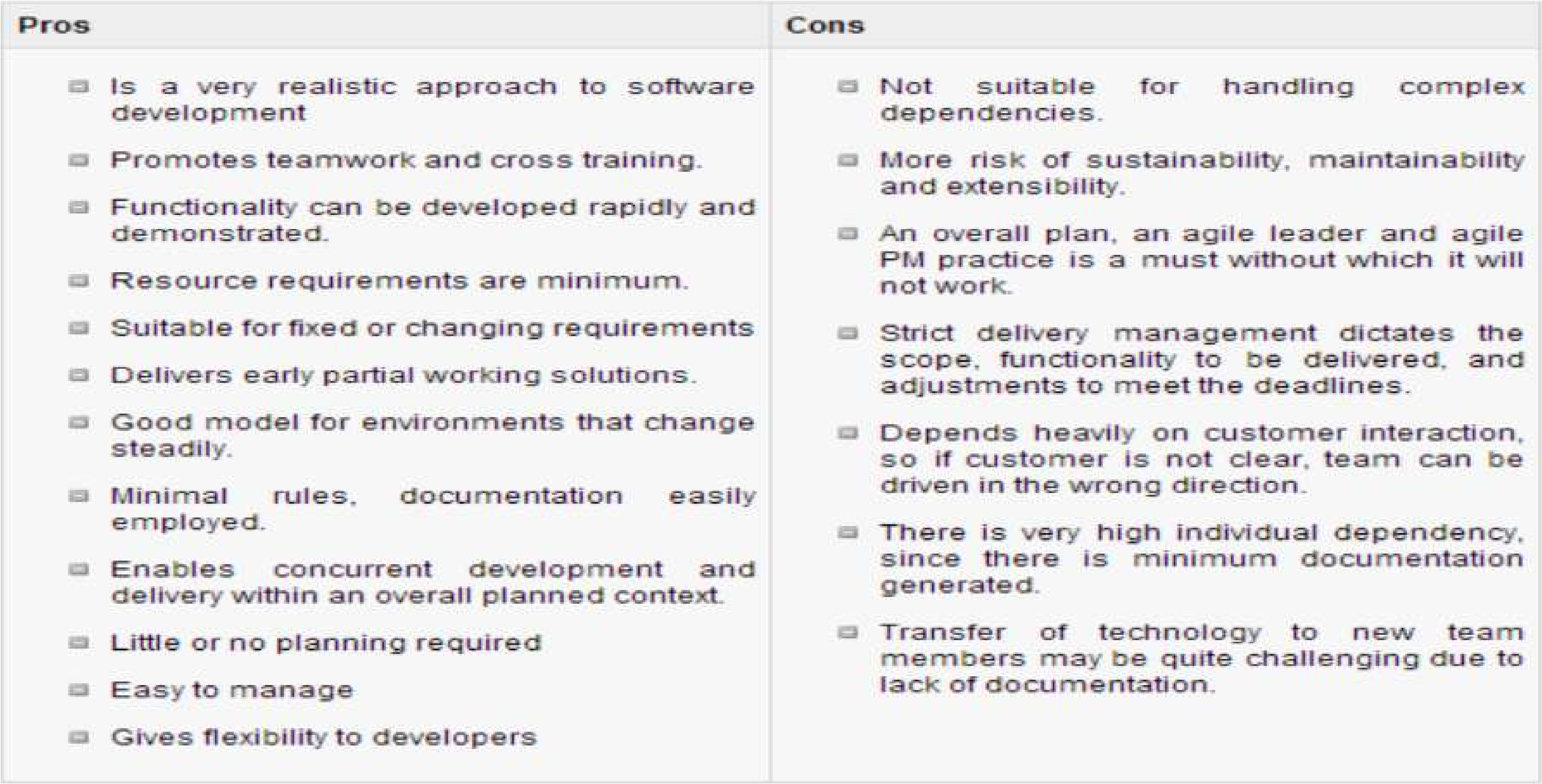
- **Customer collaboration** is valued more than **Contract negotiation**

- **Responding to change** is valued more than **Following a plan**

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

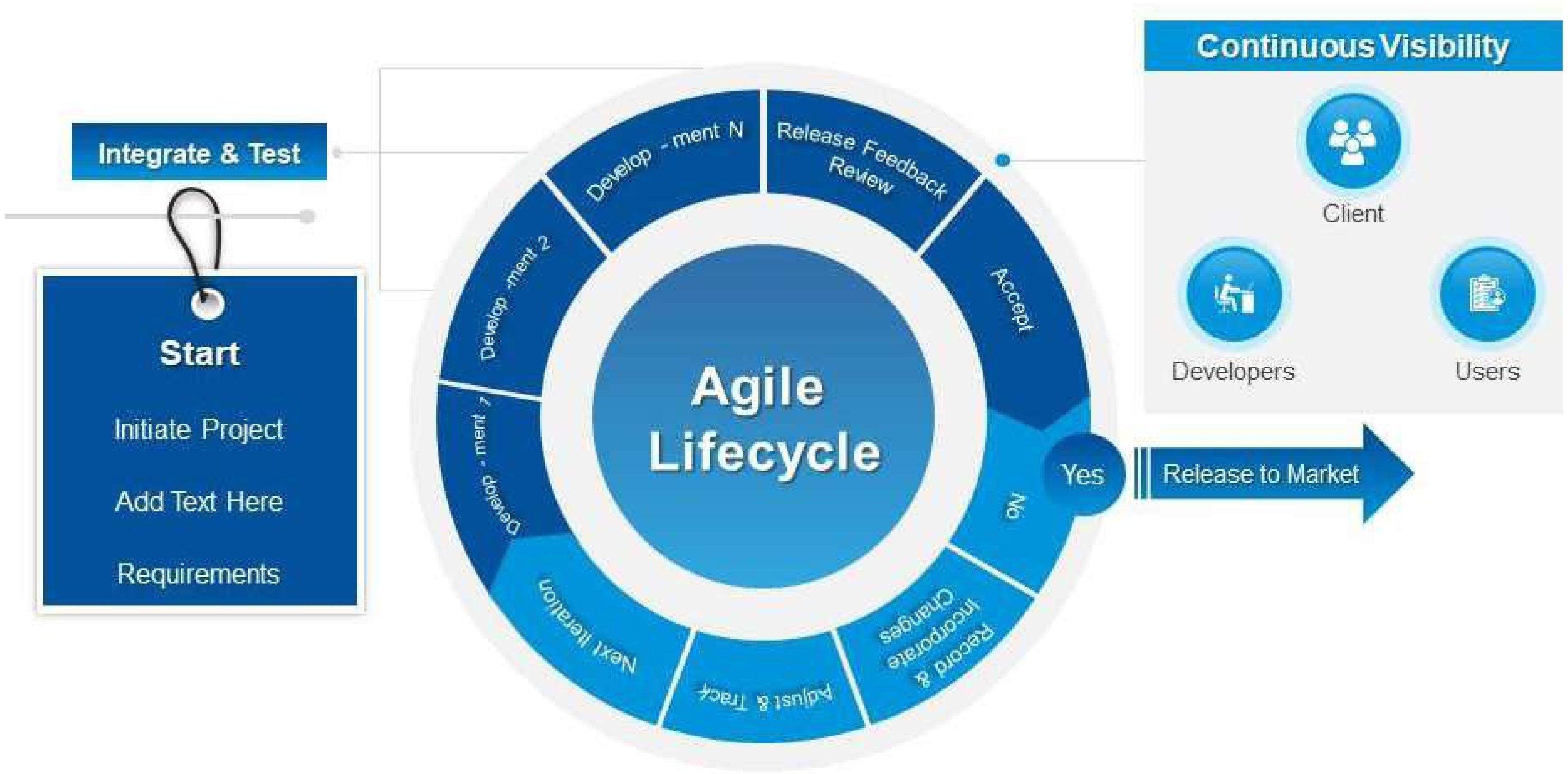


Pros & Cons of Agile methodologies



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.

What if SDLC is made Agile?



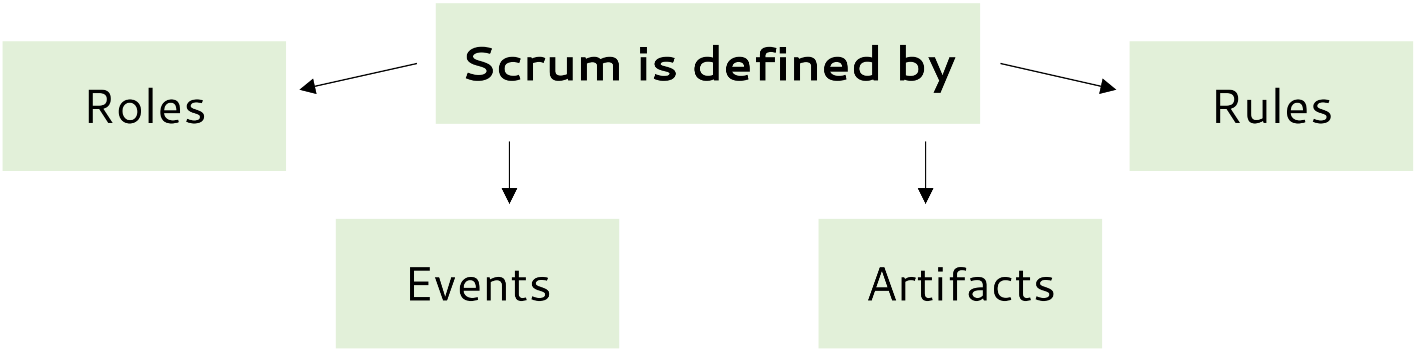
Agile methodologies: Scrum

Origin – Rugby

In rugby, everyone has a part and they huddle periodically to take stock

What is Scrum?

- Methodology/framework for developing, delivering and sustaining software components and products
- Iterative approach towards software development
- Provides mechanisms to apply agile practices



Scrum is defined by

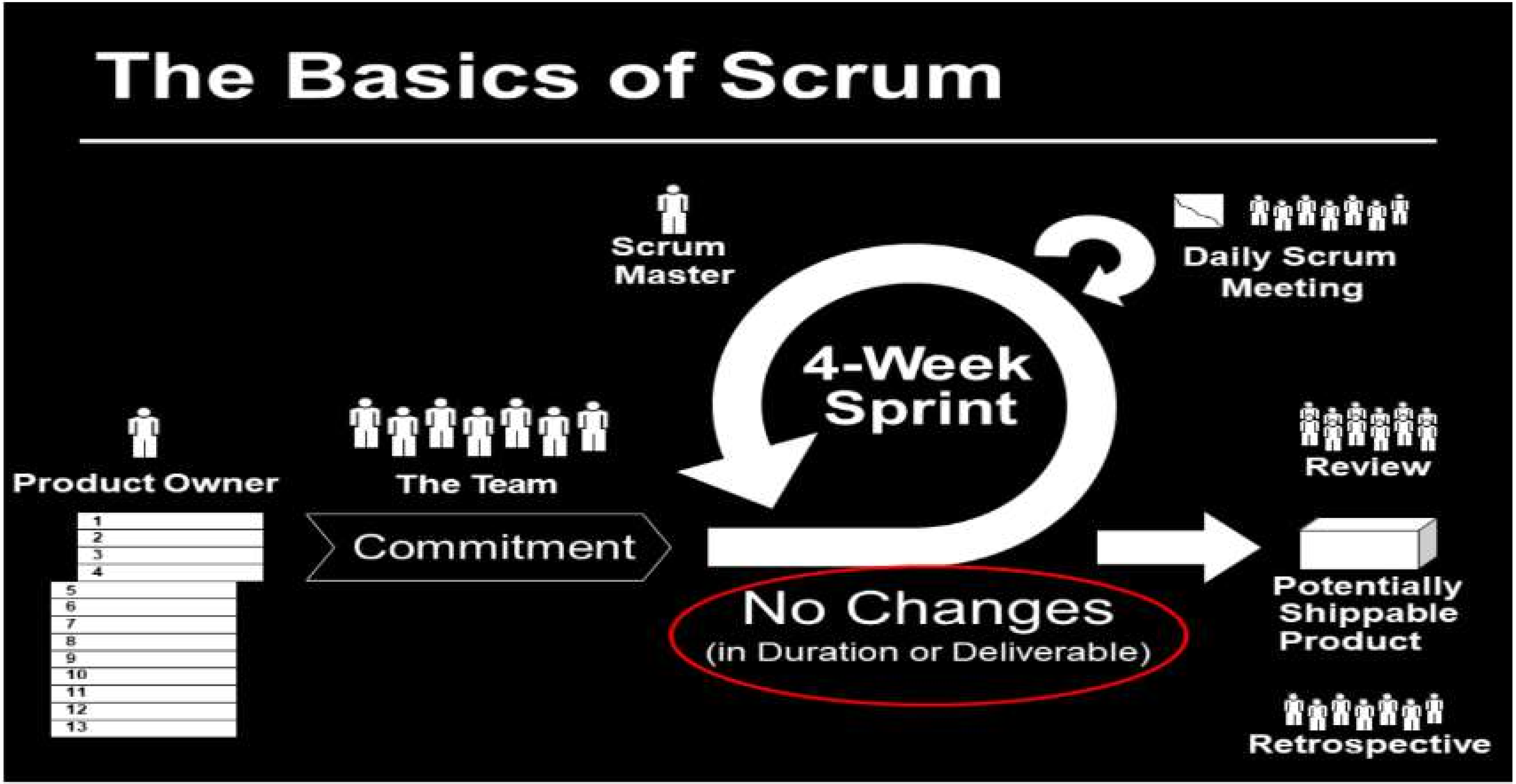
Roles

Rules

Events

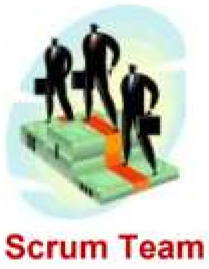
Artifacts

Agile methodologies – Scrum



Agile methodologies – Scrum Roles

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



- Cross functional and self organizing
- Consists of contributors to deliverable
- Responsible for delivering shippable increments

SCRUM

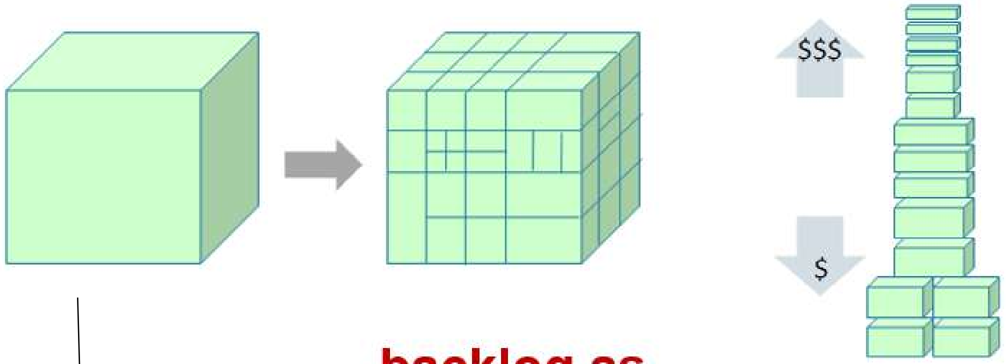
- Is a facilitator (Not a manager!)
- Removes impediments & facilitates meetings
- Ensures team sticks to scrum theory and practices

PROJECT OWNER

- Voice of stakeholder/team
- Creates or manages product backlog

Agile methodologies – Scrum

Artifacts



Product Backlog

backlog as user stories

Weighted and ranked stories

Results in *Roadmap*

What is Product Backlog?
The project/product is described as a list of features called Product Backlog.

What does Product Backlog include?

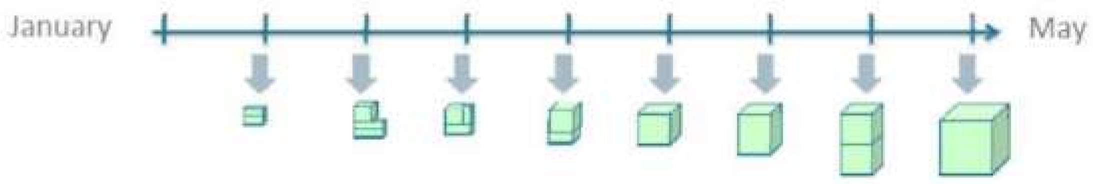
- New features
- Changes to existing features
- Bug fixes
- Infrastructure setups etc.

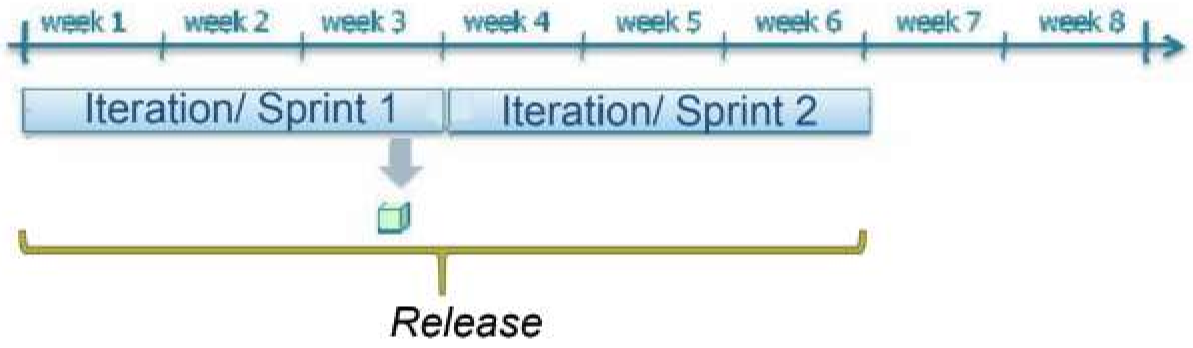
The stories are ranked by importance by estimating the amount of work needed to be done by **Scrum team**

Agile methodologies – Scrum Events

SPRINT

- Short fixed iterations (usually 2 – 4 weeks)
- Potentially shippable code demonstrated after each iteration
  (time boxing)





Total effort/ iteration => number of user stories/iteration

One release may contain multiple iterations

SPRINT PLANNING MEETING

- Used to determine which of the product backlog items will be worked on & delivered in the next iteration

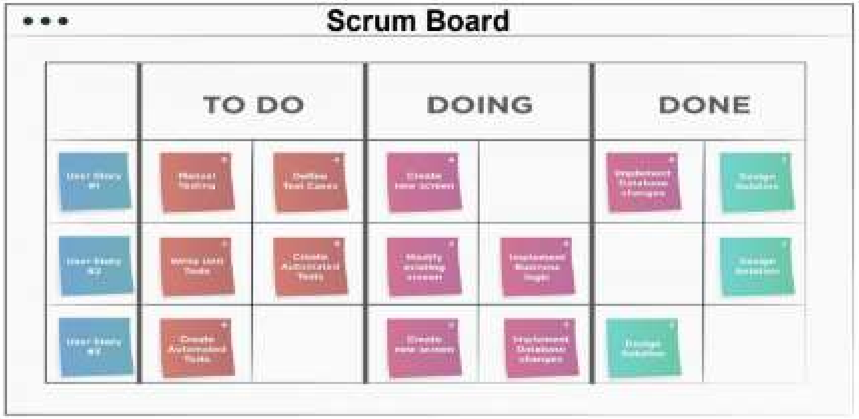
Agile methodologies – Scrum Events

Daily scrum meeting (also called stand up meeting). Things discussed in these meetings:

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

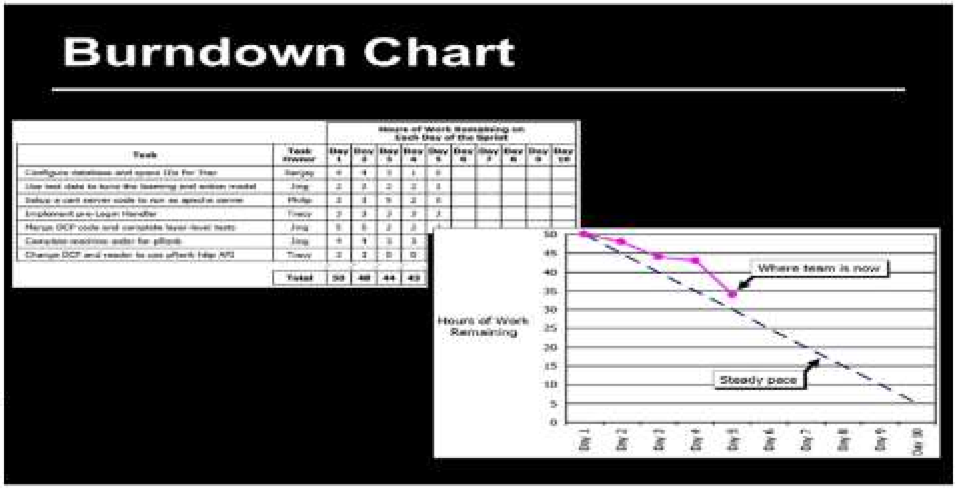
End of sprint deliverable – Shippable product

Updating the sprint backlog on Scrum board/Burndown chart



Scrum Board

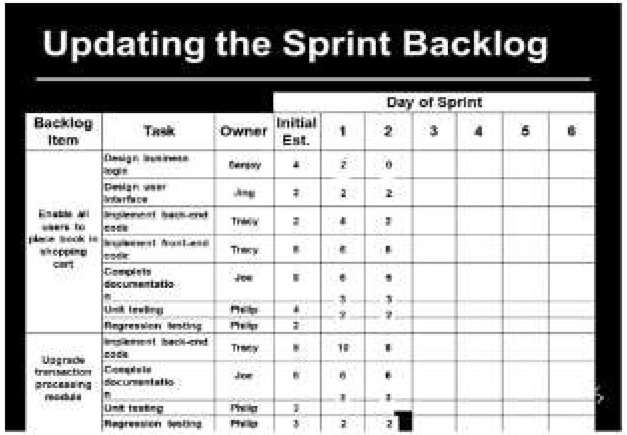
	TO DO		DOING		DONE	
User Story #1	Manual Testing	Define Test Cases	Create new screen		Implement Database changes	Design Solution
User Story #2	Write Unit Tests	Create Automated Tests	Modify existing screen	Implement Business logic		Design Solution
User Story #3	Create Automated Tests		Create new screen	Implement Database changes	Design Solution	



Burndown Chart

		Hours of Work Remaining on Each Day of the Sprint									
Task	Task Owner	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Configure database and space IDs for Trac	Sanjay	4	4	1	1	0					
Use test data to tune the learning and action model	Jing	2	2	2	2	1					
Setup a cart server code to run as apache server	Philip	2	3	5	2	0					
Implement pre-Login Handler	Tracy	2	3	2	2	2					
Merge DCP code and complete layer-level tests	Jing	5	5	2	2	1					
Complete machine order for pRank	Jing	4	4	3	3						
Change DCP and reader to use pRank http API	Tracy	2	2	0	0						
	Total	50	48	44	43						

Hours of Work Remaining
50
45
40
35
30
25
20
15
10
5
0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Where team is now
Steady pace



Updating the Sprint Backlog

				Day of Sprint					
Backlog Item	Task	Owner	Initial Est.	1	2	3	4	5	6
Enable all users to place book in shopping cart	Design business logic	Sanjay	4	2	0				
	Design user interface	Jing	2	2	2				
	Implement back-end code	Tracy	2	4	2				
	Implement front-end code	Tracy	8	6	8				
	Complete documentation	Joe	8	6 3	6 3				
	Unit testing	Philip	4	2	2				
	Regression testing	Philip	2						
Upgrade transaction processing module	Implement back-end code	Tracy	8	10	8				
	Complete documentation	Joe	8	6 3	6 3				
	Unit testing	Philip	3						
	Regression testing	Philip	3	2	2				

Agile methodologies – Sprint Pre Planning

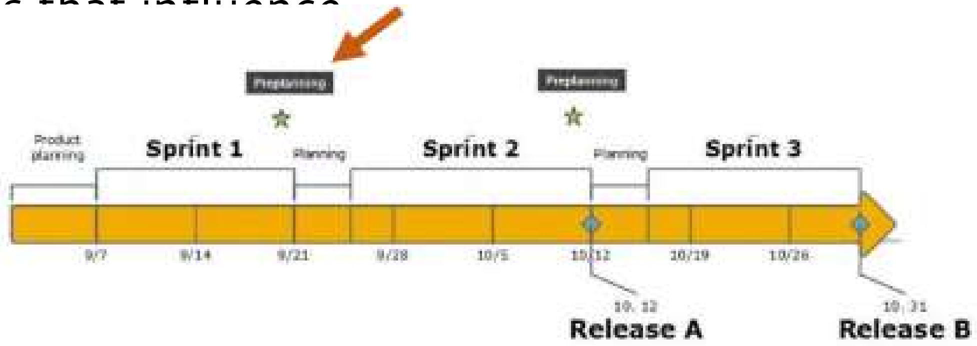
What is Pre Planning meeting?

A Pre-Planning meeting enables the business and stakeholders to focus on prioritization
and preparation of requirements far advance before the sprint planning session.

Why is Pre Planning meeting necessary?

- Every sprint will need to have one prioritized list of requirements (set by business represented by the product owner) so there can be a focus on, and deliver the most valuable and needed requirements in a very short time
- It is quite challenging to prepare the backlog for any upcoming sprint
- Product Owner needs to talk and align requirements of multiple stakeholders which is not easy as every stakeholder has their own priorities that influence the plans of other stakeholders

There can be one or a couple of more of these Pre Planning ses



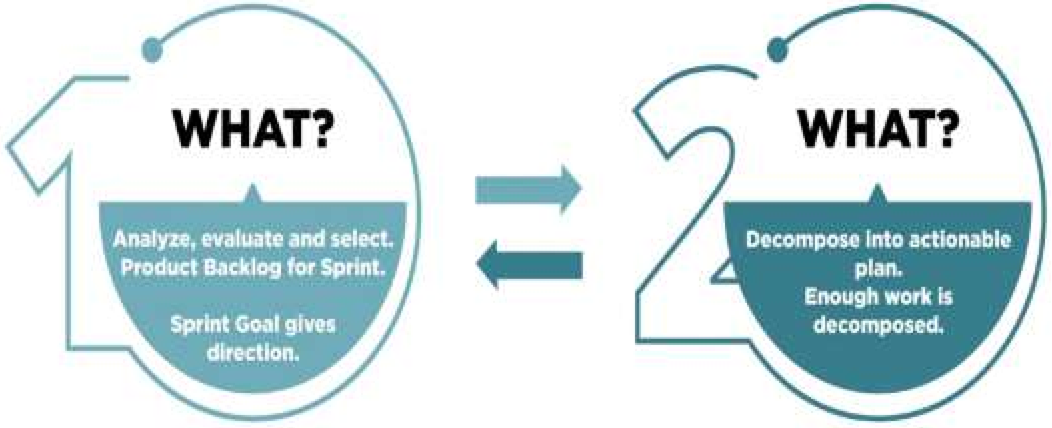
What are the outcomes of the Pre Planning meeting?

- Scope for the next sprint agreed.
- The readiness of requirements is indicated to the product owner
- Rough estimation in story-points

Agile methodologies – Sprint Planning

What is Sprint Planning?

- Event that kick starts the Sprint
- Agenda – define the scope of delivery and how to accomplish that work
- Sets a common goal for the team during the sprint


WHAT?
Analyze, evaluate and select. Product Backlog for Sprint.
Sprint Goal gives direction.
WHAT?
Decompose into actionable plan.
Enough work is decomposed.

Who attends the Sprint Planning?

Scrum Master	• Facilitate the Sprint planning meeting • Ensures agreement on the Sprint goal and product backlog items

Inputs to Sprint Planning

- Product backlog
- Sprint team capacity
- Past performance of Dev team

Activities of Sprint Planning

- Identify Sprint goal
- Choose user stories
- Plan for capacity

Agile methodologies – Sprint Review

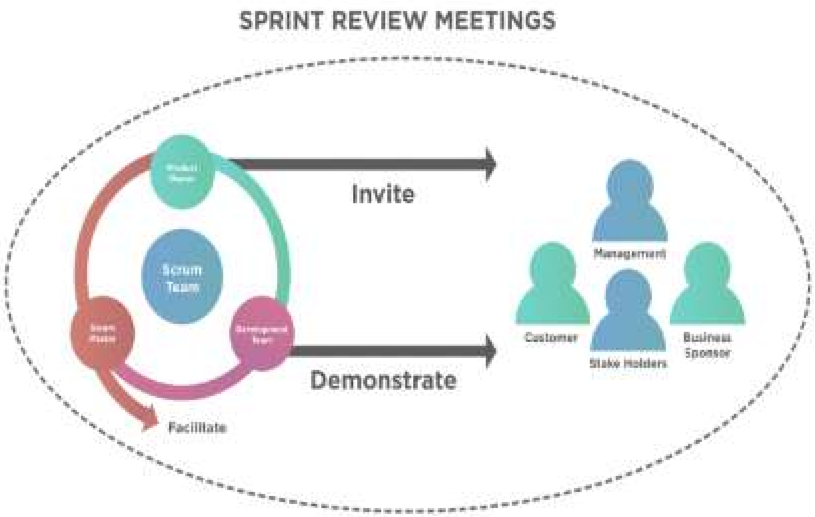
What is Sprint Review meeting?

It is used to demonstrate story features

What is done during Sprint Review meeting?

Product owner does the following:

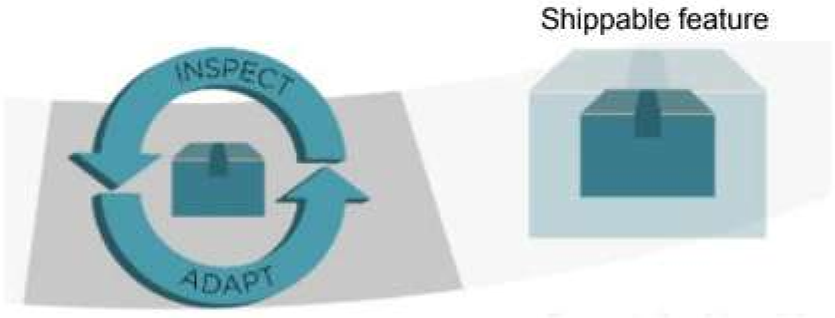
- Evaluates against preset criteria
- Gets feedback from clients and stakeholders
- Ensures the delivered increment meets the business need
- Helps support reprioritizing of the product backlog
- Optimize the release plan if needed



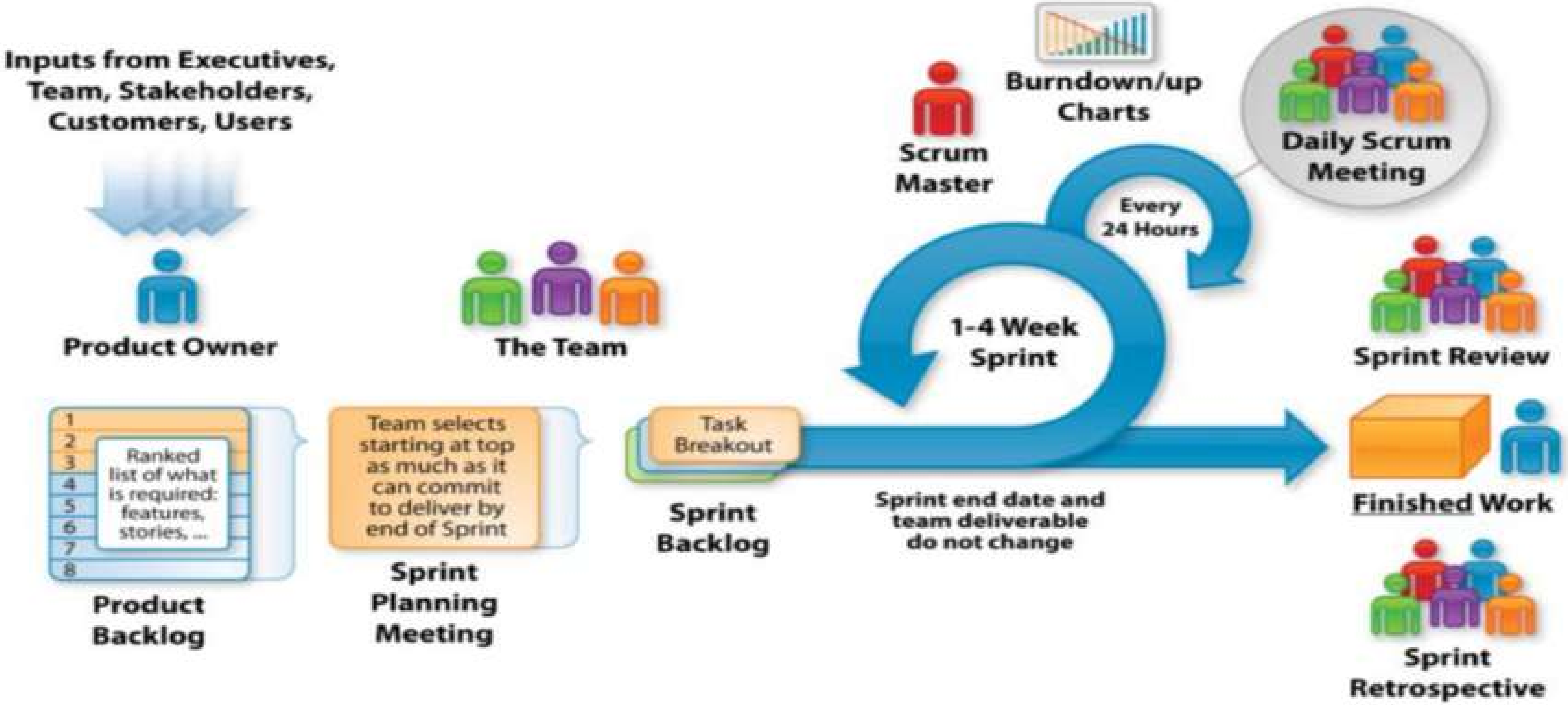
What is Sprint Retrospective?

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 Scrum Master.

It is used for optimizing the process after every iteration.



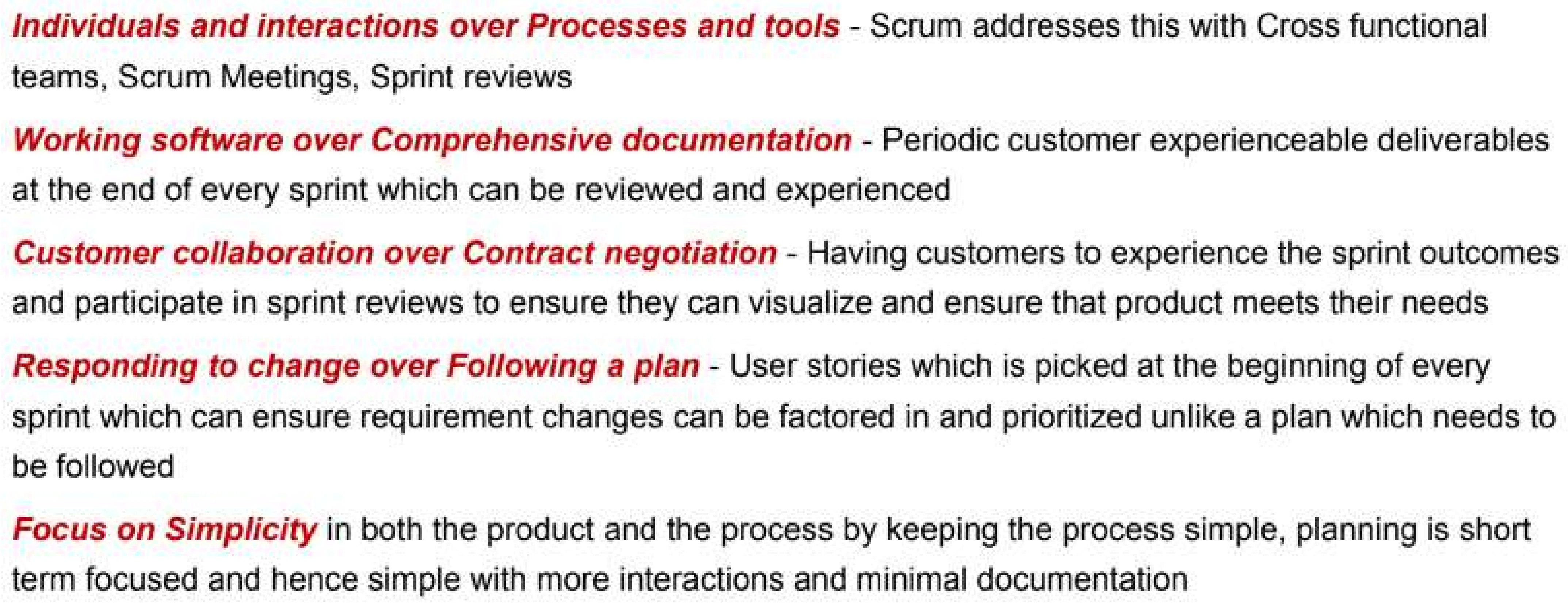
Agile methodologies – Scrum Summary



Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

Agile methodologies – Scrum In-Class Exercise

How is Scrum aligned to the Agile Manif o?



Individuals and interactions over Processes and tools - Scrum addresses this with Cross functional teams, Scrum Meetings, Sprint reviews

Working software over Comprehensive documentation - Periodic customer experienceable deliverables at the end of every sprint which can be reviewed and experienced

Customer collaboration over Contract negotiation - Having customers to experience the sprint outcomes and participate in sprint reviews to ensure they can visualize and ensure that product meets their needs

Responding to change over Following a plan - User stories which is picked at the beginning of every sprint which can ensure requirement changes can be factored in and prioritized unlike a plan which needs to be followed

Focus on Simplicity in both the product and the process by keeping the process simple, planning is short term focused and hence simple with more interactions and minimal documentation

Agile methodologies – Wordplay!

AGILE? I'M SURE THE METHODOLOGY SUB-COMMITTEE CAN HELP!
USUALLY MOST REQUESTS FOR PROCESS VARIANCE GET APPROVED WITHIN SIX MONTHS.
AS LONG AS YOU CAN COMPLETE ALL THE NORMAL DOCUMENTATION, PROCESS REVIEW CHECKPOINTS, STATUS REPORTS, AND CHANGE CONTROL MANAGEMENT, I'M SURE YOUR AGILE THING WILL BE APPROVED.

Agile methodologies – Extreme Agile Programming (XP)

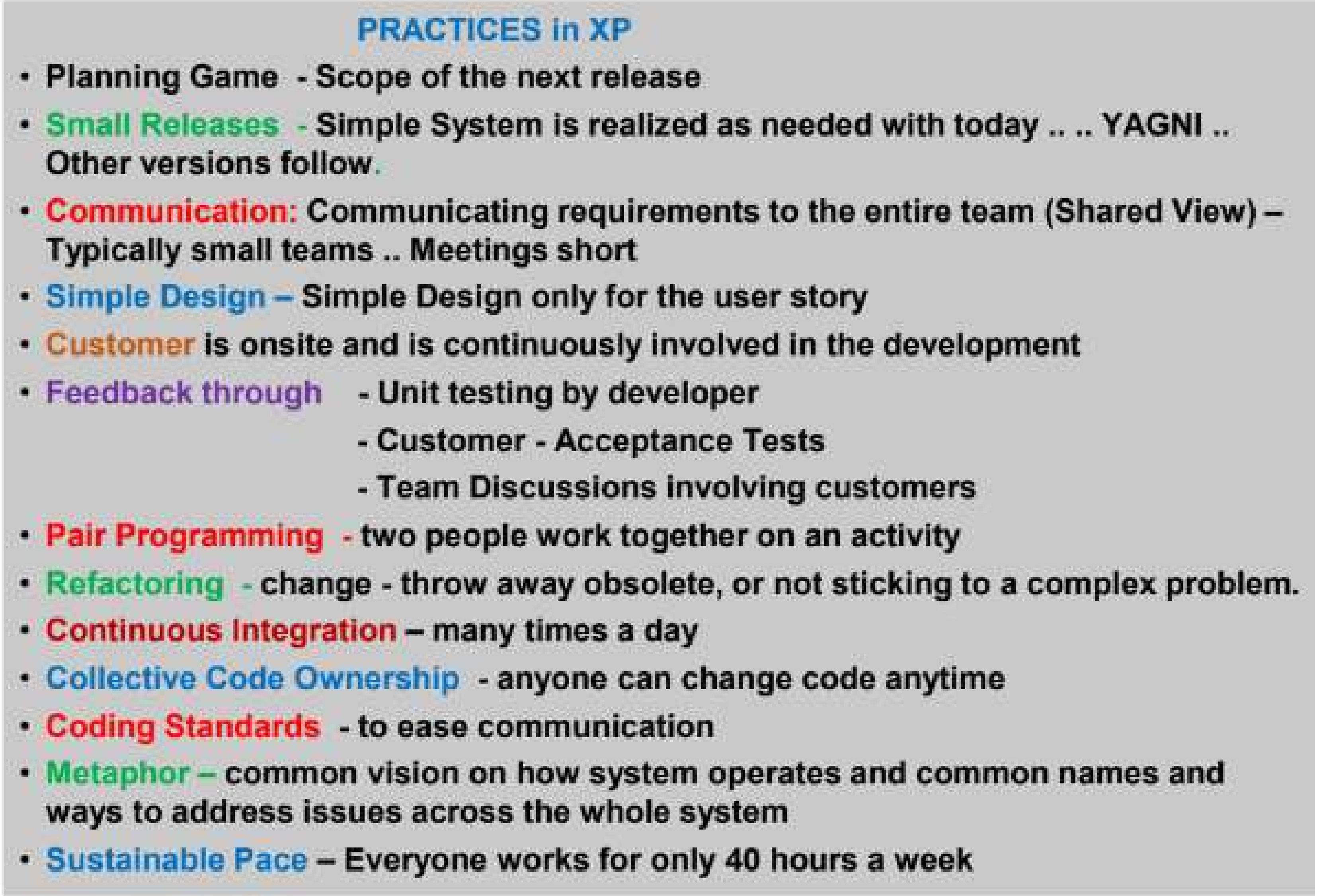
What is XP?

- The development team estimates, plans, and delivers the highest priority user stories in the form of working, tested software on an iteration by iteration basis
- Delivery of working software at very frequent intervals, typically every 1–2 weeks
- Continuous feedback and test–driven development

SCRUM	XP
Framework for management of project	Specifies engineering practices like pair programming, test driven development
Requirement change granularity is once	Requirement change granularity is anytime
Features not developed in strict order	Features developed in strict order

Usage: Requirements are unsure
System is not too big
Customer is onsite

Agile methodologies – Extreme Agile



PRACTICES in XP

- Planning Game - Scope of the next release
- Small Releases - Simple System is realized as needed with today .. .. YAGNI .. Other versions follow.
- Communication: Communicating requirements to the entire team (Shared View) – Typically small teams .. Meetings short
- Simple Design – Simple Design only for the user story
- Customer is onsite and is continuously involved in the development
- Feedback through - Unit testing by developer
  - Customer - Acceptance Tests
  - Team Discussions involving customers
- Pair Programming - two people work together on an activity
- Refactoring - change - throw away obsolete, or not sticking to a complex problem.
- Continuous Integration – many times a day
- Collective Code Ownership - anyone can change code anytime
- Coding Standards - to ease communication
- Metaphor – common vision on how system operates and common names and ways to address issues across the whole system
- Sustainable Pace – Everyone works for only 40 hours a week

Agile methodologies – Lean Agile

Agile and Lean are two popular approaches that help teams deliver faster, more sustainable results and thus value to customers

AGILE

Aim for iterative development that delivers early prototype of a new product or service or a subset of features out into customers' hands as quickly as possible

- Small batches
- Iterative and continuous course correction and delivering of components
- Focused on course corrections during development
- Focus is to develop a product which addresses the customer needs and expectations

LEAN

Seek to identify and eliminate activity that is not valued by the customer or end user

- Eliminating waste
- Continuous inspection to adapt and improve. (typically called Kaizen)
- Looks to boost performance
- Focus is to provide a product which addresses the customer needs and expectations in the most efficient fashion

Agile methodologies – Lean Agile Practices

Eliminate waste

Amplify learning using active feedback

Decide as late as possible

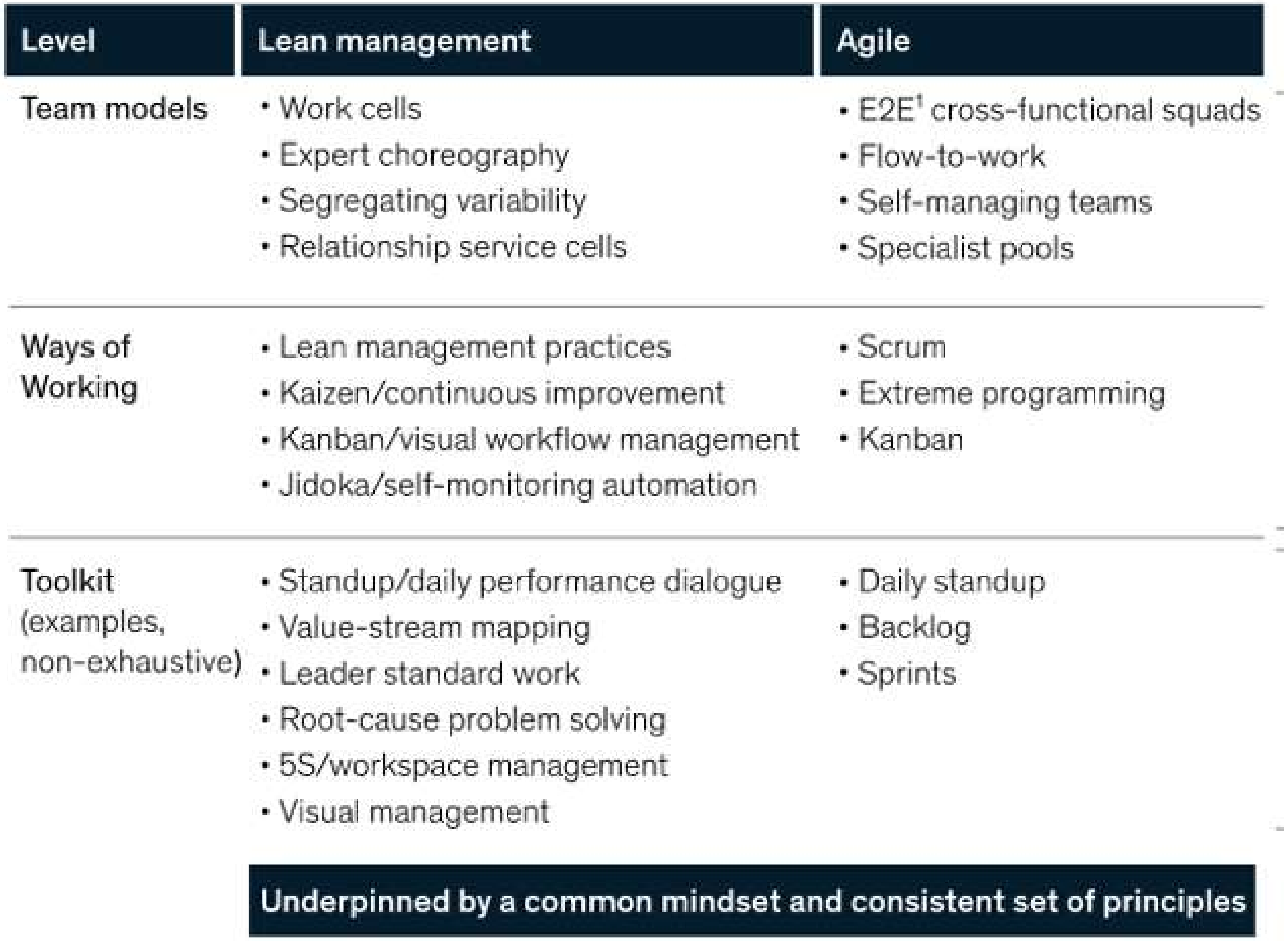
Deliver as fast as possible

Empower the team to follow a controlled low overhead plan

Build integrity in as processes

Consider the whole system

Agile methodologies – Lean Agile



Level	Lean management	Agile
Team models	• Work cells • Expert choreography • Segregating variability • Relationship service cells	• E2E[1] cross-functional squads • Flow-to-work • Self-managing teams • Specialist pools
Ways of Working	• Lean management practices • Kaizen/continuous improvement • Kanban/visual workflow management • Jidoka/self-monitoring automation	• Scrum • Extreme programming • Kanban
Toolkit (examples, non-exhaustive)	• Standup/daily performance dialogue • Value-stream mapping • Leader standard work • Root-cause problem solving • 5S/workspace management • Visual management	• Daily standup • Backlog • Sprints

Underpinned by a common mindset and consistent set of principles

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R
Department of Computer Science and Engineering

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

Software Engineering

Reuse focused Software Development Approaches – CBSE

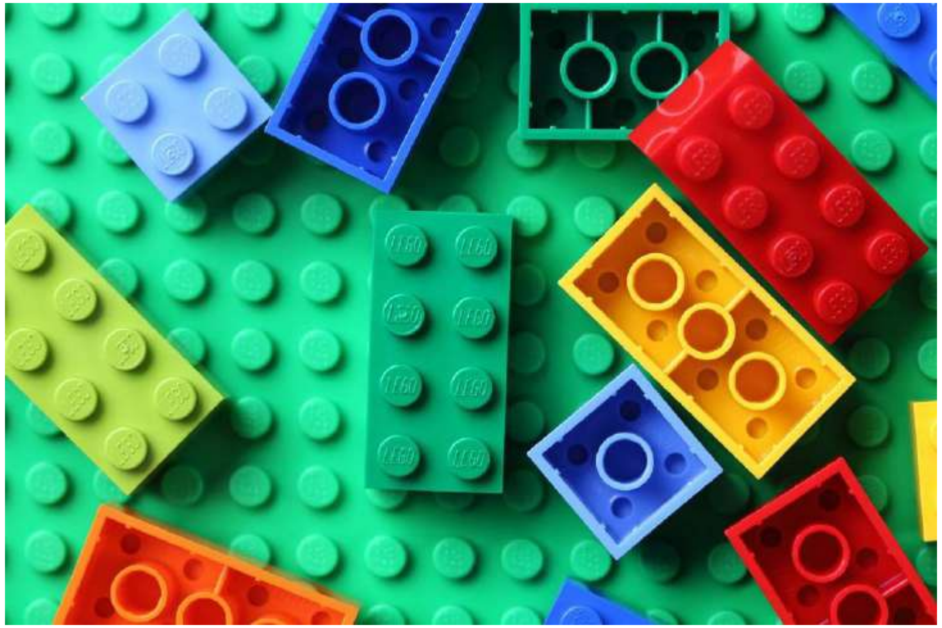
Dr. Jayashree R

Department of Computer Science and Engineering

Introduction – Lego

We all know what Lego is. It is a set of building blocks in different sizes and colors.
They can be combined together to form different shapes.

Lego blocks are generic and easily composable.
But, Lego blocks can only be combined with other Lego blocks. they cannot be combined with any other blocks like Meccano!



Can you identify the common themes across Lego blocks?

Reuse

Quickly assemble models

Build complex models from simple blocks

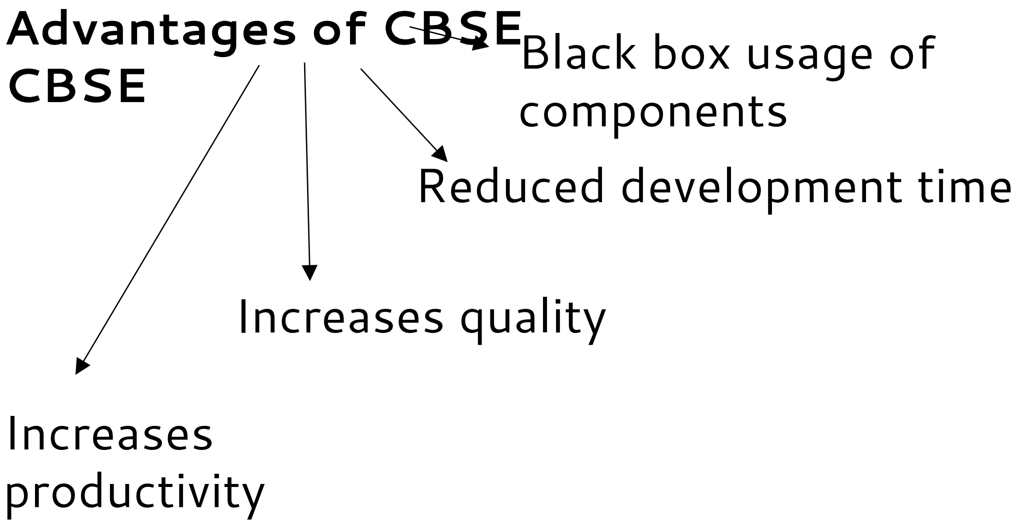
CBSE – Component Based Software Engineering

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

Why do we need CBSE?

- Increase in complexity of systems
- Reuse rather than re-implement and shorten development time



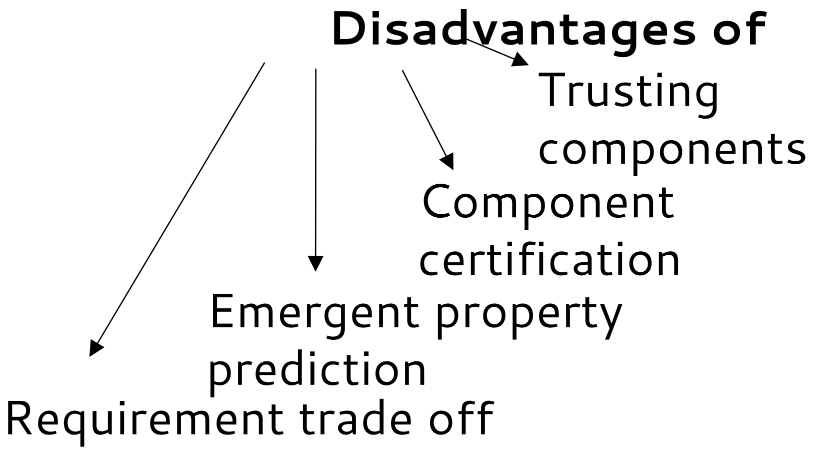
Advantages of CBSE

Black box usage of components

Reduced development time

Increases quality

Increases productivity



Disadvantages of CBSE

Trusting components

Component certification

Emergent property prediction

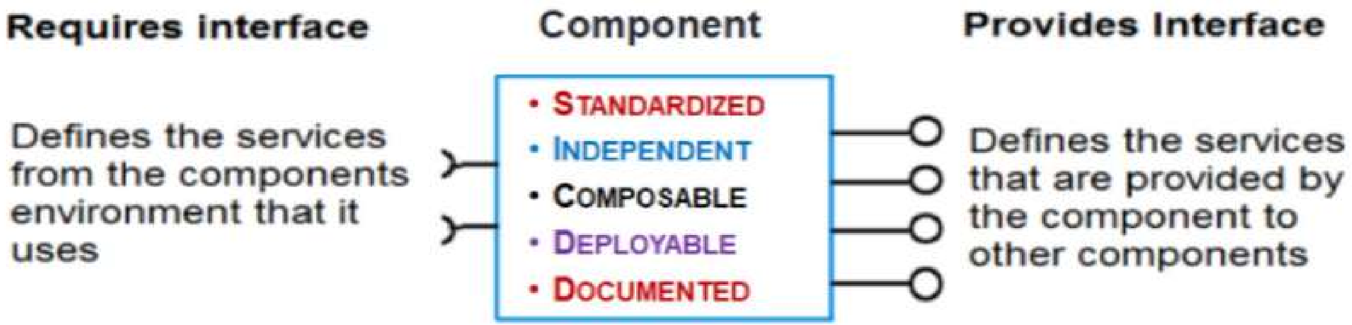
Requirement trade off

Essentials of CBSE

- Independent components that are completely specified by the public interfaces
- Component standards that facilitate the integration of components
- Middleware that provides software support for component integration
- Development process that is geared up to CBSE

Software Component

What is a software component?
Independent executable entity that can be made up of one or more executable objects.
It has explicit dependencies through "required" interfaces and "provides".



Requires interface

Defines the services from the components environment that it uses

Component

- STANDARDIZED
- INDEPENDENT
- COMPOSABLE
- DEPLOYABLE
- DOCUMENTED

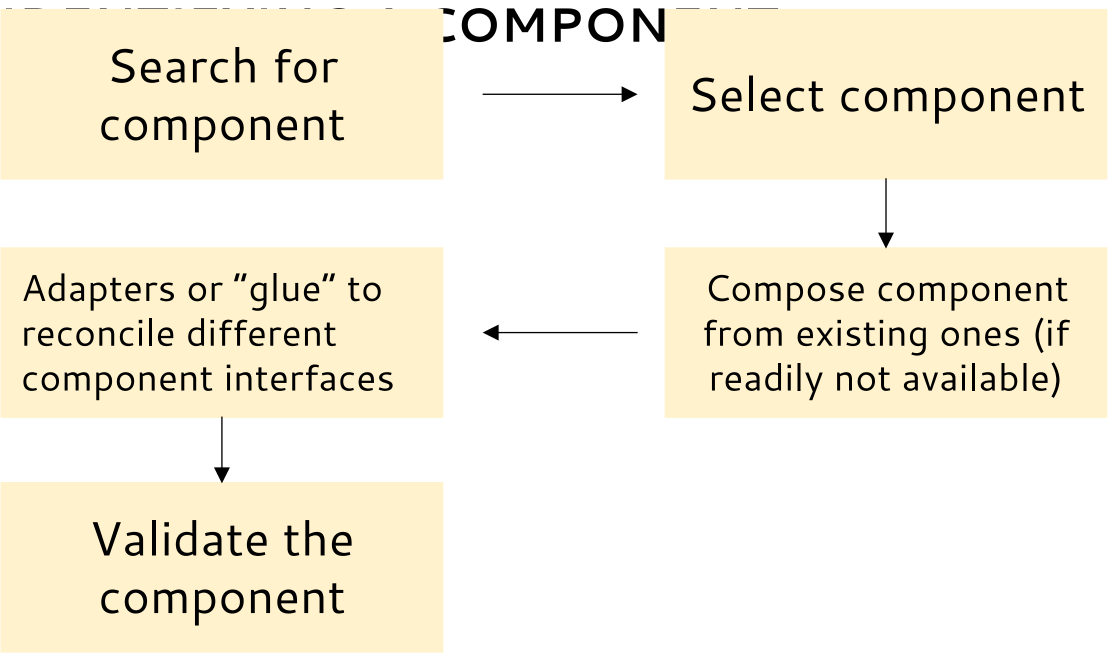
Provides Interface

Defines the services that are provided by the component to other components

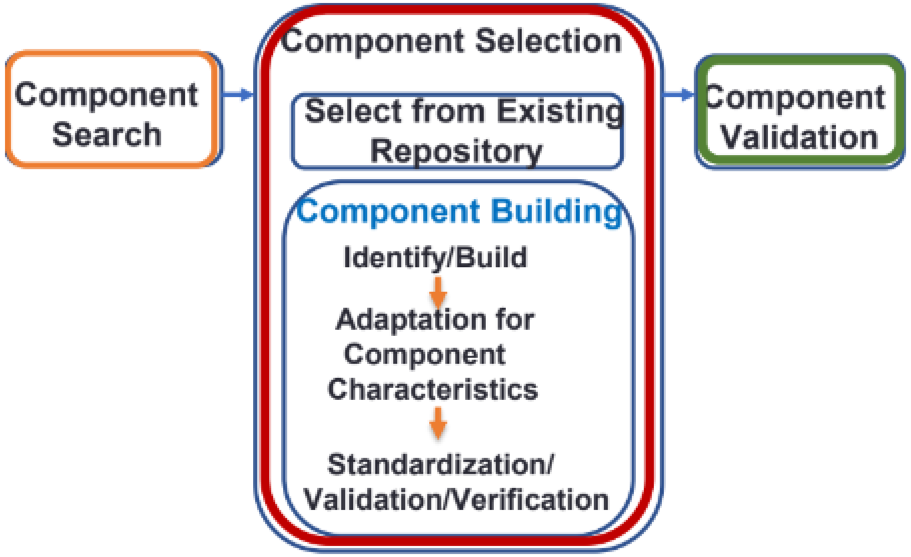
Software Component

The component interface is published and all interactions are through the published interface.

What does a software component do?
Implements a functionality without regard to where the component is executing or its programming language

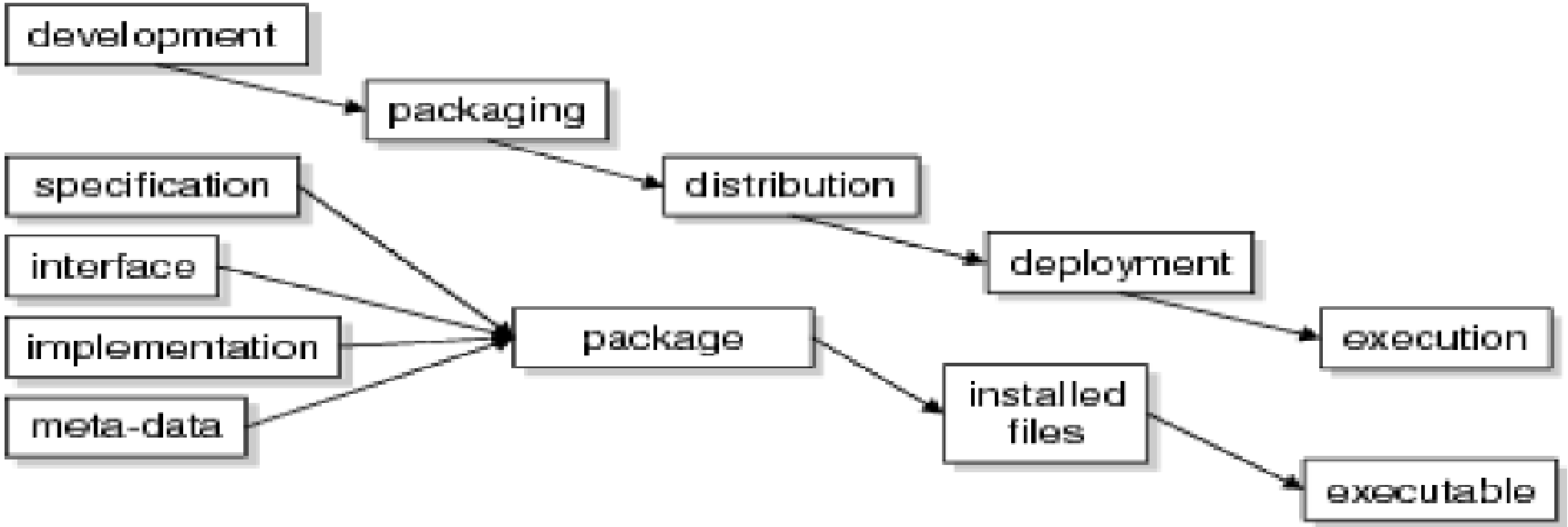


COMPON
Search for component
Select component
Compose component from existing ones (if readily not available)
Adapters or "glue" to reconcile different component interfaces
Validate the component



Component Search
Component Selection
Select from Existing Repository
Component Building
Identify/Build
Adaptation for Component Characteristics
Standardization/ Validation/Verification
Component Validation

Component Development Stages



Different forms of component representation:

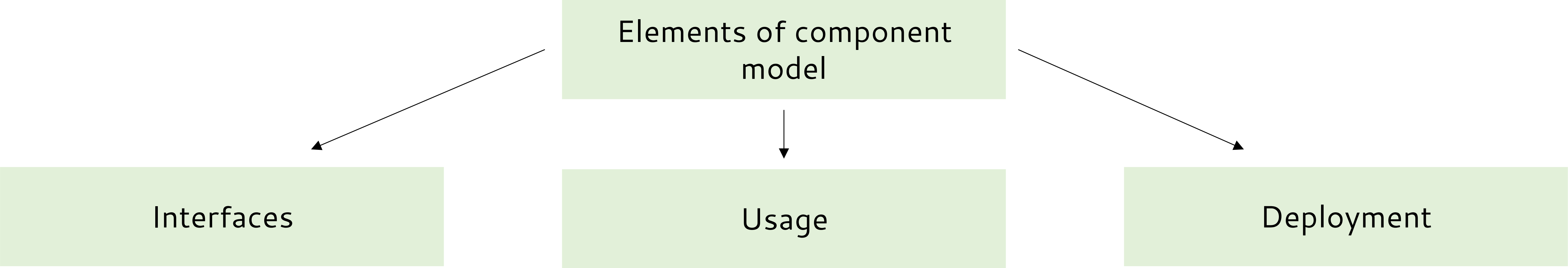
During development – UML

When packaging – .zip

In the execution stage – blocks of code and data

Component model

Component model – defines the types of building block, which can be composed with other components to create a software system



Elements of component model

Interfaces

Usage

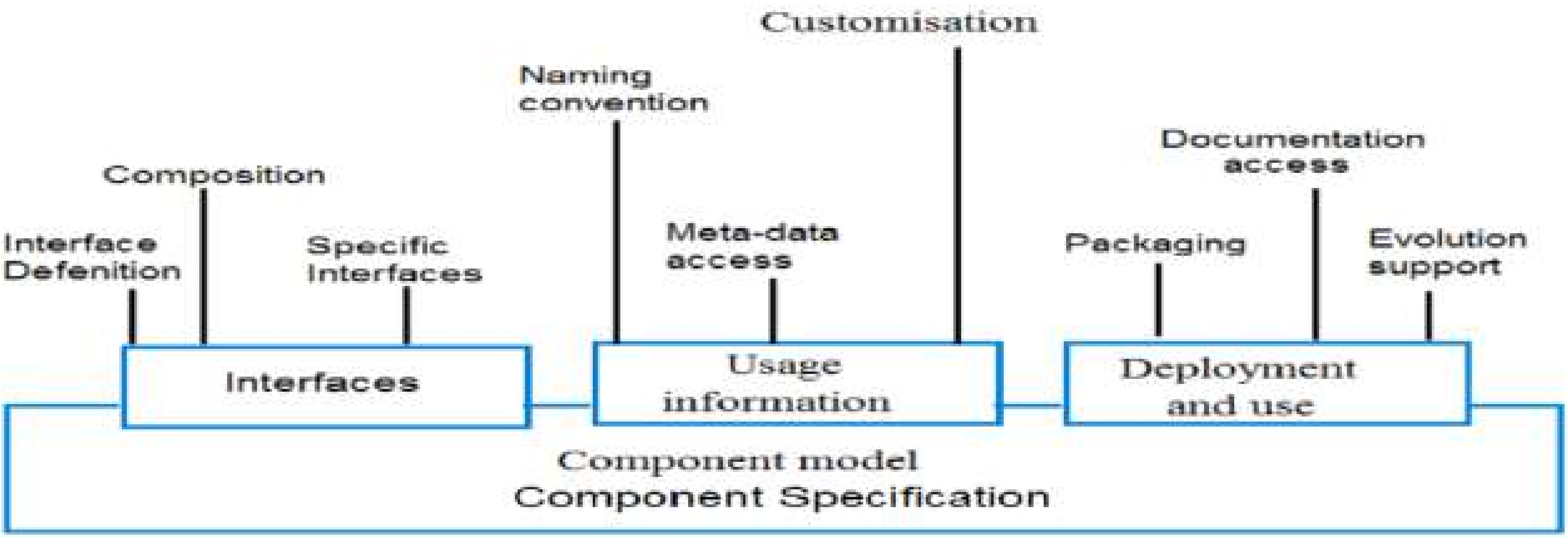
Deployment

Defines how component can interact and also defines operation names, parameters and exceptions

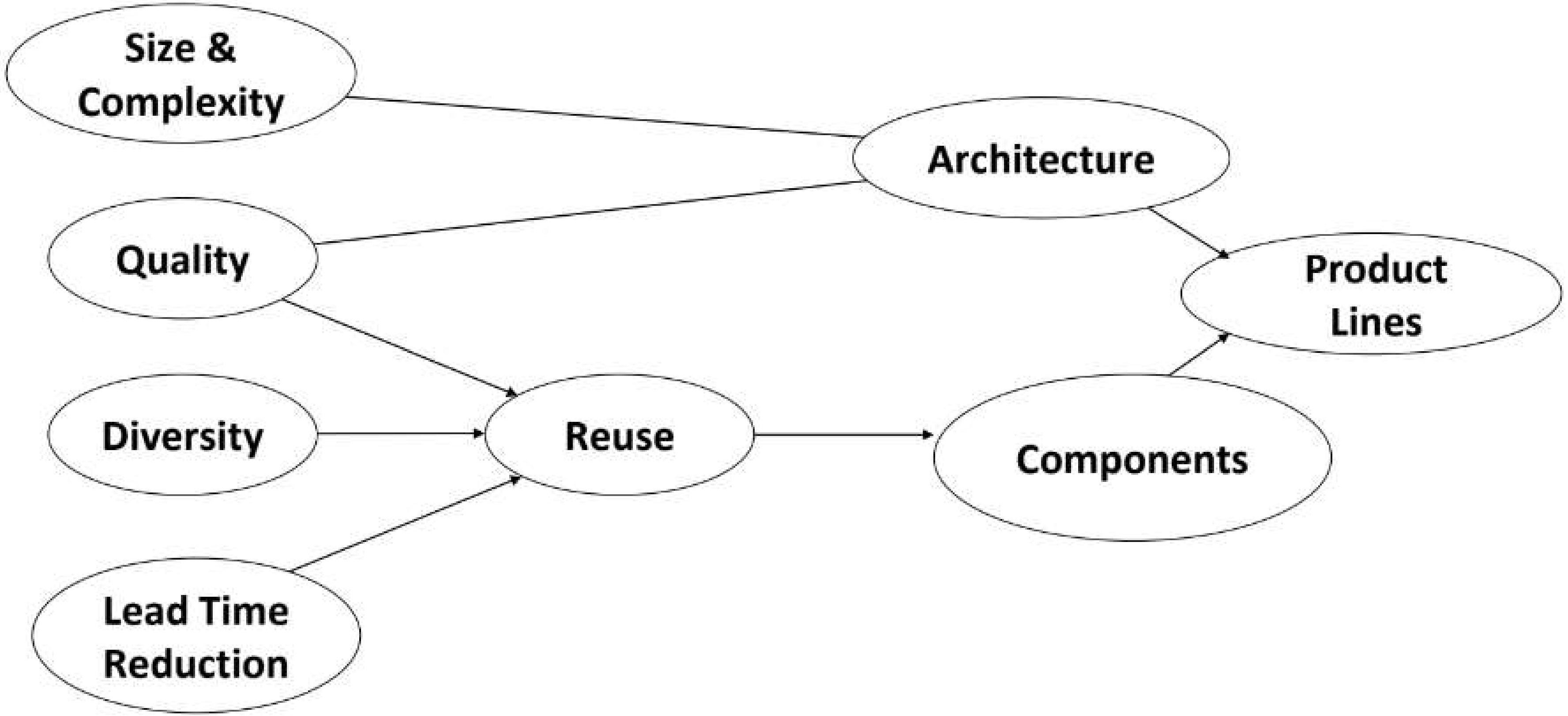
In order for components to be distributed and accessed remotely, they need to have a globally unique name or handle associated with them

Specification of how components should be packaged for deployment as independent, executable entities

Component model



Product Lines



Software Product Lines

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

Software Product Line Engineering makes it possible to

- Create software for different products
- Use variability to customize the software to each different product

Key Drivers for effective product lifecycle re-use

Software product lines enhance reuse through predictive software reuse (rather than opportunistic reuse)

Software artifacts are created when reuse is predicted in one or more products in a well defined product line

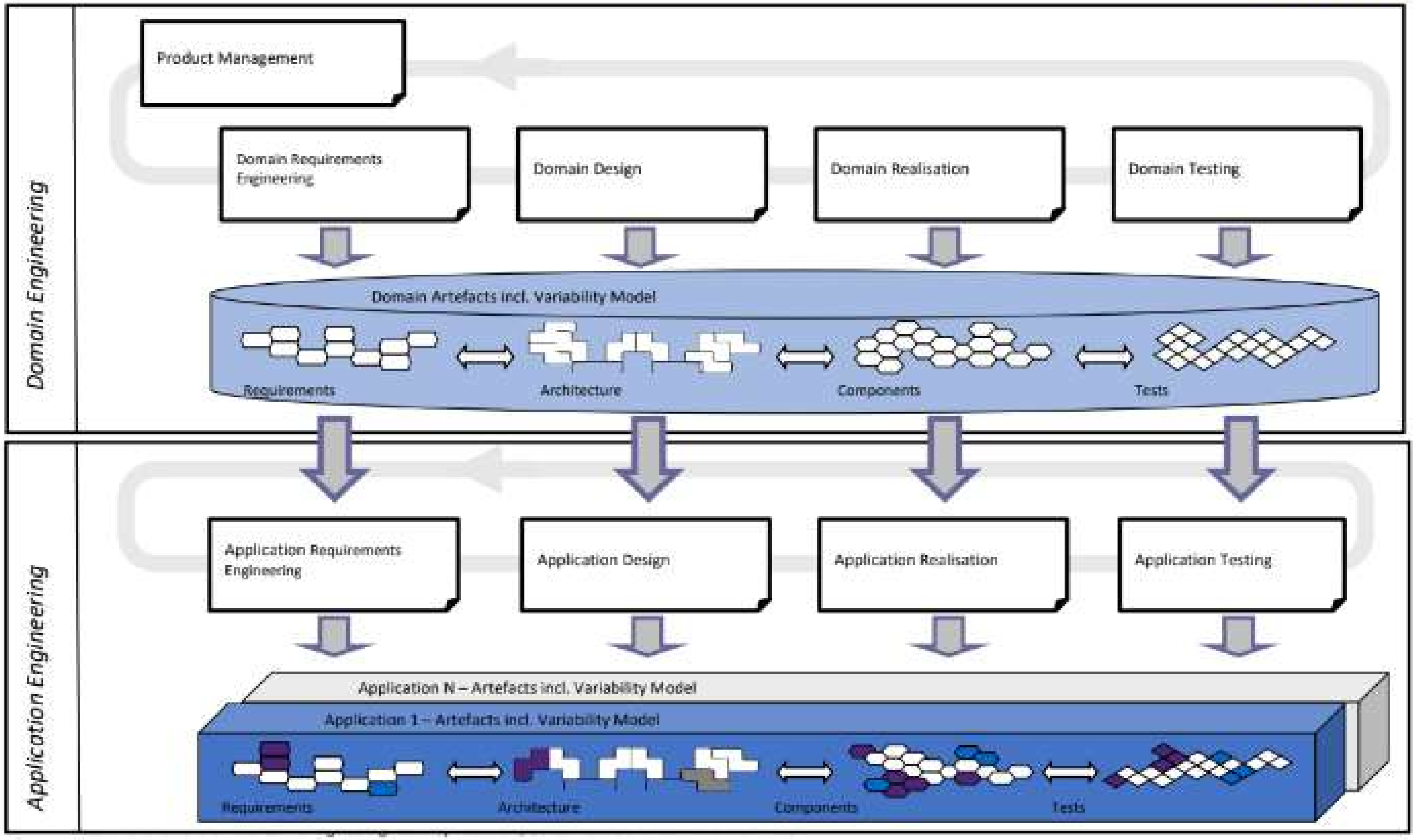
These artifacts could be built as components which are reusable or could be looked at as design patterns which could be built using some fine grained components for a particular solution

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



THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering
jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Apoorva B S (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

Software Engineering

Requirement Engineering

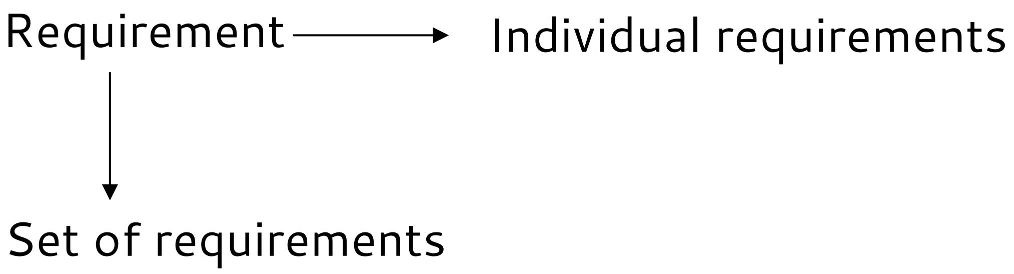
Dr. Jayashree R

Department of Computer Science and Engineering

Requirements Engineering

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.



Requirement → Individual requirements

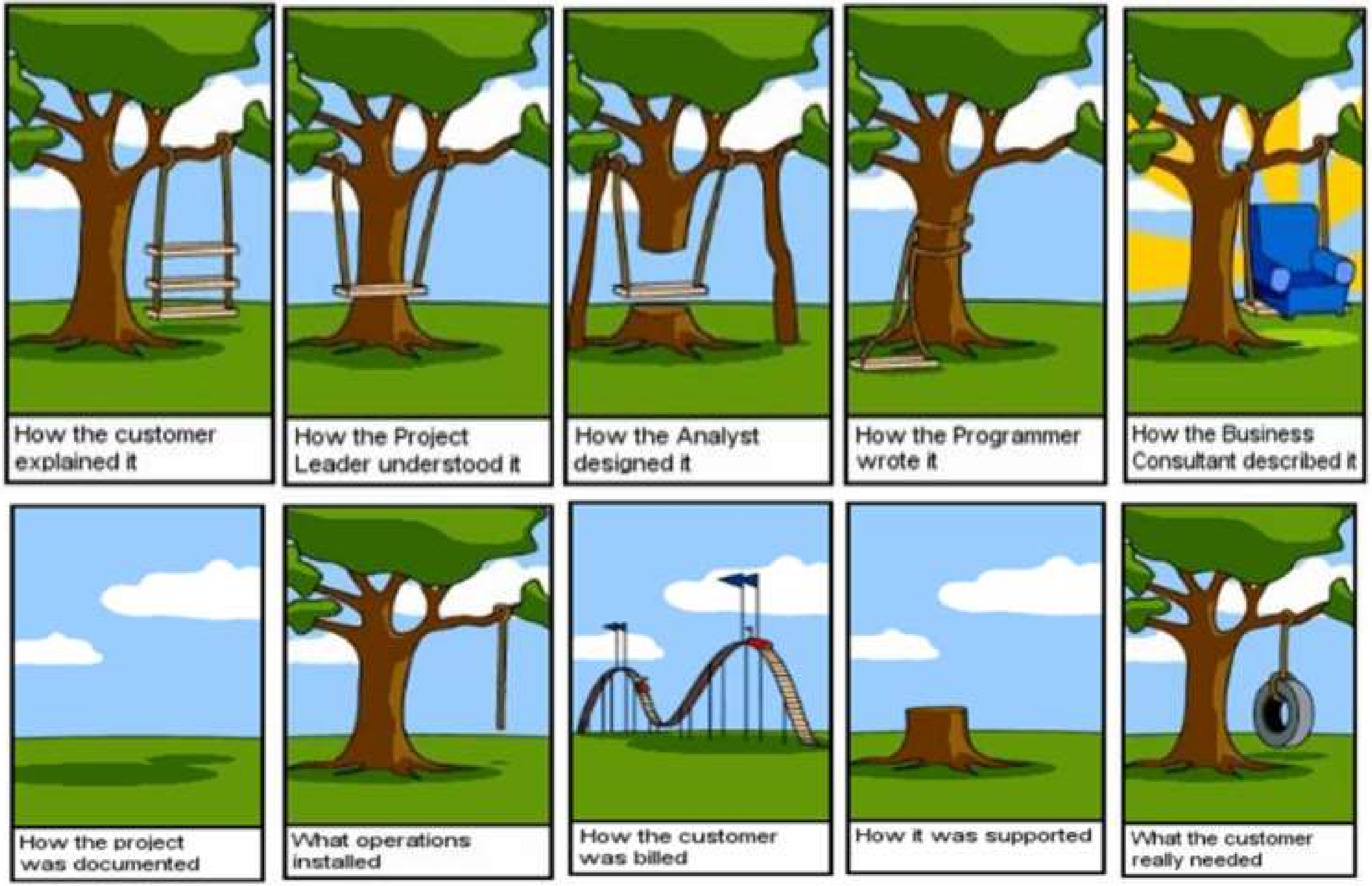
Requirement ↓ Set of requirements

REQUIREMENTS ENGINEERING

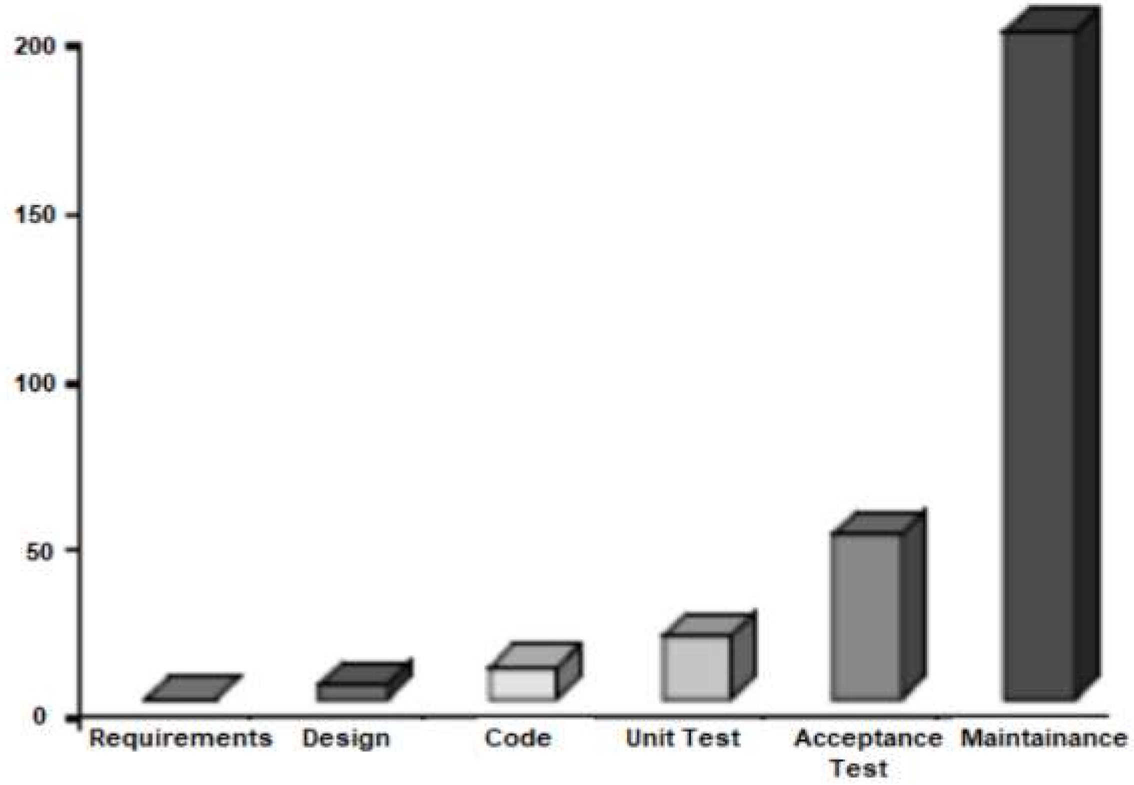
First step in any software intensive development lifecycle irrespective of model

- Difficult, error prone and costly
- Critical for successful development of all down stream activities
- Requirement errors are expensive to fix

Parody!

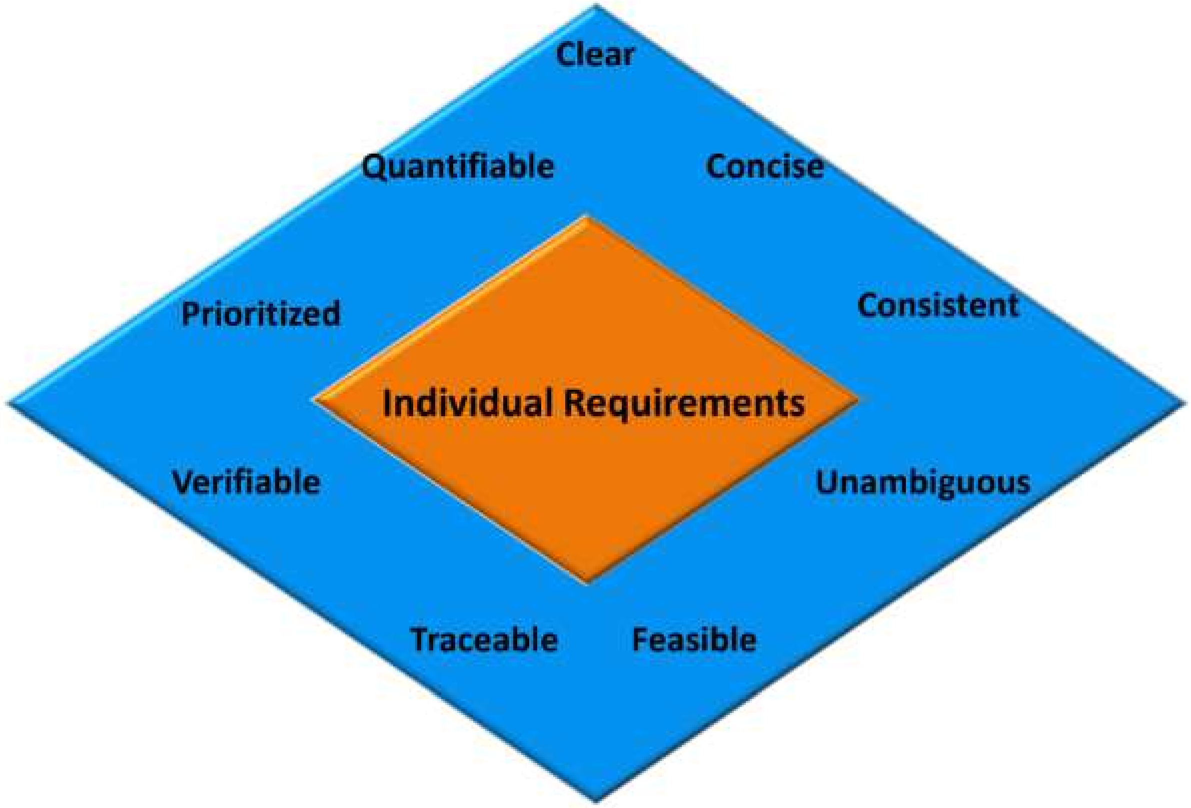

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Cost of repair as a function of



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

Properties of requirement



Concise – Requirements should describe a single property

Requirements Engineering – In-Class Exercises

Use the properties of requirement to transform the given sentences into requirements.

All screens must appear quickly on the monitor

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

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

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

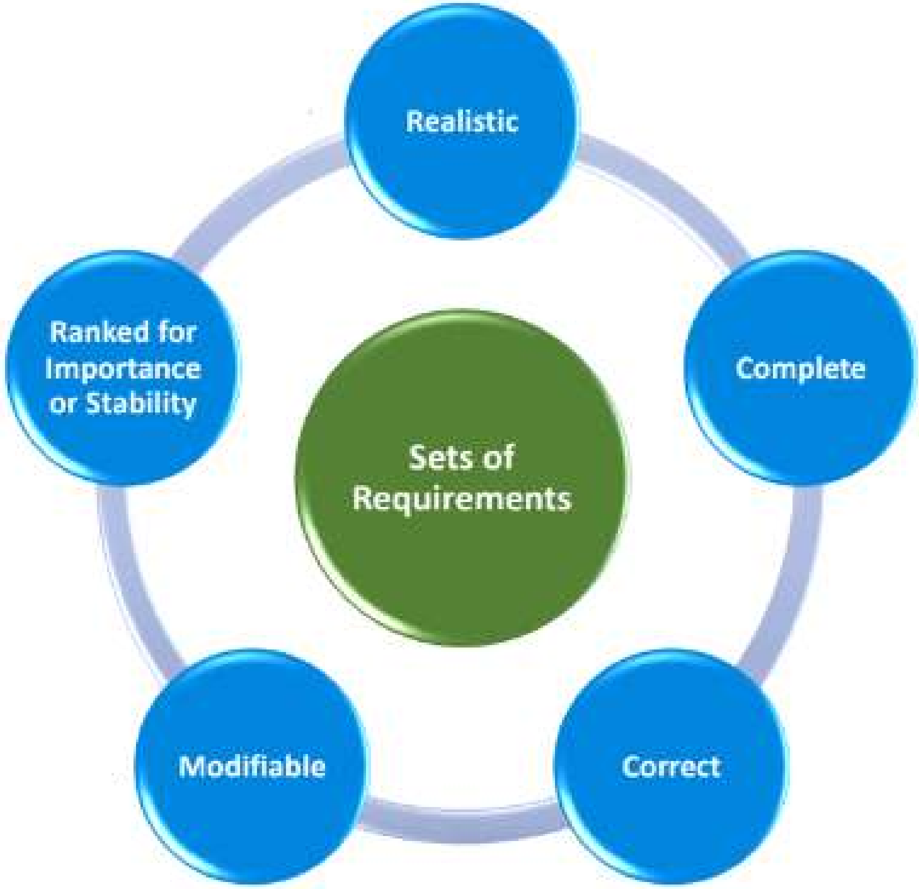
The system must generate a batch end report when a batch is completed or aborted

The system must generate a discrepancy report when a batch is aborted (Traceable)

The system must be user friendly

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)

Properties of a set of requirements



When there are many requirements but limited time or budget, choices must be made about which to include or exclude. Factors such as:

- changes in customer needs
- improved developer understanding of the products
- changes in organizational policy

will affect the stability of requirements.

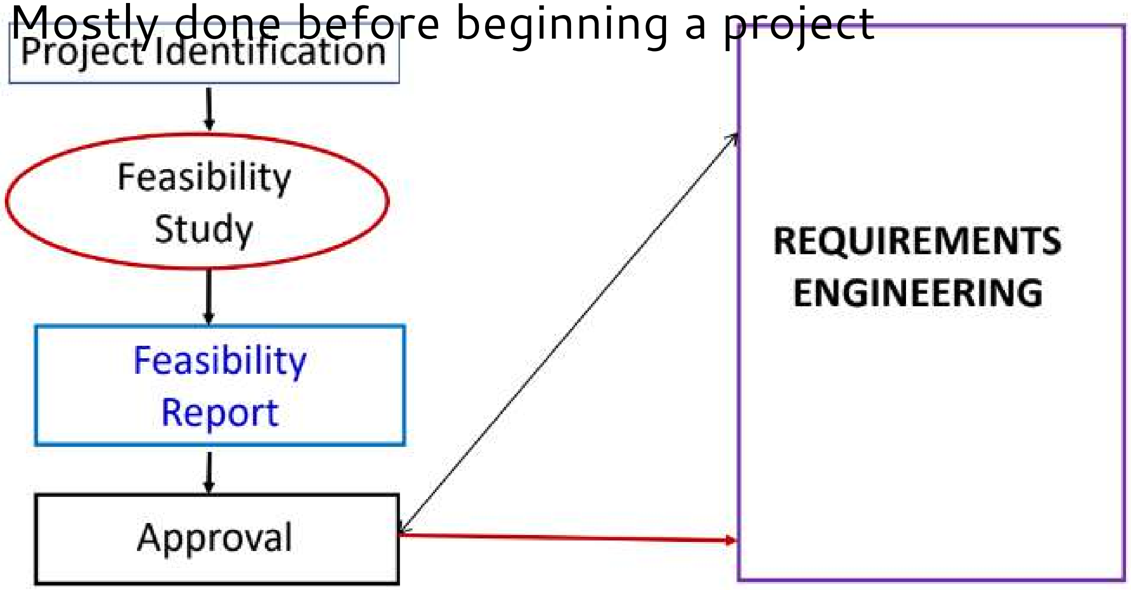
Feasibility Study

What is Feasibility Study?

Short, low-cost study to asses the practicality of the project and whether it should be done

When is Feasibility Study conducted?

Mostly done before beginning a project



ACTIVITIES IN FEASIBILITY STUDY

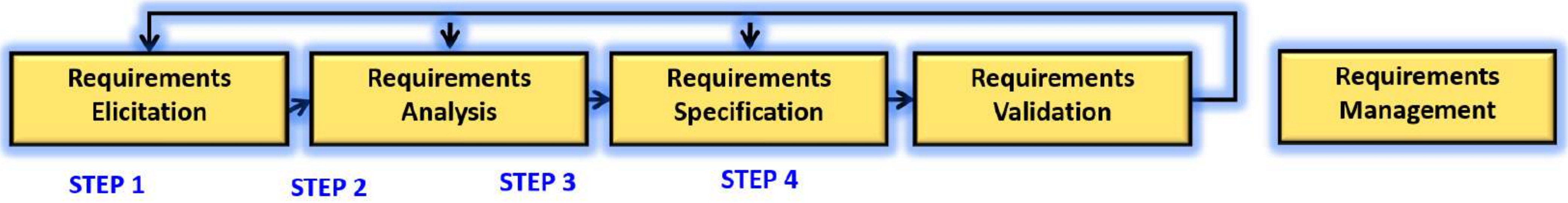
- Figure out the client or the sponsor or the user who would have a stake in the project
- Find the current solution to the problem
- Find the targeted customers and the future market place
- Potential benefits
- Scope
- High level block level understanding of the solution
- Considerations to technology
- Marketing strategy
- Financial projection
- Schedule and high level planning and budget requirements
- Issues, assumptions, risks and constraints
- Alternatives and their consideration
- Potential project organization

Ends with GO or NO-GO

Requirements Engineering process

A "four + one" set of activities to produce specifications or requirements

It is an iterative process



Requirements Validation – Helps ensure the right requirements are realized

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Apoorva B S (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

Software Engineering

Requirement Elicitation & Requirement Analysis

Dr. Jayashree R

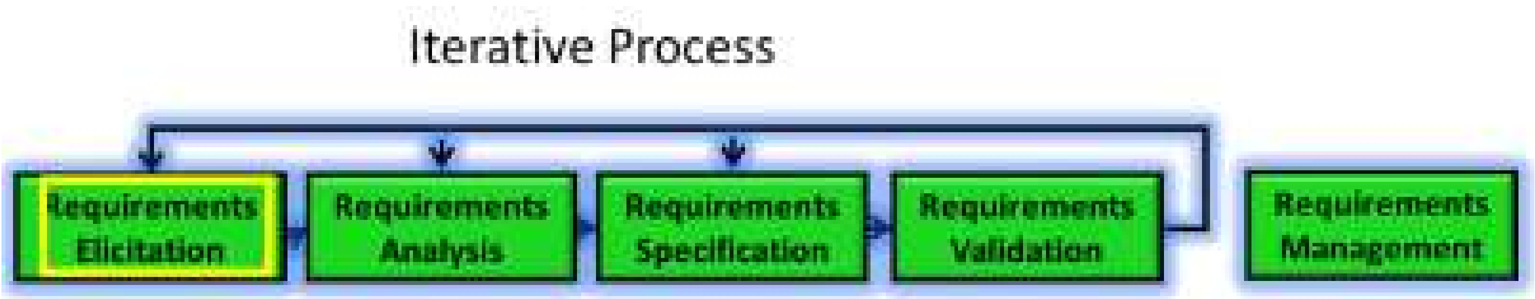
Department of Computer Science and Engineering

Process of Requirement Elicitation

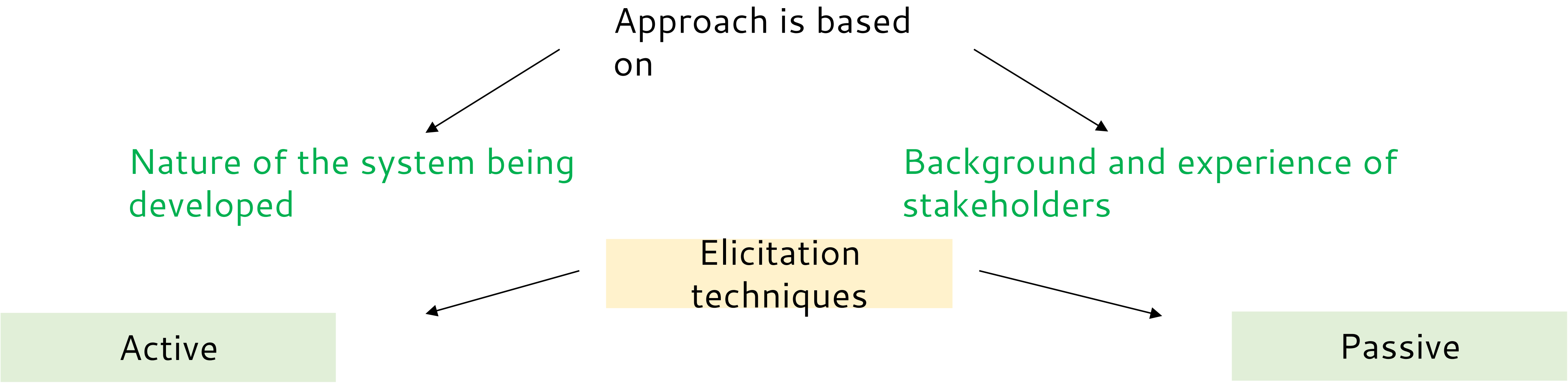
It is the process of working proactively with all stakeholders gathering their needs, articulating their problem, identify and negotiate potential conflicts thereby establishing a clear scope and boundary for a project.

It 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



Approach is based on

Nature of the system being developed

Background and experience of stakeholders

Elicitation techniques

Active

Passive

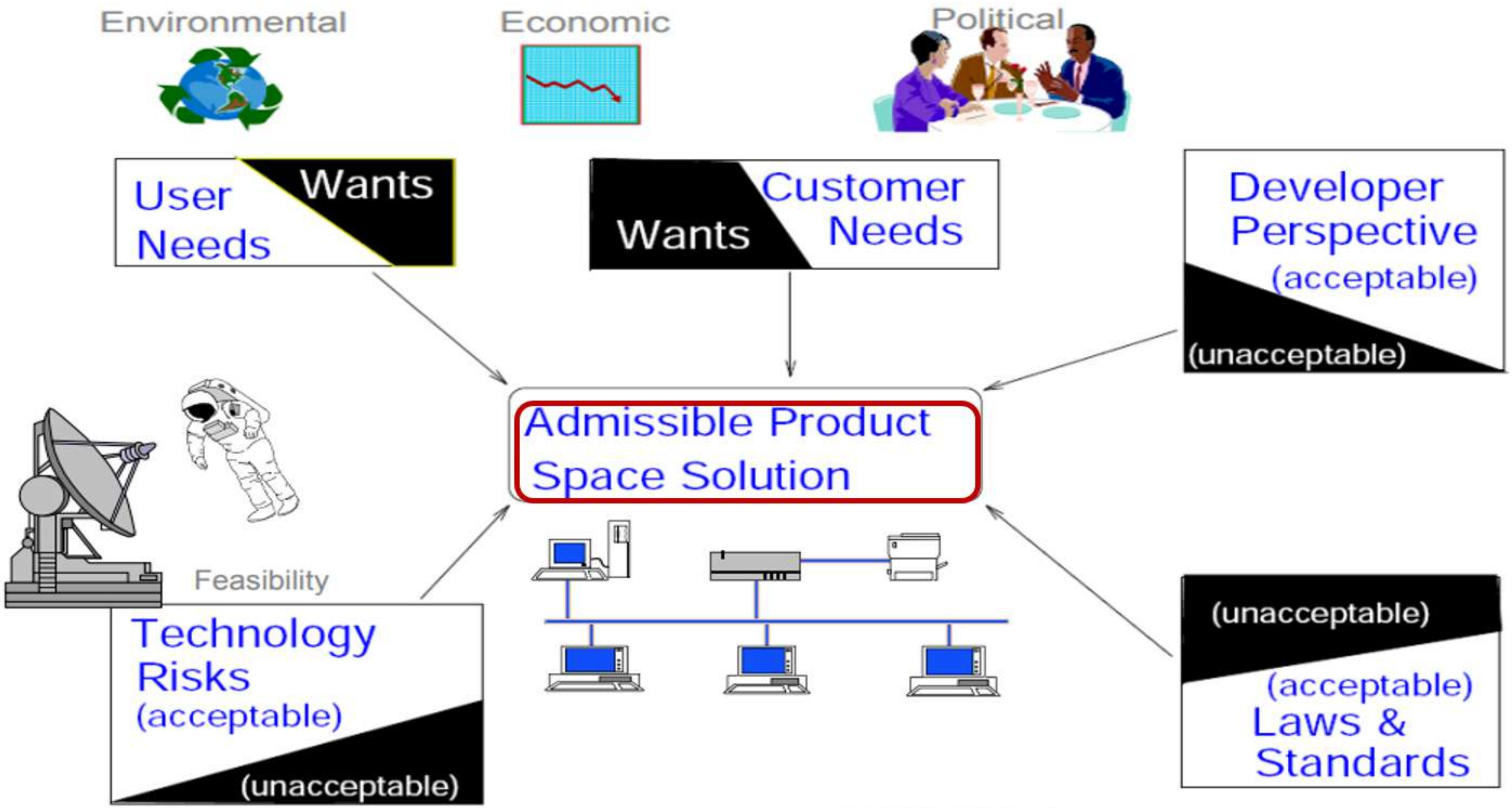
Ongoing interaction between the stake holders and users.

- Interviews
- Facilitated meetings
- Role-playing
- Prototypes
- Ethnography
- Scenarios

Infrequent interaction between the stake holders and users.

- Use cases
- Business process analysis & modelling
- Workflows
- Questionnaires
- Checklists
- Documentation

Process of Requirement Analysis



V3.0 © 2011, IEEE All rights reserved

Process of Requirement Analysis

1. Understand requirements in depth

2. Classify requirements into coherent clusters

3. Model the requirements

4. Analyze requirements using fish bone diagram

5. Recognize and resolve conflicts

6. Negotiate requirements

7. Prioritize requirements – MoSCoW

8. Identify risks

9. Decide on build or buy – COTS solution

Process of Requirement Analysis

Understand requirements in depth

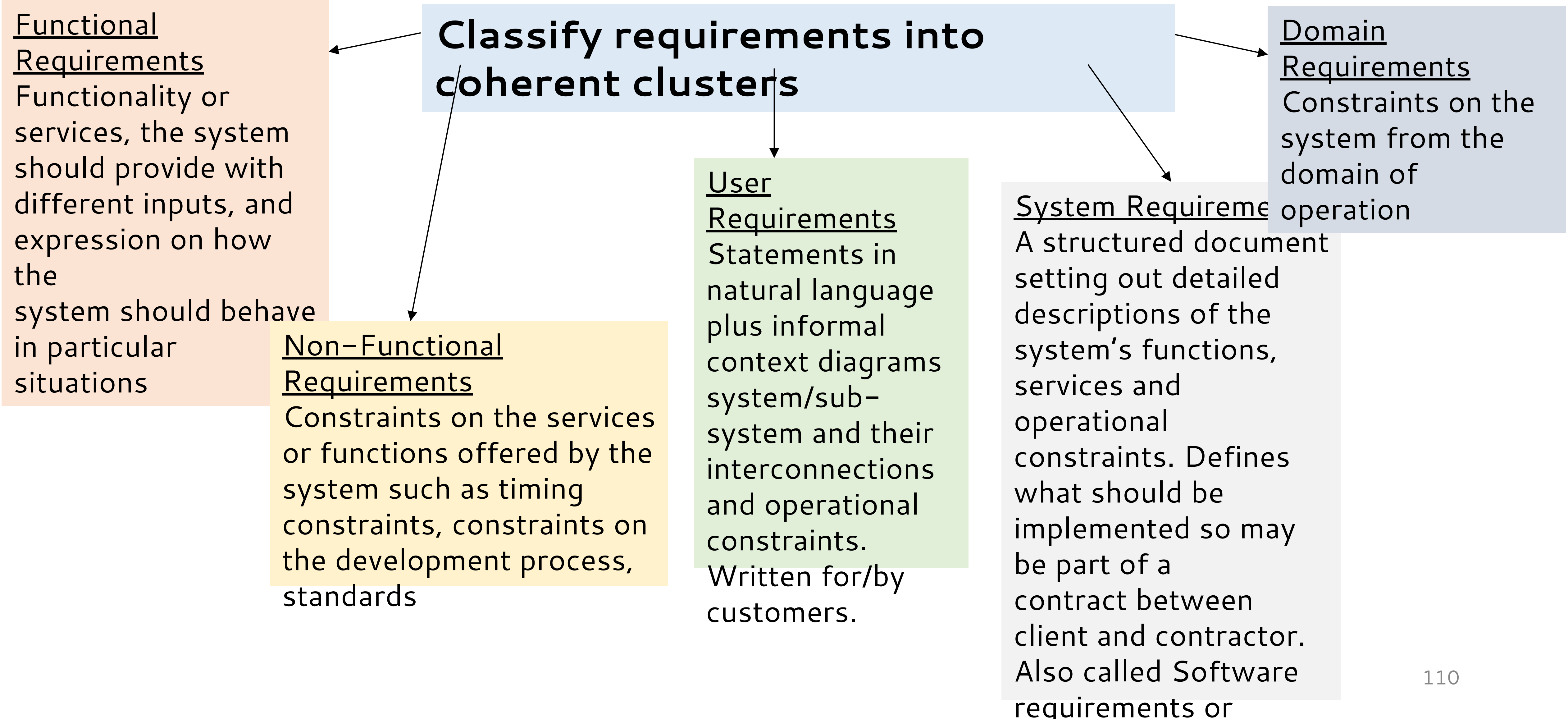
This has to be done from a product and a process perspective
Requirement or Problem needs to be correctly internalized

A Storm coming in

Could be a problem to a game

Could be a blessing to a farmer

Process of Requirement



Classify requirements into coherent clusters

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

Non-Functional Requirements
Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards

User Requirements
Statements in natural language plus informal context diagrams system/sub-system and their interconnections and operational constraints. Written for/by customers.

System Requireme
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 or

Domain Requirements
Constraints on the system from the domain of operation

Identify the class of Requirement

System shall assign a unique tracking number to each shipment

Functional
Requirement

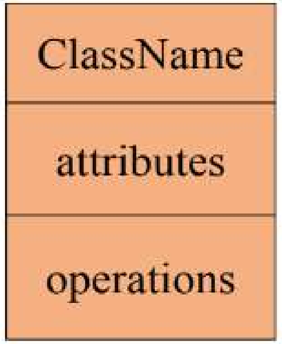
With 100 concurrent users a database record shall be fetched over the network in less than 3ms

Non-Functional
Requirement

Process of Requirement Analysis

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.
We can use the following Class notation as a model.

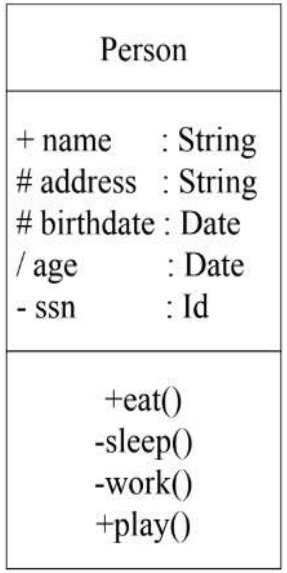


ClassName

attributes

operations

For example, to represent a person –



Person

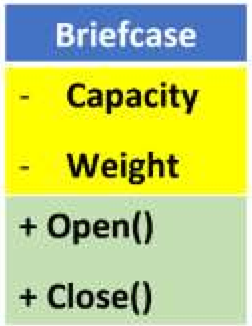
+ name : String
# address : String
# birthdate : Date
/ age : Date
- ssn : Id

+eat()
-sleep()
-work()
+play()

+, #, – and ˜ are types of visibility

+: public
– : private
#: protected
˜: package

Think about:
Try to model a briefcase similarly



Briefcase

- Capacity
- Weight

+ Open()
+ Close()

Process of Requirement Analysis

Primary goals of Modelling

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

Models could be ***Structural Models*** and ***Behavioral models***

Structural Models

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

Example: Class diagram

Behavioral Models

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

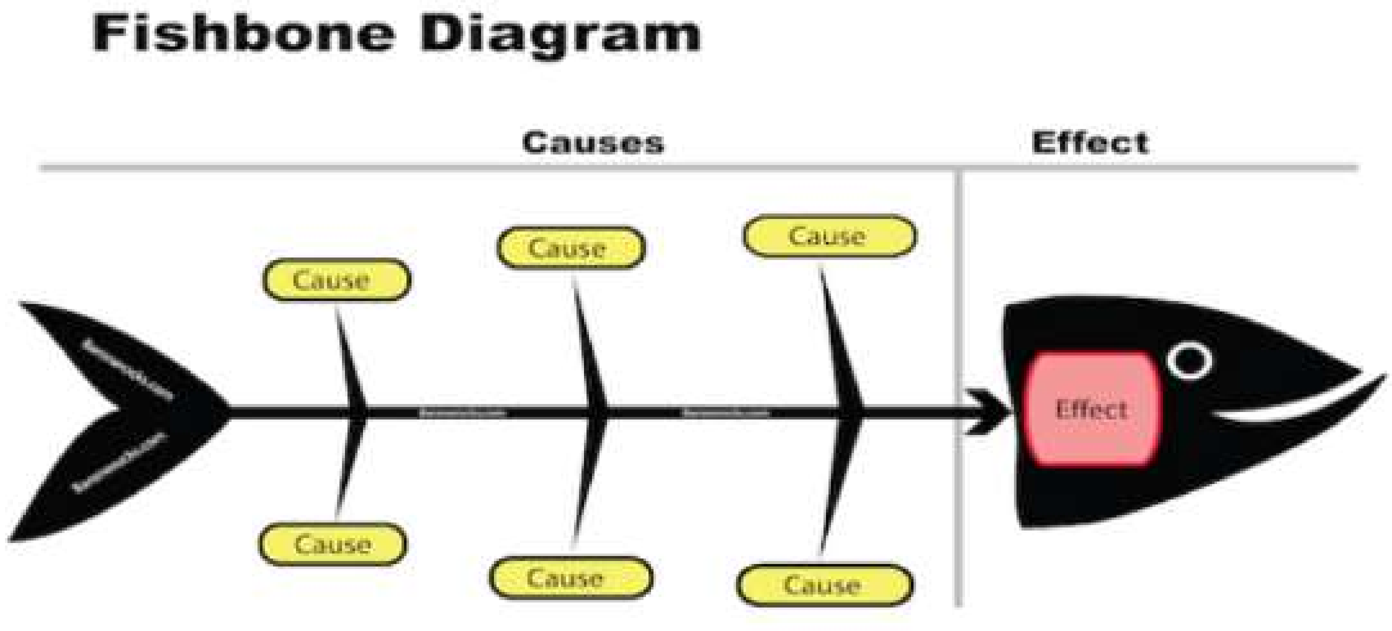
Example: Use Case diagram

Use-case models discussed earlier are the popular models used during analysis

Process of Requirement Analysis

Analyze requirements using fish bone diagram

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



Process of Requirement Analysis

Recognize and resolve conflicts

Functionality vs Cost vs Timelines

Negotiate requirements

Prioritize the requirements (MoSCoW –Must have, Should have, Could have, Won’t have)

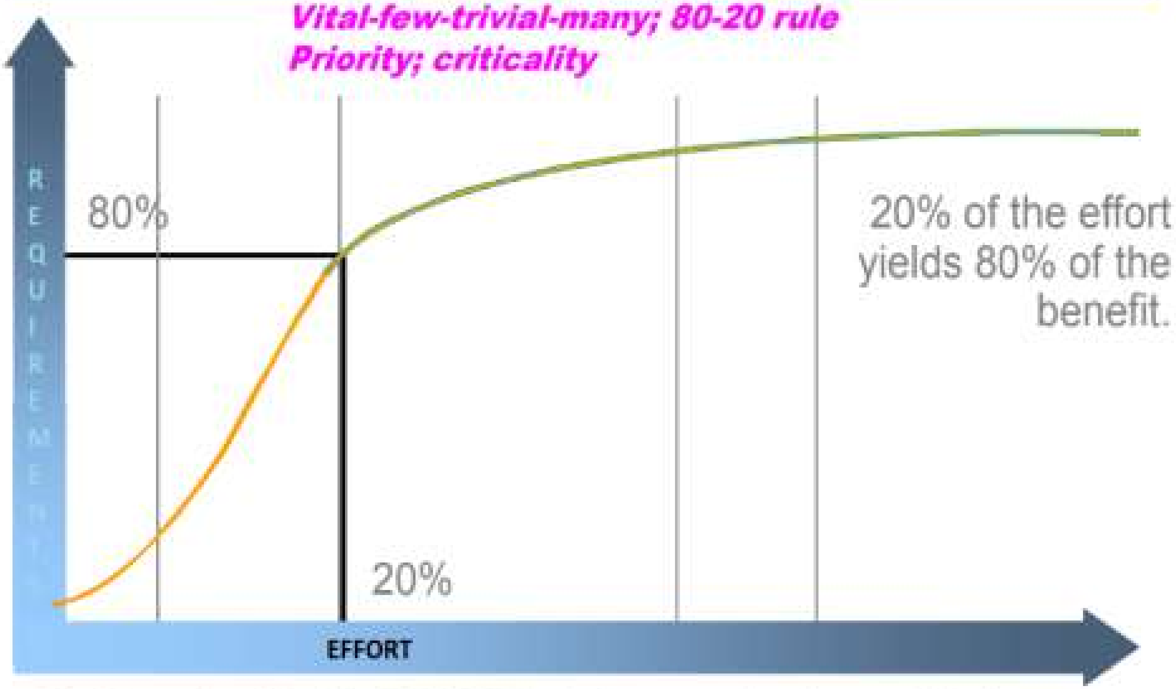
Pareto Analysis (80-20 to focus on vital few to trivial many)
Vital Few (Prioritize and needed critically) – Trivial Many (Lower priority) 80-20 Rule

Process of Requirement Analysis

Focus on Largest Contributors - *Pareto's Law*

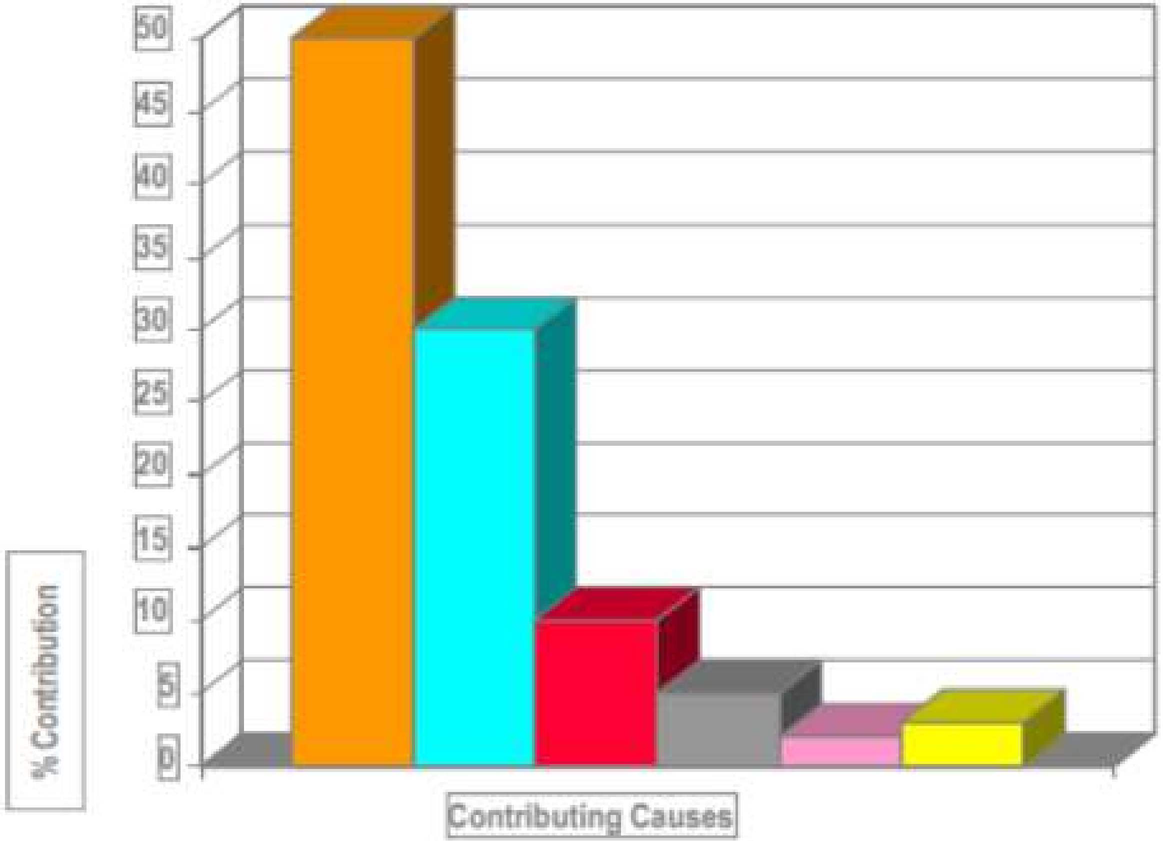
Vital-few-trivial-many; 80-20 rule

Priority; criticality



20% of the effort yields 80% of the benefit.

Rank in order. Use the 80-20 Rule to focus on the top contributing causes to address the greatest portion of the problem.

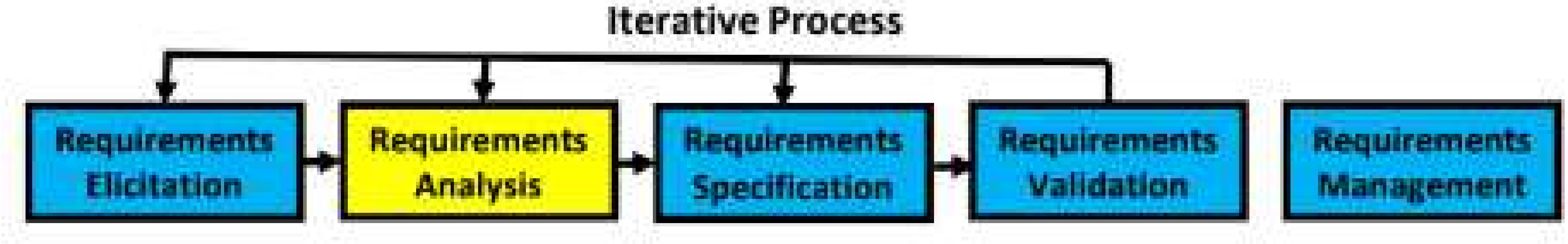


Process of Requirement Analysis

Identify risks

Decide on build or buy – COTS solution

Commercial Off The Shelf solution – COTS



Unified Modelling Language – UML

What is UML?

UML (Unified Modeling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of complex software systems. Controlled by OMG consortium – Object Management Group.

UML plays an important role in defining different perspectives of a system

Design

Implementation

Process

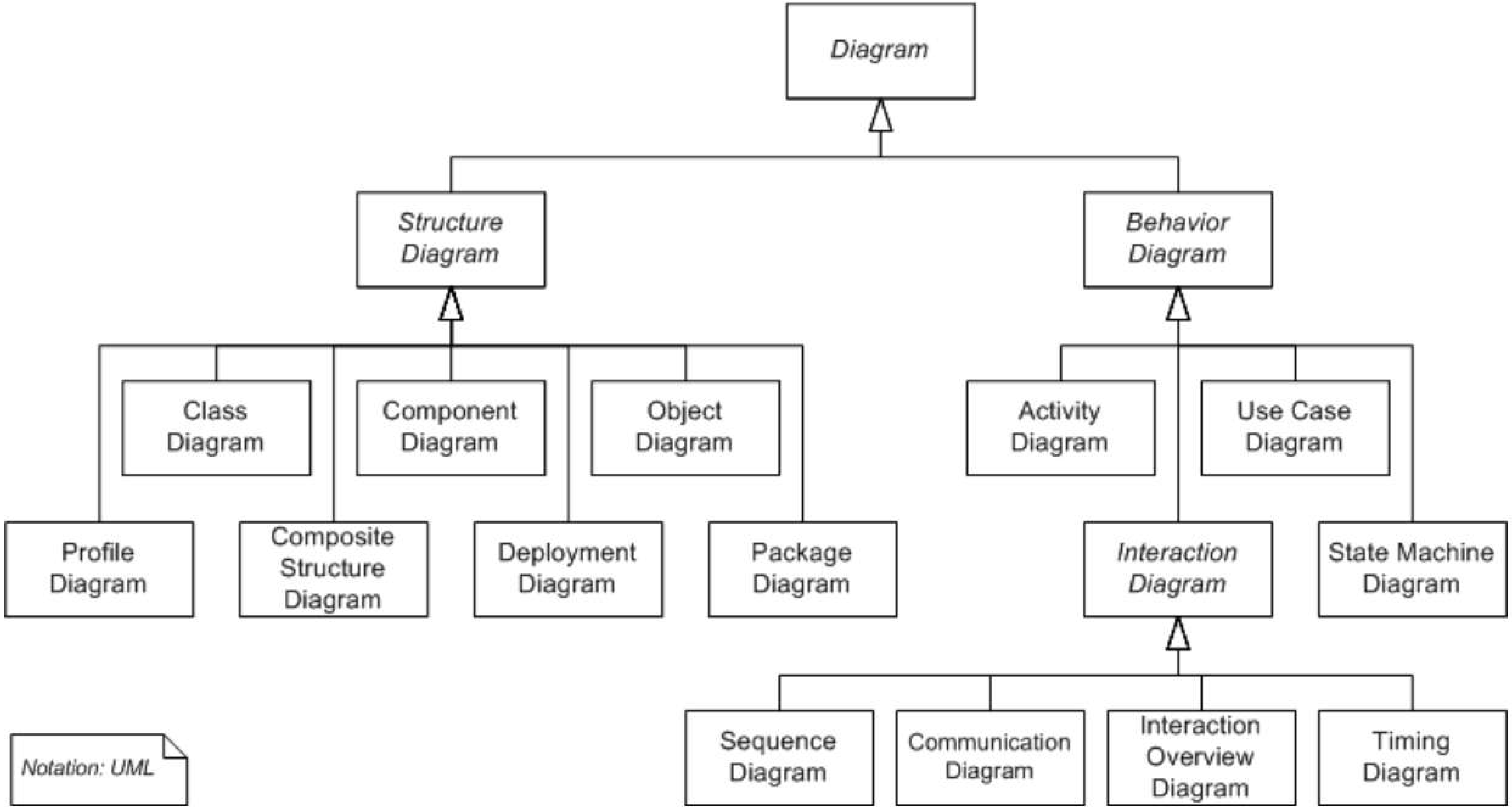
Deployment

Use Case view representing the functionality of the system connecting all of them

Why is UML used?

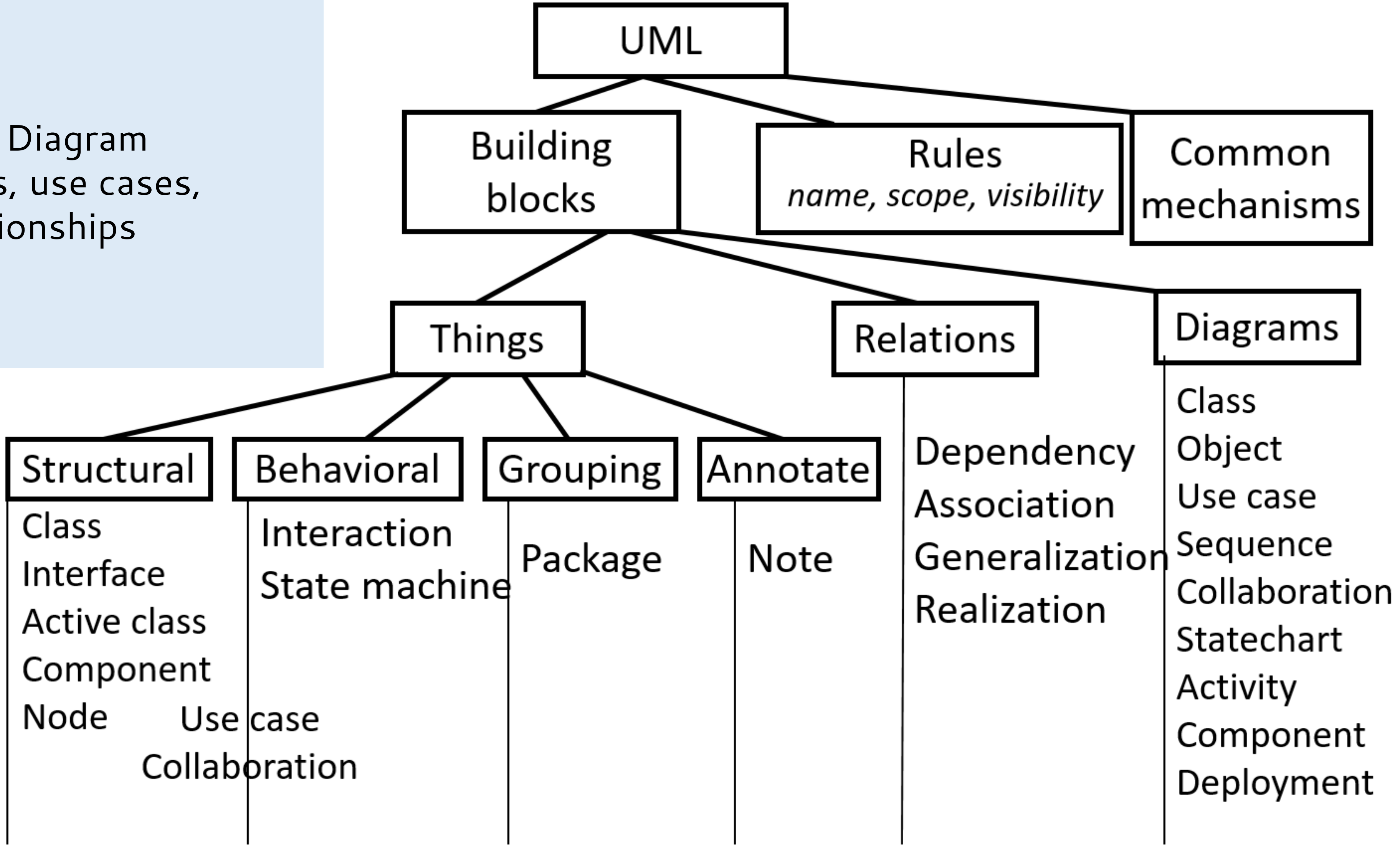
UML Use-case models are predominantly used with Modeling Systems, to discuss the dynamic behavior of the system when it is running/operating. Its often used to used to gather the requirements of a system including internal and external influences.

Unified Modelling Language – UMl



Conceptual UML model

Use Case Diagram
shows actors, use cases,
and relationships

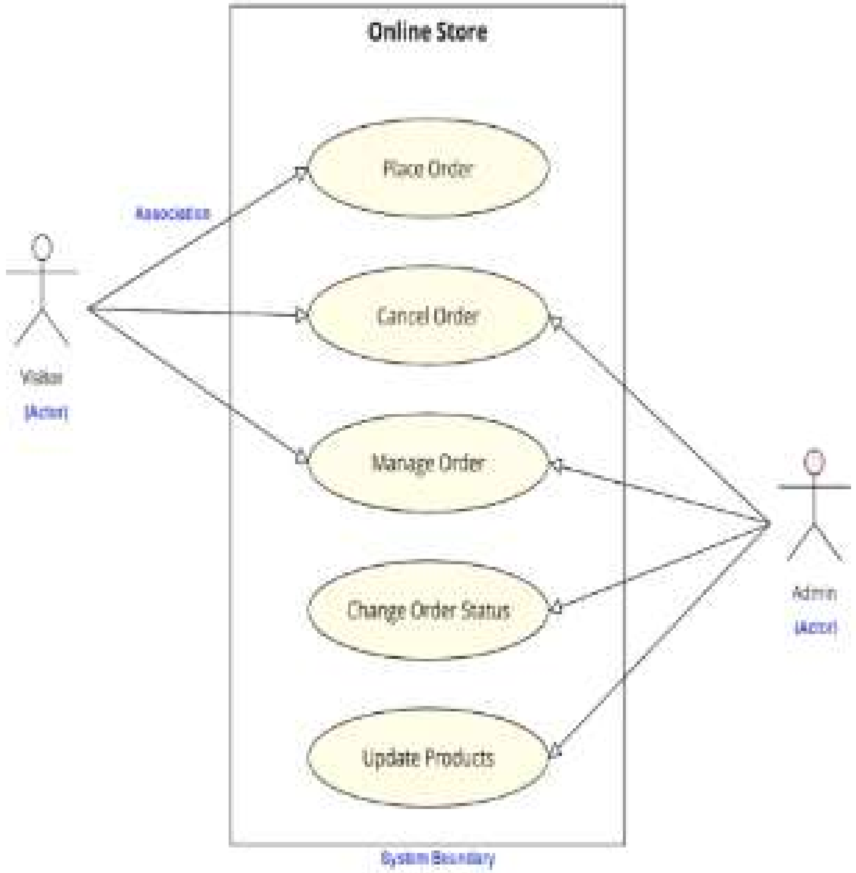


Using Use Case diagram

Use case from a user’s point of view outlines how the proposed system will perform a task expected to be performed, while responding to a request or task of a role/actor/user.

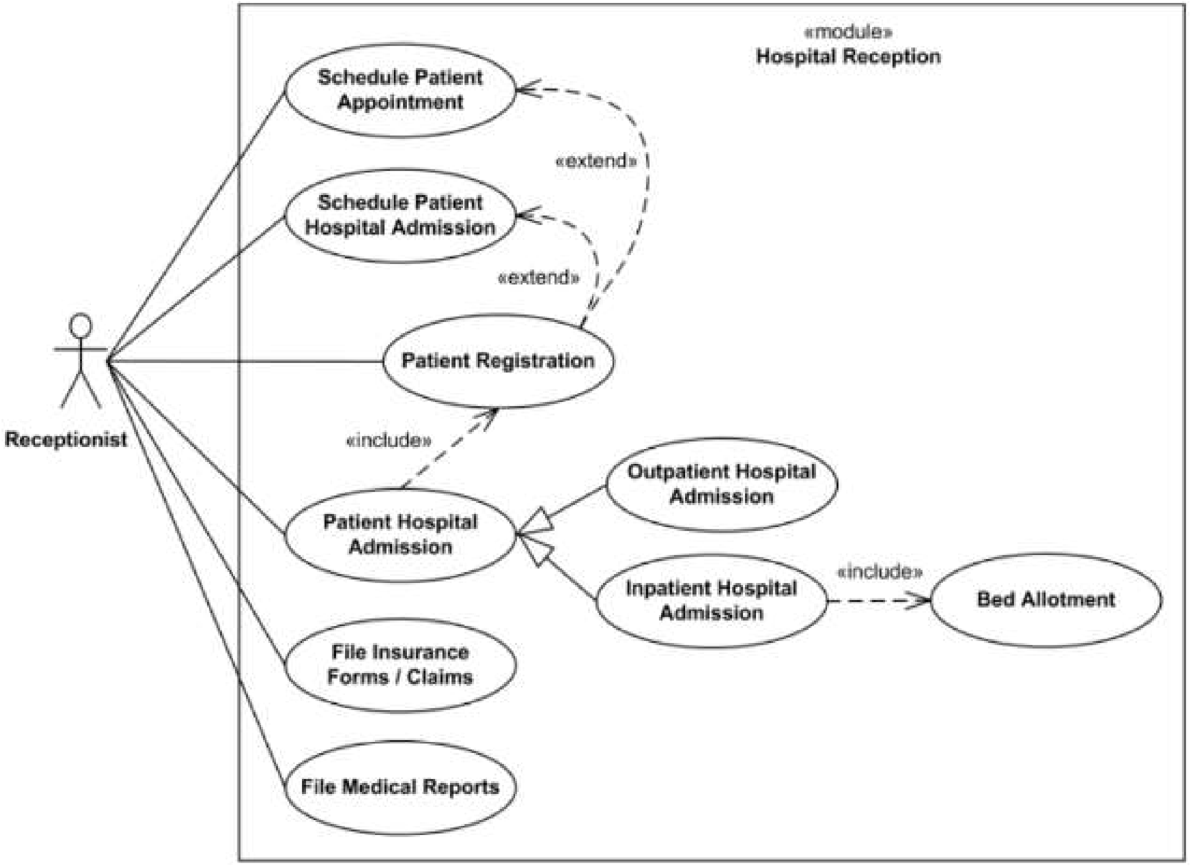
Use case diagrams are used to visualize, specify, construct, and
document the (intended) behavior of the system, during requirements capture and analysis. Used by developers, domain experts and end-users.

Actor is someone (can be a human or other external system)
interacting with the use case (system function), named by noun but is not part of the system.



Use Case diagram – In-Class Exercise

Using Use Case diagram, to depict the job of a hospital receptionist. Include scheduling appointments, admissions, bed allotment, filing insurance and filing medical reports as some of the use cases.


«module»
Hospital Reception
Schedule Patient Appointment
«extend»
Schedule Patient Hospital Admission
«extend»
Patient Registration
Receptionist
«include»
Outpatient Hospital Admission
Patient Hospital Admission
Inpatient Hospital Admission
«include»
Bed Allotment
File Insurance Forms / Claims
File Medical Reports

THANK YOU

Dr. Jayashree R

Department of Computer Science and Engineering

jayashree@pes.edu

Software Engineering

Introduction to Software Engineering

Dr. Jayashree R

Department of Computer Science and Engineering

Teaching Assistant: Apoorva B S (Sem VII)

Acknowledgement: Dr. Phalachandra HL, authors of the prescribed textbooks and materials

Software Engineering

Requirement Specification, Requirement Validation & Requirement Management

Dr. Jayashree R

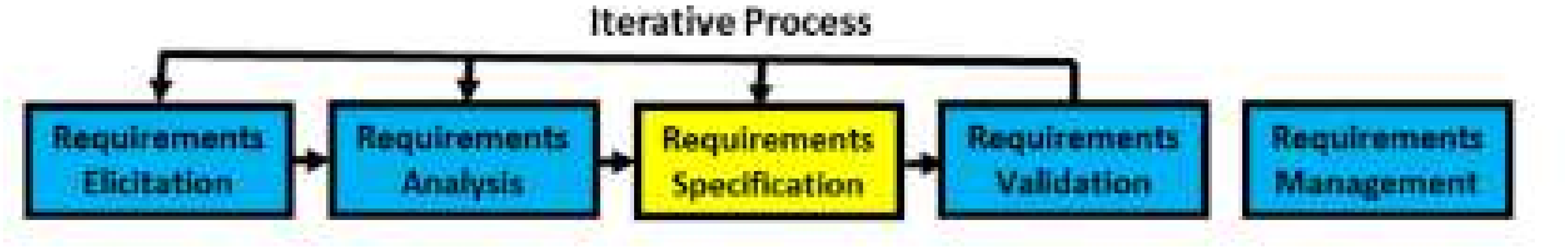
Department of Computer Science and Engineering

Requirement Specification

After elicitation and analysis, we need to specify the 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.



Documentation Characteristics

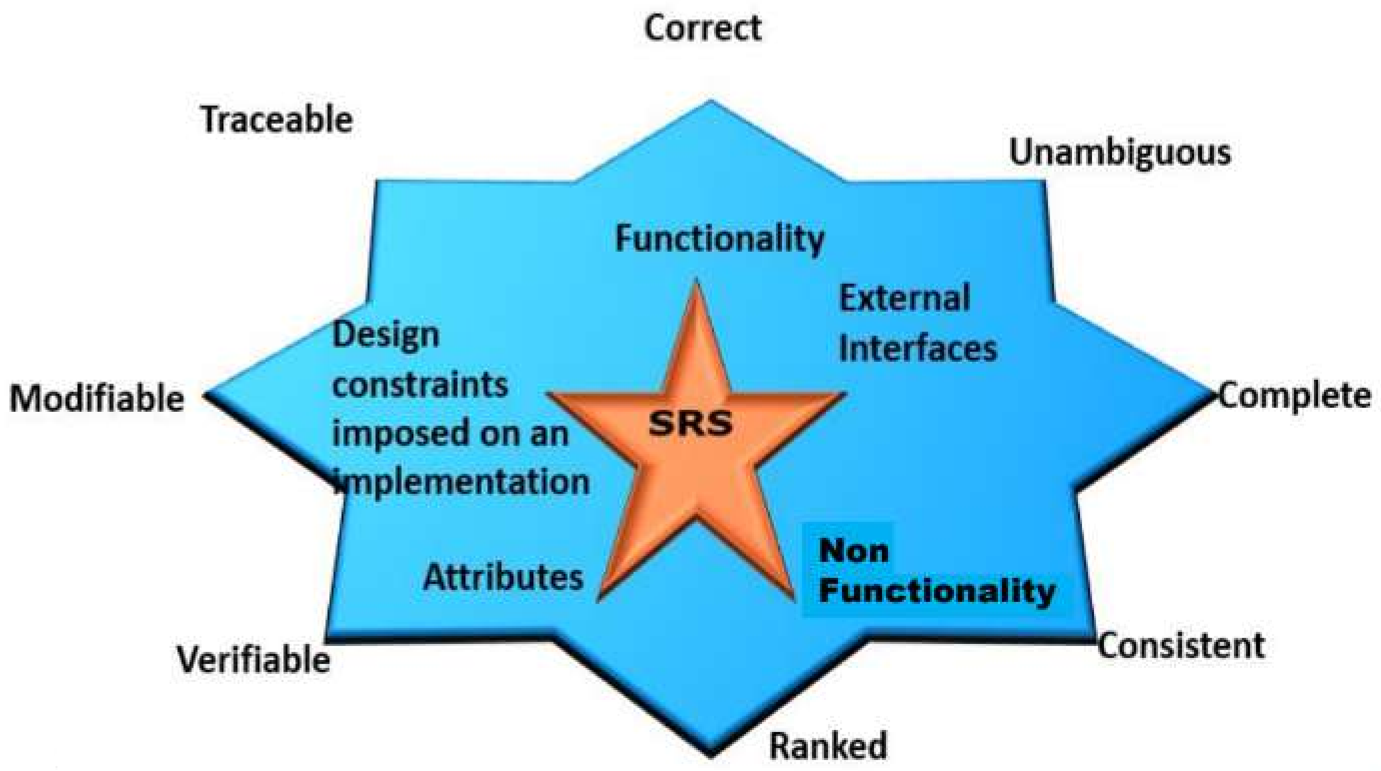
Reasons for documentation

- Visibility
- Formalization leads to better clarity
- User support
- Team communication
- Maintenance and evolution

Characteristics of documentation

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

Requirement Specification



Software Requirement 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 which drive the functionality. Example: Performance, Availability, Portability etc.

Design constraints imposed on an implementation:

- Required standards in effect
- Implementation language
- Policies for database integrity
- Resource limits
- Security
- Operating environment(s) etc.

An example of Software Requirement 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

Requirement Validation

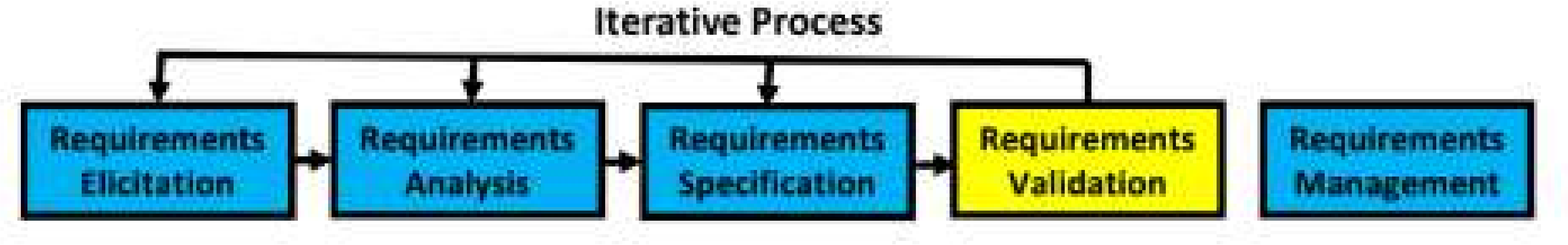
The purpose of requirements is to help ensure that the requirements does what the customer wants. This is an important phase because repairing requirement errors in downstream phases can be expensive.

VALIDATION & VERIFICATION

Validation determines whether the software requirements if implemented, will solve the right problem and satisfy the intended user needs

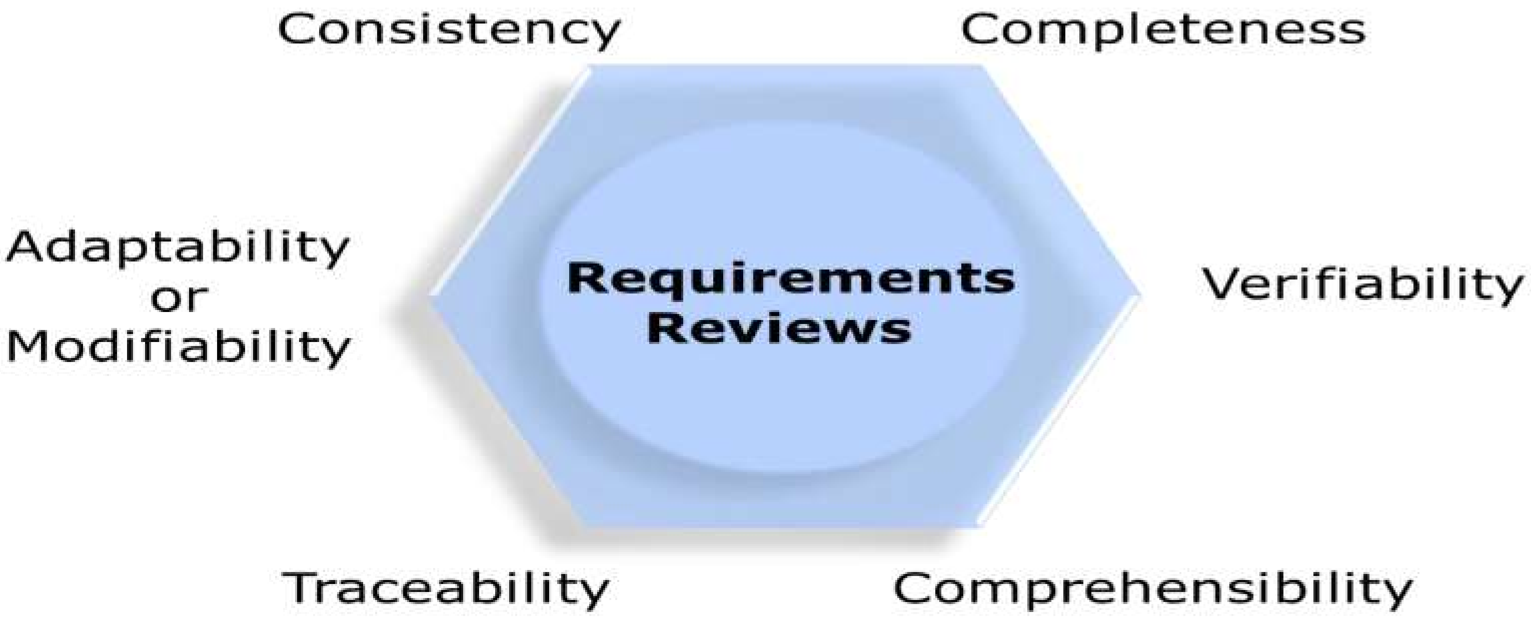
Verification determines whether the requirements have been specified correctly

(Reviews are used for both validation and verification)



Requirement Validation

Requirement Reviews



Consistency

Completeness

Adaptability
or
Modifiability

Requirements
Reviews

Verifiability

Traceability

Comprehensibility

Requirement Validation

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 – With many user interactions
Example: Design of online billing systems
Systems with little or no user interaction may not benefit as much from prototyping
Example: Batch processing

Model Validation

- Ensuring that the models represent all essential functional requirements
- Demonstrating that each model is consistent in itself
- Usage of the Fish Bone Analysis technique for validation

Acceptance Criteria

To check if there are requirements matching with that the Acceptance criteria

Requirements Management

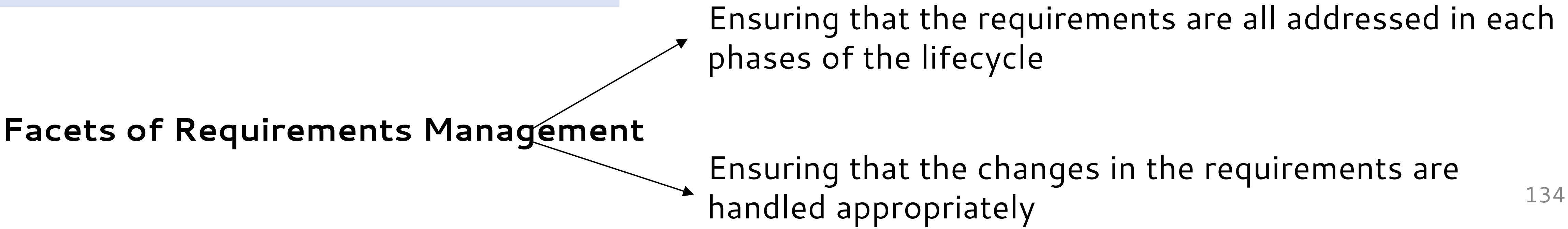
Requirements specification is the baseline on which the future lifecycle phases will need to build upon

Can you think of reasons why requirements might change?

Better understanding of the problem

Customer internalizing the problem and solution

Evolving environment and technology landscape

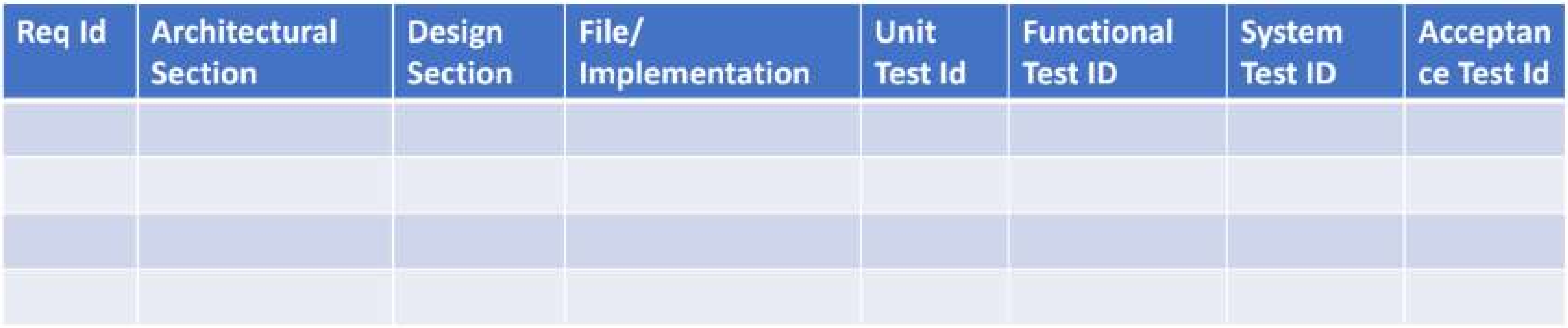


Facets of Requirements Management

Ensuring that the requirements are all addressed in each phases of the lifecycle

Ensuring that the changes in the requirements are handled appropriately

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 are traced across the SDLC using the requirement traceability matrix (RTM)

- Forward Tracing
- Backward Tracing

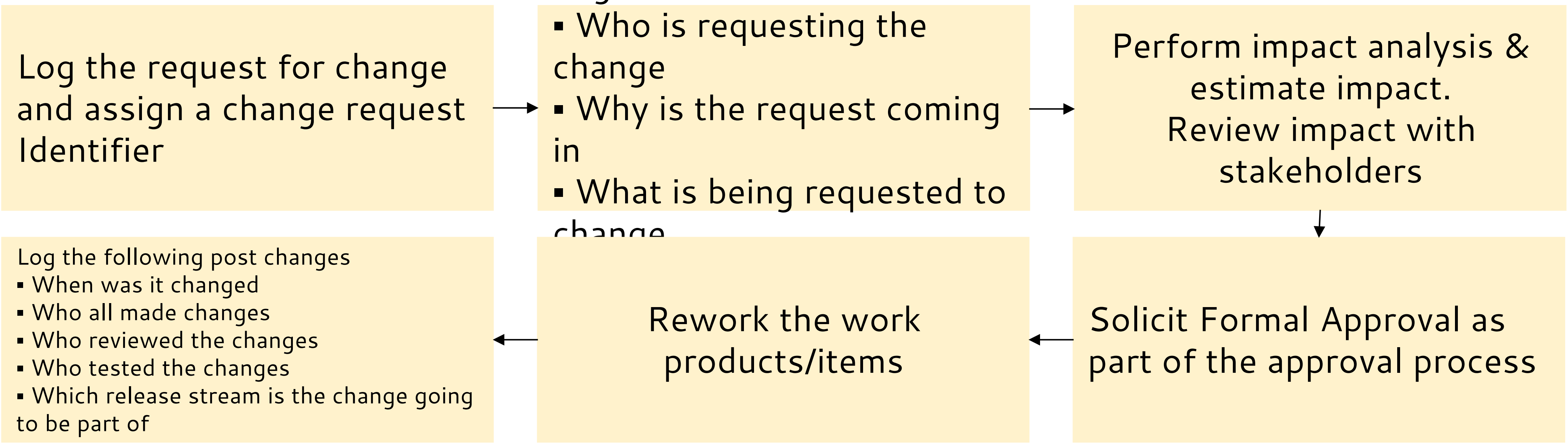
Every phase of the SDLC progressively fills the RTM

Requirement Change Management

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/
In the perspective of managing the changes, change requests go through a formal change management process.

REQUIREMENT CHANGE PROCESS



Log the request for change and assign a change request Identifier

Log
- Who is requesting the change
- Why is the request coming in
- What is being requested to change

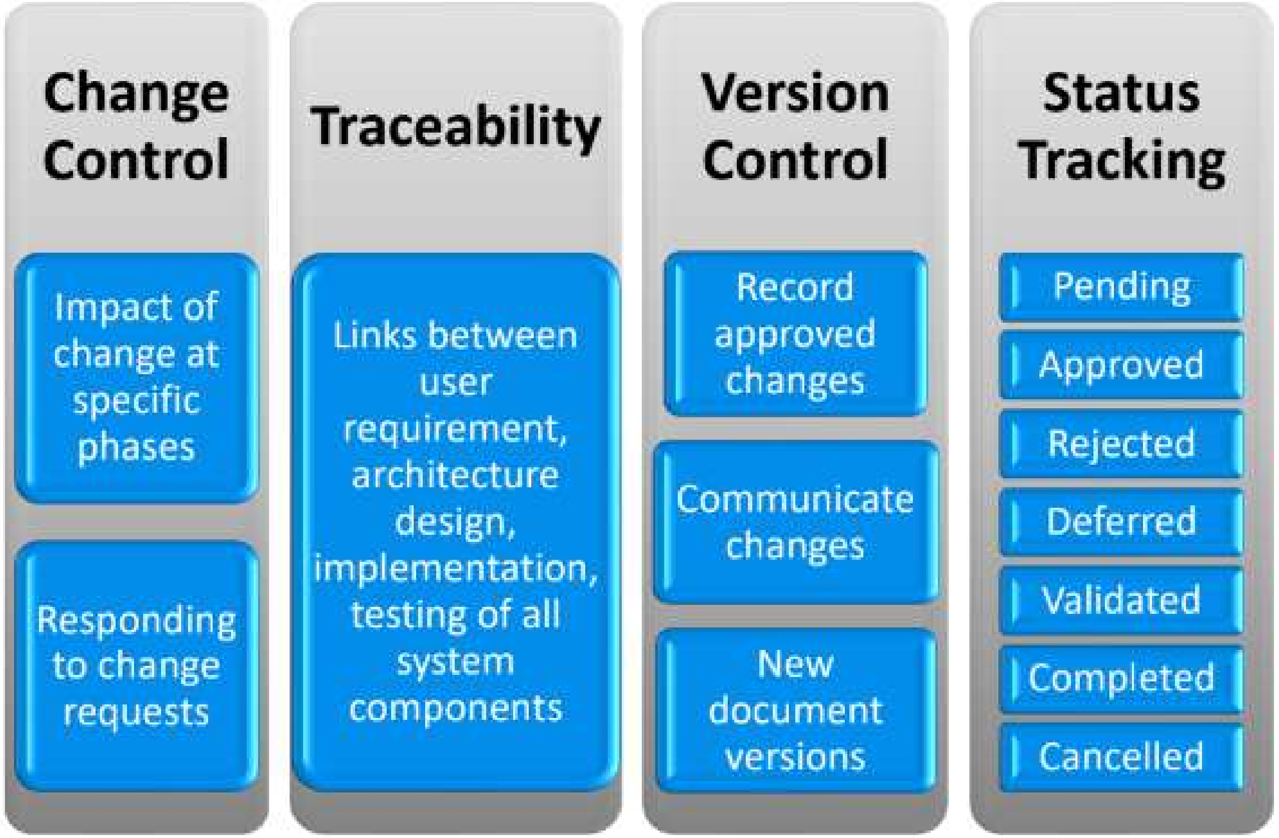
Perform impact analysis & estimate impact.
Review impact with stakeholders

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

Requirement Change Management

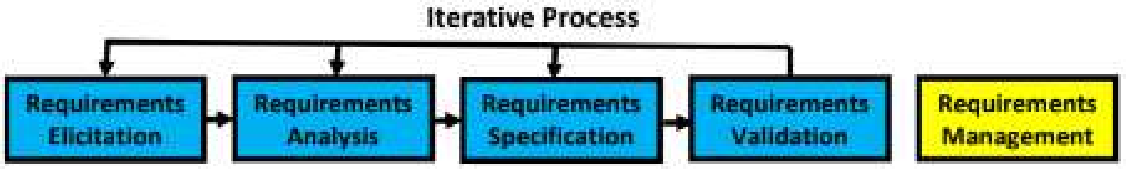


Change Control
- Impact of change at specific phases
- Responding to change requests

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



Iterative Process

Requirements Elicitation → Requirements Analysis → Requirements Specification → Requirements Validation

Requirements Management

THANK YOU

Dr. Jayashree R
Department of Computer Science and Engineering
jayashree@pes.edu