



## **UNIT 1**

# **18CSC206J -SOFTWARE ENGINEERING & PROJECT MANAGEMENT**



## Course Learning Rationale (CLR)

**CLR-1:** Familiarize the software life cycle models and software development process

**CLR-2:** Understand the various techniques for requirements, planning and managing a technology project



## Course Learning Outcomes (CLO):

**CLO-1** : Identify the process of project life cycle model and process

**CLO-2** : Analyze and specify software requirements through a productive working Relationship with project stakeholders



# What is Software?

*The product that software professionals build and then support over the long term.*

*Software encompasses:*

- (1) **instructions** (computer programs) that when executed provide desired features, function, and performance;*
- (2) **data structures** that enable the programs to adequately store and manipulate information and*
- (3) **documentation** that describes the **operation and use of the programs.***



# Features of Software?

- Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change).



# Features of Software?

- Hardware has bathtub curve of failure rate ( high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built.
- Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

# Software Applications



- 1. System software: such as compilers, editors, file management utilities
- 2. Application software: stand-alone programs for specific needs.
- 3. Engineering/scientific software: Characterized by “number crunching” algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc
- 4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- 5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- 6. Web Apps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- 7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

# Software Process



- A process is a collection of activities, actions and tasks that are performed when some work product to be created





# Software project management

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery). A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.



# Activities

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

## **The list of activities are as follows:**

1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management



# Process framework

## Why process :

A process defines who is doing what, when and how to reach a certain goal.

- To build complete software process.
- Identified a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- It encompasses a set of umbrella activities that are applicable across the entire software process.



# Process Framework

- Each framework activities is populated by a set **for *software engineering actions*** – a collection of related tasks.
- Each action has individual *work task*.

## Process framework

### Framework Activity # 1

Software Engineering action: # 1.1

work tasks:

work products:

Quality assurance points

Projects milestones

•  
•  
•

Software Engineering action: # 1.K

work tasks:

work products:

Quality assurance points

Projects milestones

## Process framework

### Framework Activity # n

Software Engineering action: # n.1

work tasks:

work products:

Quality assurance points

Projects milestones

•  
•  
•

Software Engineering action: # n.k

work tasks:

work products:

Quality assurance points

Projects milestones

# Generic Process Framework Activities



- Communication:
  - Heavy communication with customers, stakeholders, team
  - Encompasses requirements gathering and related activities
- Planning:
  - Workflow that is to follow
  - Describe technical task, likely risk, resources will require, work products to be produced and a work schedule.
- Modeling:
  - Help developer and customer to understand requirements (Analysis of requirements) & Design of software
- Construction
  - Code generation: either manual or automated or both
  - Testing – to uncover error in the code.
- Deployment:
  - Delivery to the customer for evaluation
  - Customer provide feedback

# Umbrella Activities



- Software project tracking and control
  - Assessing progress against the project plan.
  - Take adequate action to maintain schedule.
- Formal technical reviews
  - Assessing software work products in an effort to uncover and remove errors before goes into next action or activity.
- Software quality assurance
  - Define and conducts the activities required to ensure software quality.
- Software configuration management
  - Manages the effects of change.



- Document preparation and production
  - Help to create work products such as models, documents, logs, form and list.
- Reusability management
  - Define criteria for work product reuse
  - Mechanisms to achieve reusable components.
- Measurement
  - Define and collects process, project, and product measures
  - Assist the team in delivering software that meets customer's needs.
- Risk management
  - Assesses risks that may effect that outcome of project or quality of product (i.e. software)

# Software process model



- Process models prescribe a distinct set of activities, actions, tasks, milestones, and work products required to engineer high quality software.
- Process models are not perfect, but provide roadmap for software engineering work.
- Software process models are adapted to meet the needs of software engineers and managers for a specific project.



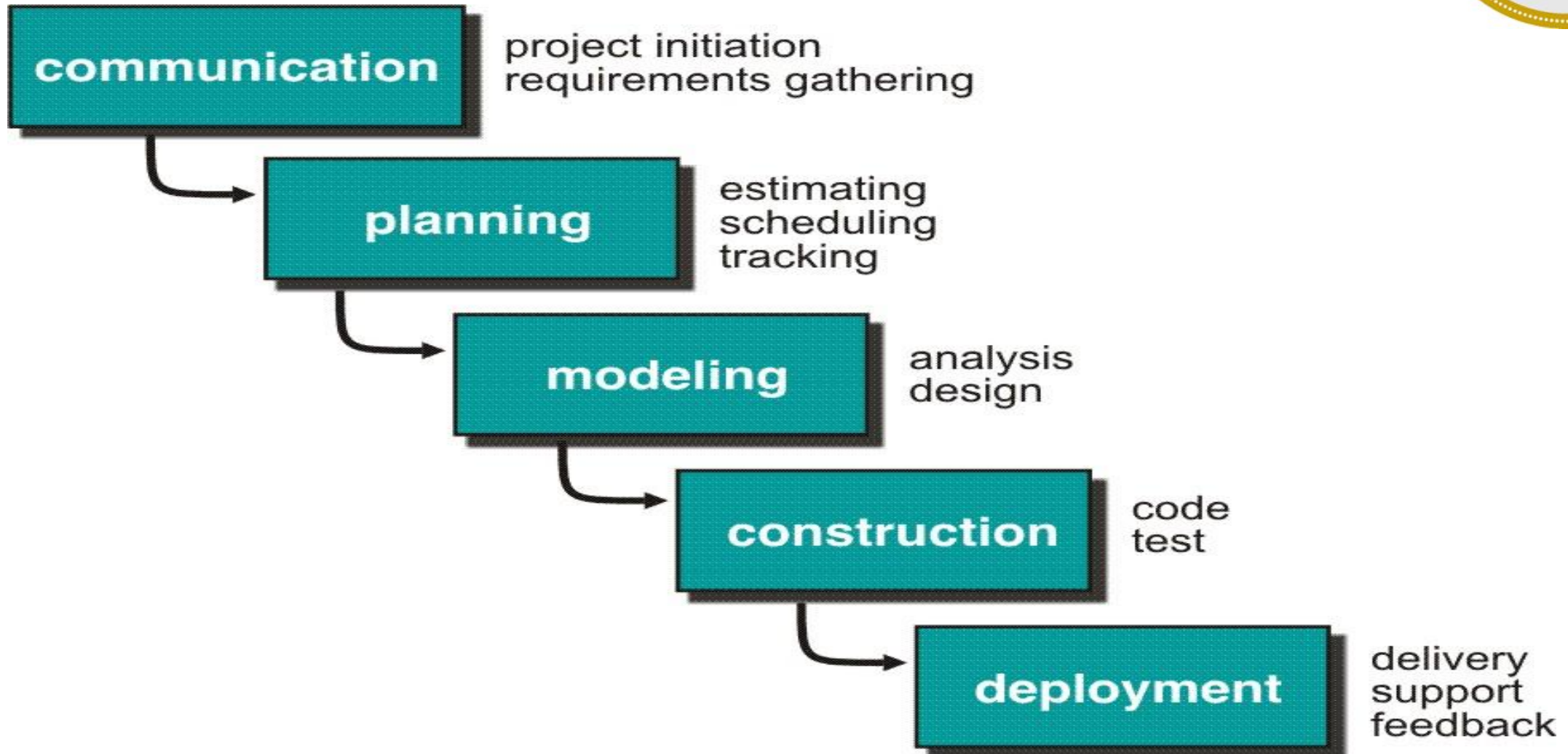


# Prescriptive Model

- Prescriptive process models advocate an orderly approach to software engineering
  - Organize framework activities in a certain order
- Process framework activity with set of software engineering actions.
- Each action in terms of a task set that identifies the work to be accomplished to meet the goals.
- The resultant process model should be adapted to accommodate the nature of the specific project, people doing the work, and the work environment.
- Software engineer choose process framework that includes activities like;
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment



# Waterfall Model or Classic Life Cycle



# Prescriptive Model



- Calling this model as “Prescribe” because it recommend a set of process elements, activities, action task, work product & quality.
- Each elements are inter related to one another (called workflow).



# Waterfall Model or Classic Life Cycle

- Requirement Analysis and Definition: What - The systems services, constraints and goals are defined by customers with system users.
- Scheduling tracking -
  - Assessing progress against the project plan.
  - Require action to maintain schedule.
- System and Software Design: How –It establishes and overall system architecture. Software design involves fundamental system abstractions and their relationships.
- Integration and system testing: The individual program unit or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- Operation and Maintenance: Normally this is the longest phase of the software life cycle. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle.

# Limitations of the waterfall model



- ❑ The nature of the requirements will not change very much During development; during evolution
- ❑ The model implies that you should attempt to complete a given stage before moving on to the next stage
  - ❑ Does not account for the fact that requirements constantly change.
  - ❑ It also means that customers can not use anything until the entire system is complete.
- ❑ The model implies that once the product is finished, everything else is maintenance.

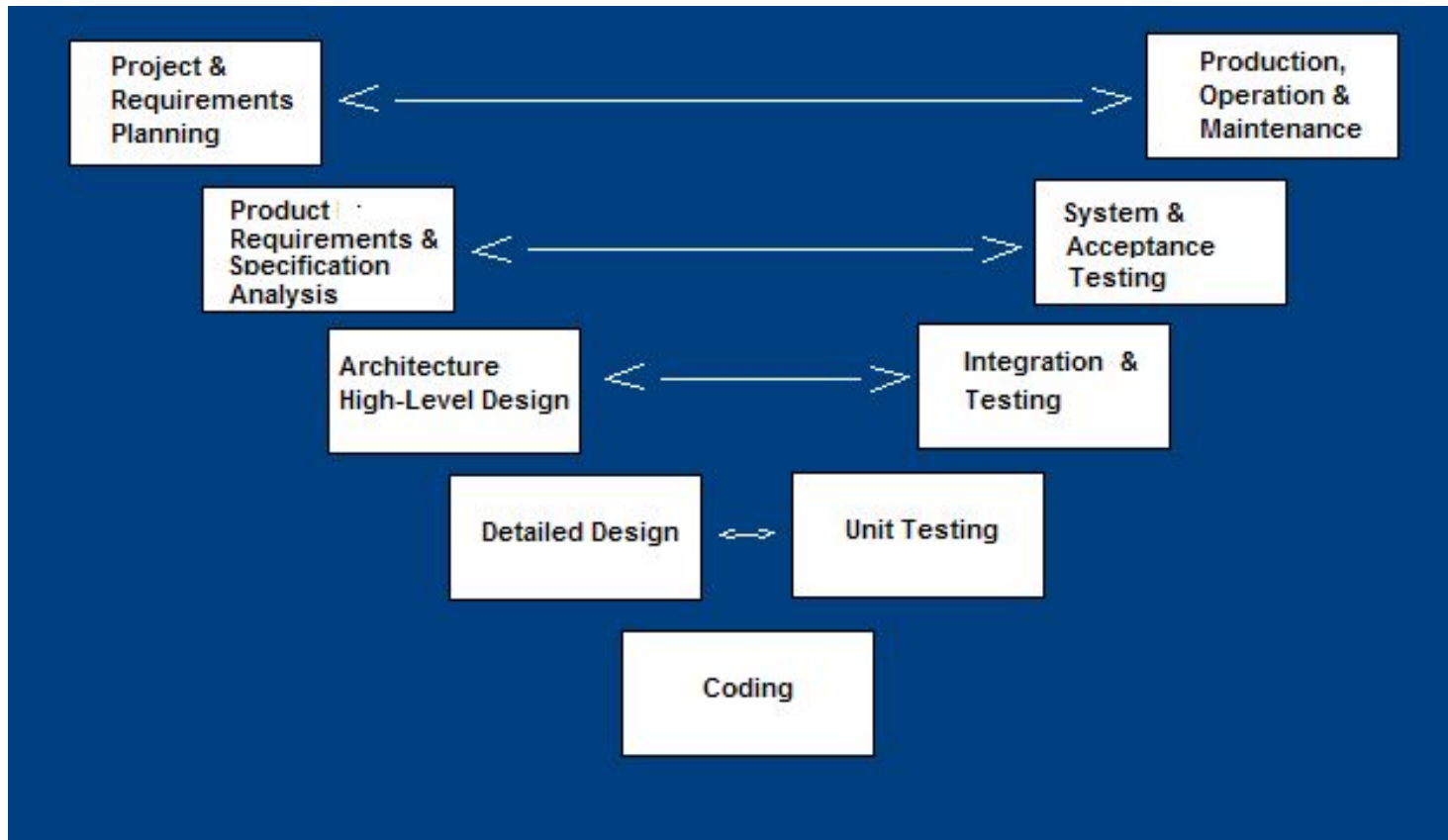


- ❑ Surprises at the end are very expensive
- ❑ Some teams sit ideal for other teams to finish
- ❑ Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Problems:
  1. Real projects are rarely follow the sequential model.
  2. Difficult for the customer to state all the requirement explicitly.
  3. Assumes patience from customer - working version of program will not available until programs not getting change fully.





# V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

# V-Shaped Steps



- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software



# V-Shaped Strengths



- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

# V-Shaped Weaknesses



- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

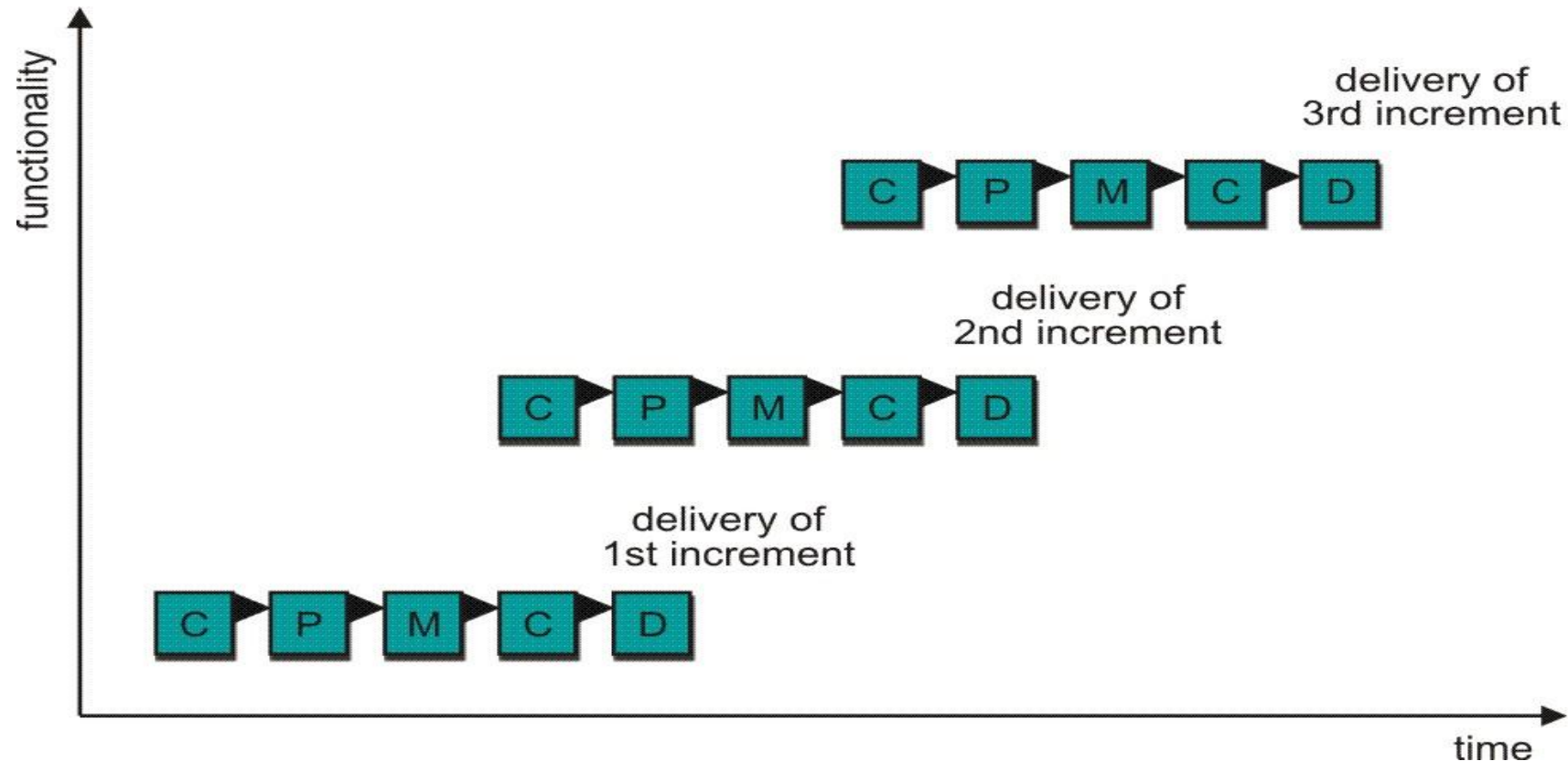


# When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- Solution and technology are known



# Incremental Process Model



**C** - Communication  
**P** - Planning  
**M** - Modeling  
**C** - Construction  
**D** - Deployment

Delivers software in small but usable pieces, each piece builds on pieces already delivered



# The Incremental Model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments increment delivering part of the required functionality.
- First Increment is often core product
  - Includes basic requirement
  - Many supplementary features (known & unknown) remain undelivered
- A plan of next increment is prepared
  - Modifications of the first increment
  - Additional features of the first increment
- It is particularly useful when enough staffing is not available for the whole project
- Increment can be planned to manage technical risks.
- Incremental model focus more on delivery of operation product with each increment.



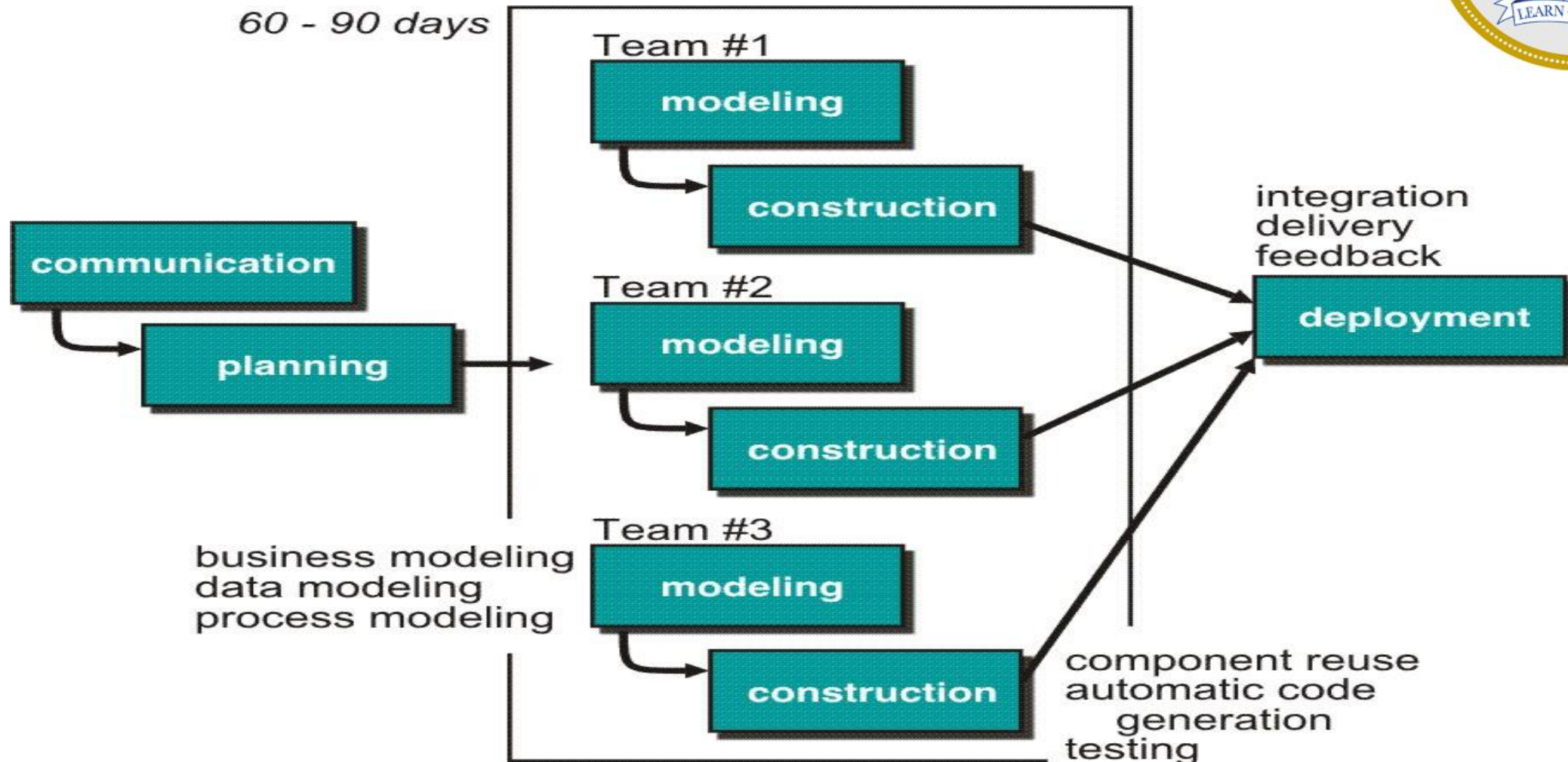
# The Incremental Model

- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.





# Rapid Application Development (RAD) Model



**Makes heavy use of reusable software components with an extremely short development cycle**



# RAD model

- Communication – To understand business problem.
- Planning – multiple s/w teams works in parallel on diff. system.
- Modeling –
  - Business modeling – Information flow among business is working.  
Ex. What kind of information drives?  
Who is going to generate information?  
From where information comes and goes?
  - Data modeling – Information refine into set of data objects that are needed to support business.
  - Process modeling – Data object transforms to information flow necessary to implement business.



# RAD Model



- If application is modularized (“Scalable Scope”), each major function to be completed in less than three months.
- Each major function can be addressed by a separate team and then integrated to form a whole.

## Drawback:

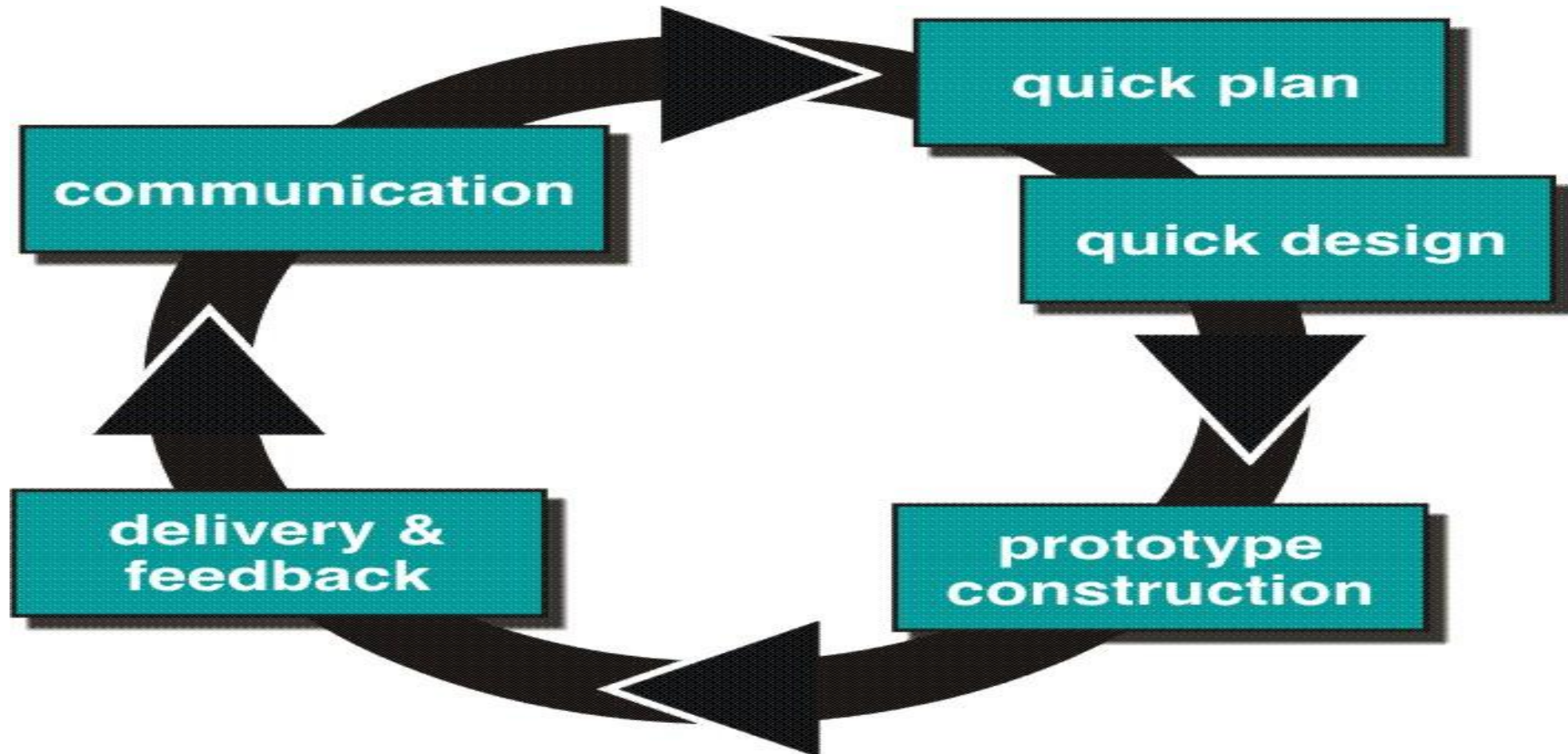
- For large but scalable projects
  - RAD requires sufficient human resources
- Projects fail if developers and customers are not committed in a much shortened time-frame
- Problematic if system can not be modularized
- Not appropriate when technical risks are high ( heavy use of new technology)

# Evolutionary Process Model



- Produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are iterative.
- Evolutionary models are:
  - Prototyping
  - Spiral Model

# Evolutionary Process Models : Prototyping





# Prototyping cohesive

- Best approach when:
  - Objectives defined by customer are general but does not have details like input, processing, or output requirement.
  - Developer may be unsure of the efficiency of an algorithm, O.S., or the form that human machine interaction should take.
- It can be used as standalone process model.
- Model assist software engineer and customer to better understand what is to be built when requirement are uncertain(Fuzzy).



- Prototyping start with communication, between a customer and software engineer to define overall objective, identify requirements and make a boundary
- Going ahead, planned quickly and modeling (software layout visible to the customers/end-user) occurs.
- Quick design leads to prototype construction.
- Prototype is deployed and evaluated by the customer/user.
- Feedback from customer/end user will refine requirement and that is how iteration occurs during prototype to satisfy the needs of the customer.

# Prototyping (cont..)



- Prototype can be serve as “the first system”.
  
- Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

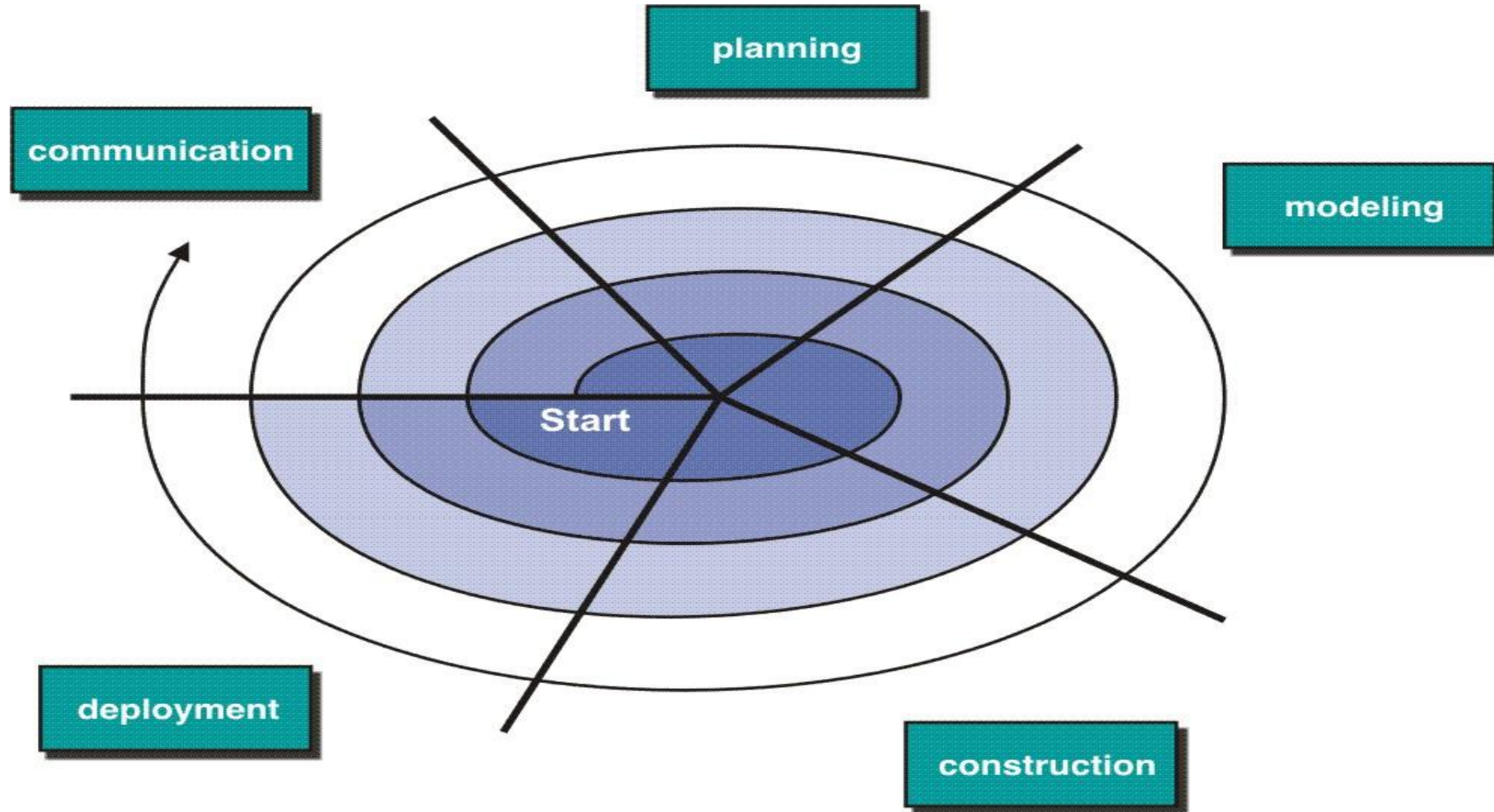




## Problem Areas:

- ❑ Customer cries foul and demand that “a few fixes” be applied to make the prototype a working product, due to that software quality suffers as a result.
- ❑ Developer often makes implementation in order to get a prototype working quickly without considering other factors in mind like OS, Programming language, etc.
- ❑ Customer and developer both must be agree that the prototype is built to serve as a mechanism for defining requirement

# Evolutionary Model: Spiral Model







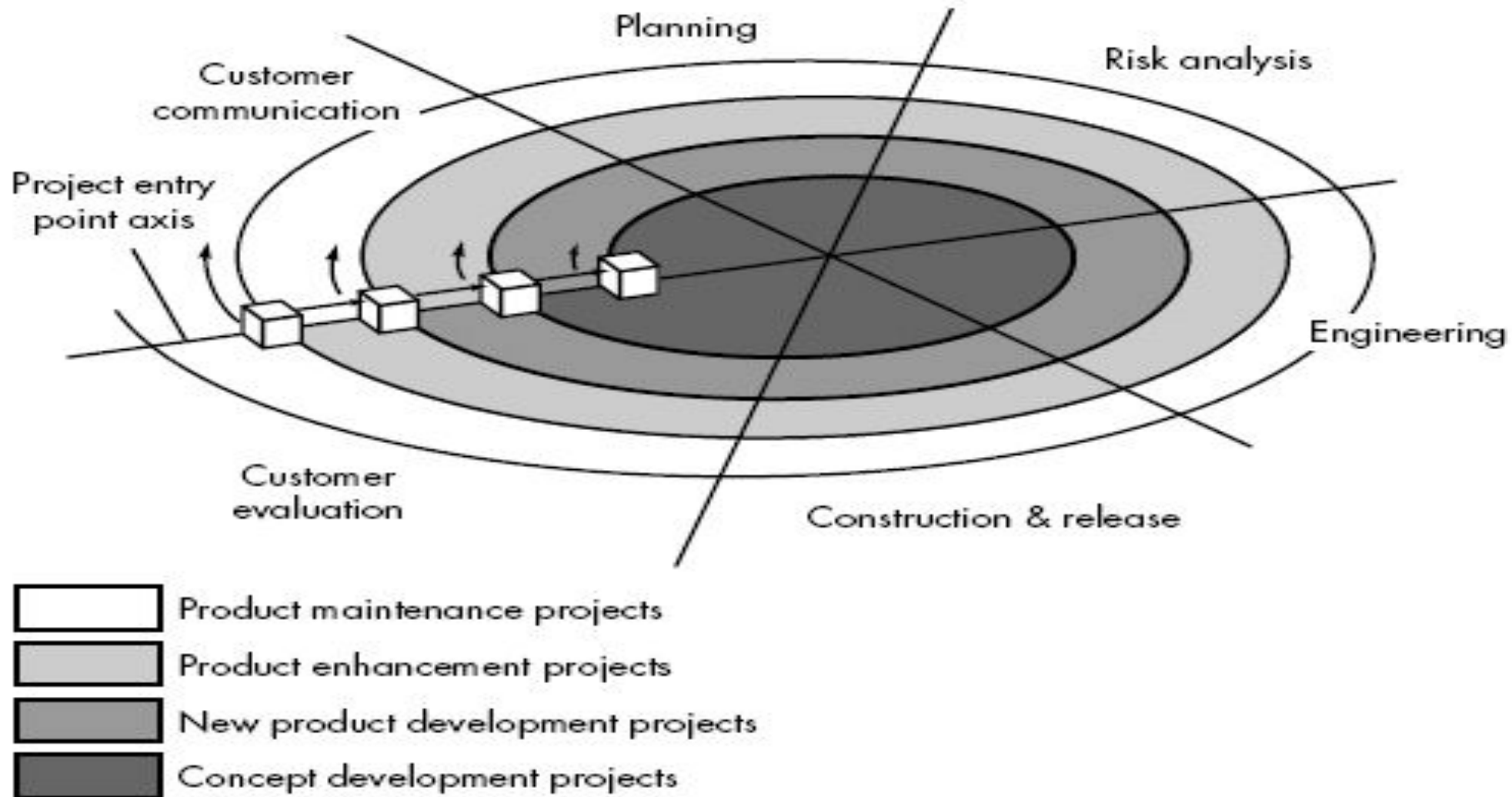
# Spiral Model

- ❑ Couples iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model
- It provide potential for rapid development of increasingly more complete version of the software.
- Using spiral, software developed in as series of evolutionary release.
  - Early iteration, release might be on paper or prototype.
  - Later iteration, more complete version of software.



- Divided into framework activities (C,P,M,C,D). Each activity represent one segment.
- Evolutionary process begins in a clockwise direction, beginning at the center risk.
- First circuit around the spiral might result in development of a product specification.
- Subsequently, develop a prototype and then progressively more sophisticated version of software.

# Spiral Model





# Spiral Model (cont.)

## Concept Development Project:

- Start at the core and continues for multiple iterations until it is complete.
- If concept is developed into an actual product, the process proceeds outward on the spiral.

## New Product Development Project:

- New product will evolve through a number of iterations around the spiral.
- Later, a circuit around spiral might be used to represent a “Product Enhancement Project”

## Product Enhancement Project:

- There are times when process is undeveloped or software team not developing new things but change is initiated, process start at appropriate entry point.



- Spiral models uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at each stage in the evolution of the product.
- It maintains the systematic stepwise approach suggested by the classic life cycle but also incorporates it into an iterative framework activity.
- If risks cannot be resolved, project is immediately terminated



# What is “Agility”?

- Effective (rapid and adaptive) response to change (team members, new technology, requirements)
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers.
- Drawing the customer into the team. Planning in an uncertain world has its limits and plan must be flexible.
- Organizing a team so that it is in control of the work performed
- Emphasize an incremental delivery strategy as opposed to intermediate products that gets working software to the customer as rapidly as feasible.



# What is “Agility”?

- Rapid, incremental delivery of software
- The development guidelines stress delivery over analysis and design although these activities are not discouraged, and active and continuous communication between developers and customers.





# The Manifesto for Agile Software Development

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*



# An Agile Process

- Is driven by customer descriptions of what is required (scenarios). Some assumptions:
  - Recognizes that plans are short-lived (some requirements will persist, some will change. Customer priorities will change)
  - Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created. )
  - Analysis, design, construction and testing are not predictable.
- Thus has to Adapt as changes occur due to unpredictability
- Delivers multiple 'software increments', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

# Agility Principles - I



1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

# Agility Principles - II



7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

# Extreme Programming (XP)



- The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.
- XP Planning
  - Begins with the listening, leads to creation of “user stories” that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a deliverable increment next release.



- A commitment (stories to be included, delivery date and other project matters) is made.
- Three ways:
  - 1. Either all stories will be implemented in a few weeks,
  - 2. high priority stories first, or
  - 3. the riskiest stories will be implemented first.
- After the first increment “project velocity”, namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments.
- Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.



# Extreme Programming (XP)

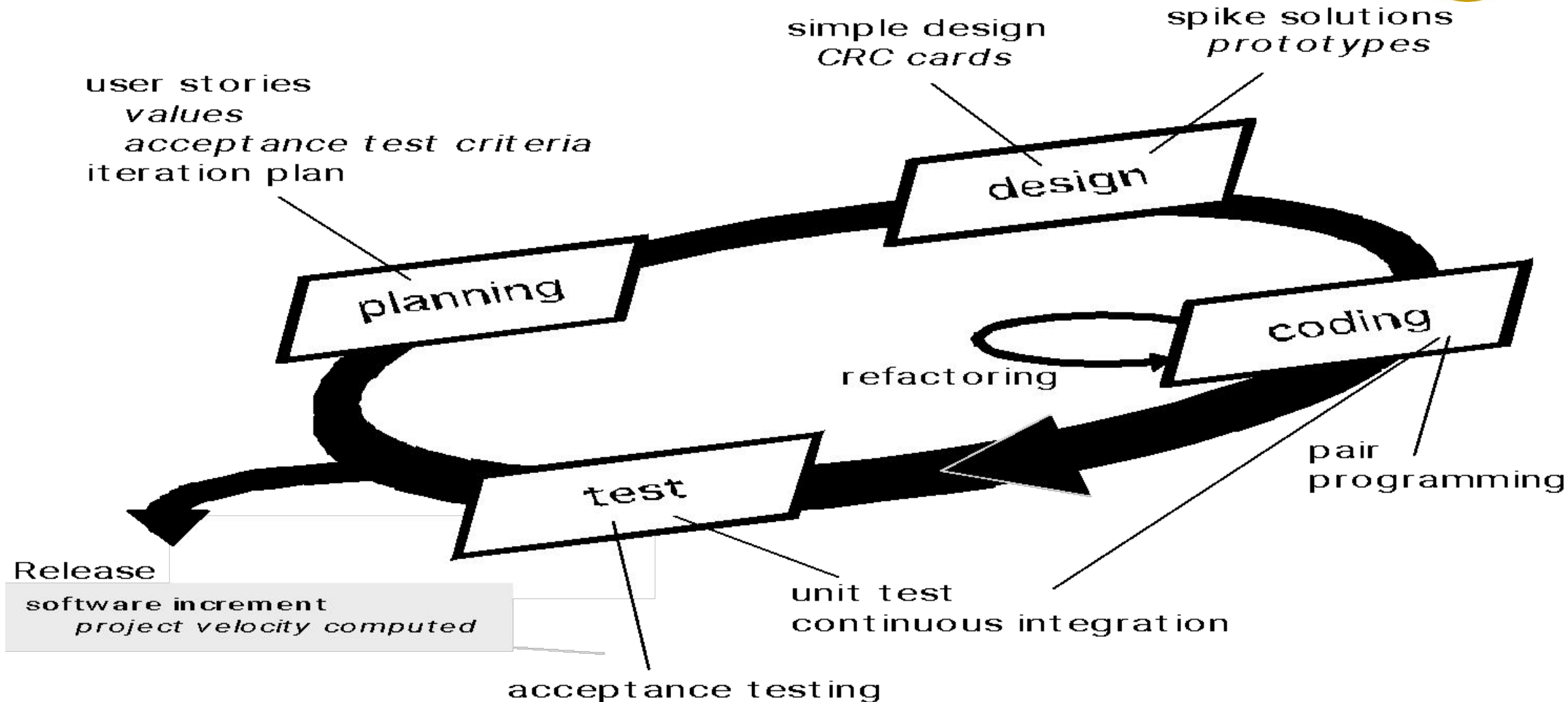
- XP Design ( occurs both before and after coding as refactoring is encouraged)
  - Follows the KIS principle (keep it simple) Nothing more nothing less than the story.
  - Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context. The only design work product of XP.
  - Encourages “refactoring”—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read.





- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences. So implementer can focus on what must be implemented to pass the test.
  - Encourages “pair programming”. Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles.
- XP Testing
  - All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components.
  - “Acceptance tests” are defined by the customer and executed to assess customer visible functionality

# Extreme Programming (XP)





# SCRUM

- Scrum's unique importance among the methods its strong promotion of self-directed teams
- Daily team measurement
- Avoidance of prescriptive process
- Demo to external stakeholders at end of each iteration
- each iteration, client-driven adaptive planning



Some key practices include:

- self-directed and self-organizing team
- no external addition of work to an iteration, once chosen
- daily stand-up meeting with special questions
- usually 30-calendar day iterations

## Scrum has been used by:

- Microsoft
- Yahoo
- Google
- Electronic Arts
- Lockheed Martin
- Philips
- Siemens
- Nokia
- IBM
- Capital One
- BBC
- Intuit
- Nielsen Media
- First American Real Estate
- BMC Software
- Ipswitch
- John Deere
- Lexis Nexis
- Sabre
- Salesforce.com
- Time Warner
- Turner Broadcasting
- Océ



## Scrum has been used for:

- Commercial software
- In-house development
- Contract development
- Fixed-price projects
- Financial applications
- ISO 9001-certified applications
- Embedded systems
- 24x7 systems with 99.999% uptime requirements
- the Joint Strike Fighter
- Video game development
- FDA-approved, life-critical systems
- Satellite-control software
- Websites
- Handheld software
- Mobile phones
- Network switching applications
- ISV applications
- Some of the largest applications in use



# REQUIREMENT ENGINEERING

## INTRODUCTION



# What is Requirement?



It is a statement describing either

- an aspect of what the proposed system must do, or
- a constraint on the system's development.

In either case it must contribute in some way towards adequately solving the customer's problem;

- the set of requirements as a whole represents a negotiated agreement among the stakeholders.
- A collection of requirements is a *requirements document*.

# What are Requirements?



- Requirements are defined during the early stages of a system development as a specification of what should be implemented or as a constraint of some kind on the system.
- They may be:
  - a user-level facility description,
  - a detailed specification of expected system behaviour,
  - a general system property,
  - a specific constraint on the system,
  - information on how to carry out some computation,
  - a constraint on the development of the system.

# Requirements

- Brainstorming
- Document Analysis
- Focus Groups
- Interface Analysis
- Interviews
- Models
- Manuals
- Observations
- Prototyping
- Reverse Engineering
- Surveys
- Workshops

## Elicitation

- Domain Models
- Use Cases
- User Stories
- Process Models
- Interface Designs
- Workflow Models
- Business Rules
- Metrics Dictionary
- Business Glossary
- Data Dictionary
- Risk Assessment
- Value Mapping

## Analysis

- Requirements Document
- Technical Requirements
- Non-Technical Requirements
- Requirements Attributes
- Prioritisation Matrix
- Risk Management Plan
- Change Management Plan

## Specification

- Quality Review
- Peer Review
- Customer Review
- IT Review
- Project Sponsor
- Phase Gate
- Requirements Presentation

## Validation

# Types of Requirement



## User requirements

- Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers. provides and its operational constraints.

## System requirements

- 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.



## Types of Requirement

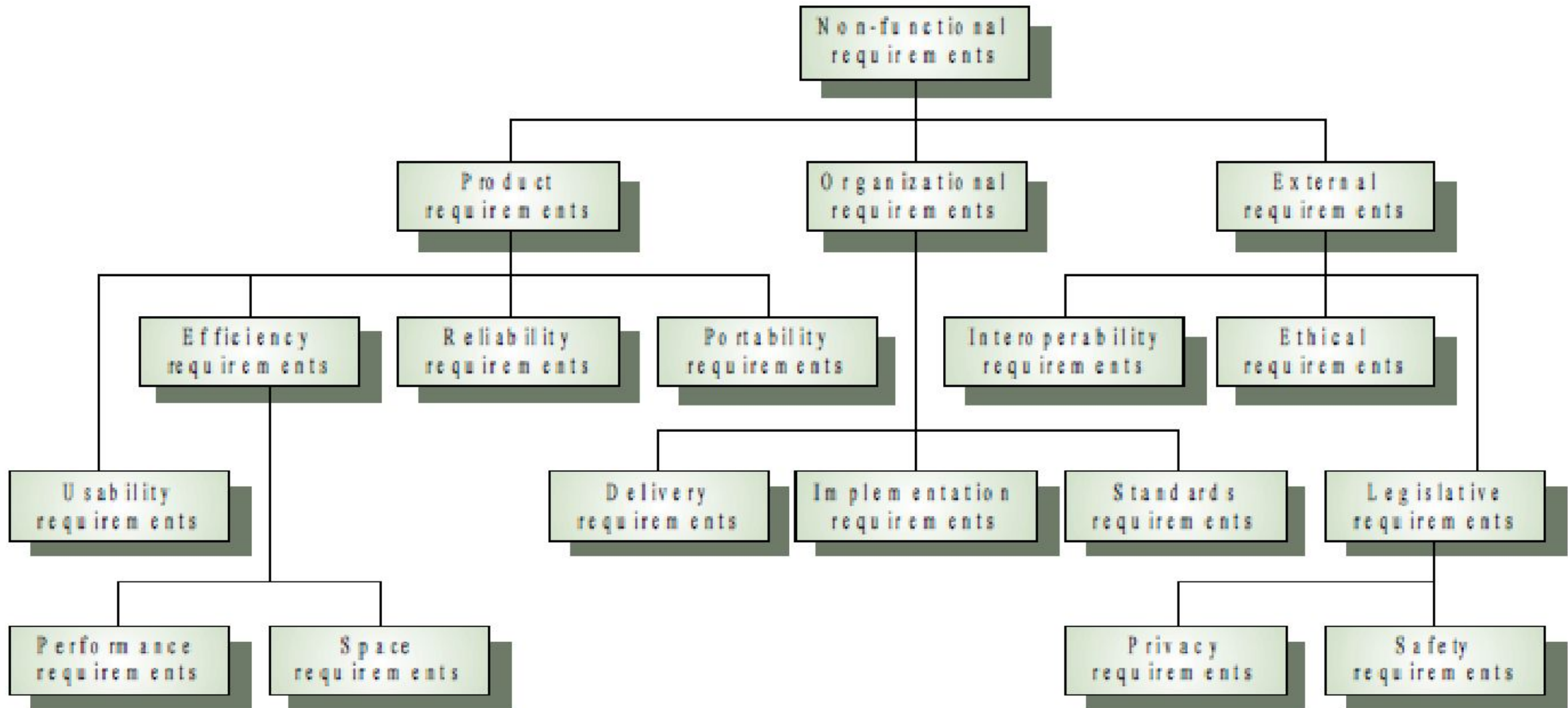
- Functional requirements:
  - Services the system should provide
  - What the system should do or not in reaction to particular situations
  - Example: “If a patient is known to be allergic to a particular medication, then prescription of that medication shall result in a warning message being issued to the prescriber”
- Non-functional requirements:
  - Constraints on the services or functions offered by the system
  - Example: “The system shall be available to all clinics during normal working hours (Mon-Fri, 0830-1730). Downtime during normal working hours shall not exceed 5 seconds in any one day”



# Functional vs. Non-Functional Requirements

Functional Requirements	Non-functional Requirements
<ul style="list-style-type: none"><li>• <u>Products</u> The system <b>shall display</b> a list of all products offered by the shop. <i>MustHave</i> The system <b>shall organise</b> the list of products by product category. <i>MustHave</i> The system <b>shall display</b> detailed product descriptions consisting of name, photograph, price and text of description on demand. <i>MustHave</i> The system shall allow the items in the catalogue to be <b>searched</b>. <i>ShouldHave</i>. The system <b>shall display</b> the number of items currently in the shopping basket on each page of the catalogue. <i>CouldHave</i></li><li>• <u>Payment</u> The system <b>shall accept</b> all major credit cards. <i>MustHave</i> The system <b>shall validate</b> payment with the credit card processing company. <i>MustHave</i></li></ul>	<ul style="list-style-type: none"><li>• <u>Capacity</u> The system shall support 1000 transactions per day. <i>ShouldHave</i> The system shall support a peak transaction rate of 10 transactions per second. <i>ShouldHave</i> The system shall support 5000 concurrent sessions. <i>MustHave</i></li><li>• <u>Availability</u> The system shall be available 24 hours per day, 360 days per year. <i>MustHave</i> The system shall not lose any transaction data. <i>MustHave</i> The system shall accept payment and raise an order within 5 seconds in 95% of the cases. <i>ShouldHave</i> The system shall log in a customer within 5 seconds. <i>ShouldHave</i></li></ul>

# Types of Non-functional Requirements







# Elicitation Techniques

## Elicitation techniques

- Stakeholder analysis
- Analysis of existing systems or documentation, background reading
- Task observation, ethnography
- Questionnaires
- Interviewing
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Pilot system
- Use cases and scenarios
- Risk analysis

# Data-Gathering Techniques<sup>[1]</sup>



Technique	Good for	Kind of data	Plus	Minus
<b>Questionnaires</b>	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
<b>Interviews</b>	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
<b>Focus groups and workshops</b>	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
<b>Naturalistic observation</b>	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
<b>Studying documentation</b>	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

[1] Preece, Rogers, and Sharp “Interaction Design: Beyond human-computer interaction”, p214



# Software Project Effort and Cost estimation & Project Planning



The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.

Schedule: Simply means the amount of time required for the completion of the job



Boehm's definition of organic, semidetached, and embedded systems:

Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience the problem



## Semi Detached

The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. E.g.: Compilers or different Embedded Systems can be considered of Semi-Detached type.

Embedded – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.



Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model

2. Intermediate COCOMO Model





- The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.
- Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases



## Estimation of Effort: Calculations –

### 1. Basic Model –

$$E = a(KLOC)^b$$

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

SOFTWARE PROJECTS	A	B
Organic	2.4	1.05
Semi Detached	3.0	1.12
Embedded	3.6	1.20



- The **effort is measured in Person-Months** and as evident from the formula is dependent on Kilo-Lines of code.
- These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.



**Intermediate Model** –The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system.

However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the

**Intermediate Model utilizes 15 such drivers for cost estimation.**



Classification of Cost Drivers and their attributes:

**(i) Product attributes –**

Required software reliability extent

Size of the application database

The complexity of the product

**(ii) Hardware attributes –**

Run-time performance constraints

Memory constraints

The volatility of the virtual machine environment

Required turnabout time



### **(iii) Personnel attributes –**

Analyst capability

Software engineering capability

Applications experience

Virtual machine experience

Programming language experience

### **(iv) Project attributes –**

Use of software tools

Application of software engineering methods

Required development schedule



### **(iii) Personnel attributes –**

Analyst capability

Software engineering capability

Applications experience

Virtual machine experience

Programming language experience

### **(iv) Project attributes –**

Use of software tools

Application of software engineering methods

Required development schedule



COST DRIVERS	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH
<b>Product Attributes</b>					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
<b>Hardware Attributes</b>					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15

**Personnel attributes**

Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	

**Project Attributes**

Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10



- ❖ The project manager is to rate these 15 different parameters for a particular project on a scale of one to three.

Then, depending on these ratings, appropriate cost driver values are taken from the above table.

These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

$$E = (a(KLOC)^b) * EAF$$

The values of a and b in case of the intermediate model are as follows:

SOFTWARE PROJECTS	A	B
Organic	3.2	1.05
Semi Detached	3.0	1.12
Embeddedc	2.8	1.20

**Example1:** Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

**Solution:** The basic COCOMO equation takes the form:

$$\text{Effort} = a_1 * (\text{KLOC})^{a_2} \text{ PM}$$

$$\text{Tdev} = b_1 * (\text{efforts})^{b_2} \text{ Months}$$

$$\text{Estimated Size of project} = 400 \text{ KLOC}$$

**(i)Organic Mode**

$$E = 2.4 * (400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5 * (1295.31)^{0.38} = 38.07 \text{ PM}$$

**(ii)Semidetached Mode**

$$E = 3.0 * (400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5 * (2462.79)^{0.35} = 38.45 \text{ PM}$$

**(iii) Embedded Mode**

$$E = 3.6 * (400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5 * (4772.8)^{0.32} = 38 \text{ PM}$$

**Example2:**

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

**Solution:** The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence  $E = 3.0(200)^{1.12} = 1133.12 \text{ PM}$   
 $D = 2.5(1133.12)^{0.35} = 29.3 \text{ PM}$

$$\begin{aligned}\text{Average Staff Size (SS)} &= \frac{E}{D} \text{ Persons} \\ &= \frac{1133.12}{29.3} = 38.67 \text{ Persons}\end{aligned}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

$$P = 176 \text{ LOC/PM}$$



# Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

*So the end result gets done on time, with quality!*

# Project Planning Task Set-I

---

- Establish project scope
- Determine feasibility
- Analyze risks
  - Risk analysis is considered in detail in Chapter 25.
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources



# Estimation

---

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - experience
  - access to good historical information (metrics)
  - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

**Project Scope**  
**Estimates**  
**Risks**  
**Schedule**  
**Control strategy**

A large blue arrow points from the left box to the right box, indicating a flow or relationship between the two.

**Software  
Project  
Plan**

# To Understand Scope ...

---

- Understand the customers needs
- understand the business context
- understand the project boundaries
- understand the customer's motivation
- understand the likely paths for change



# What is Scope?

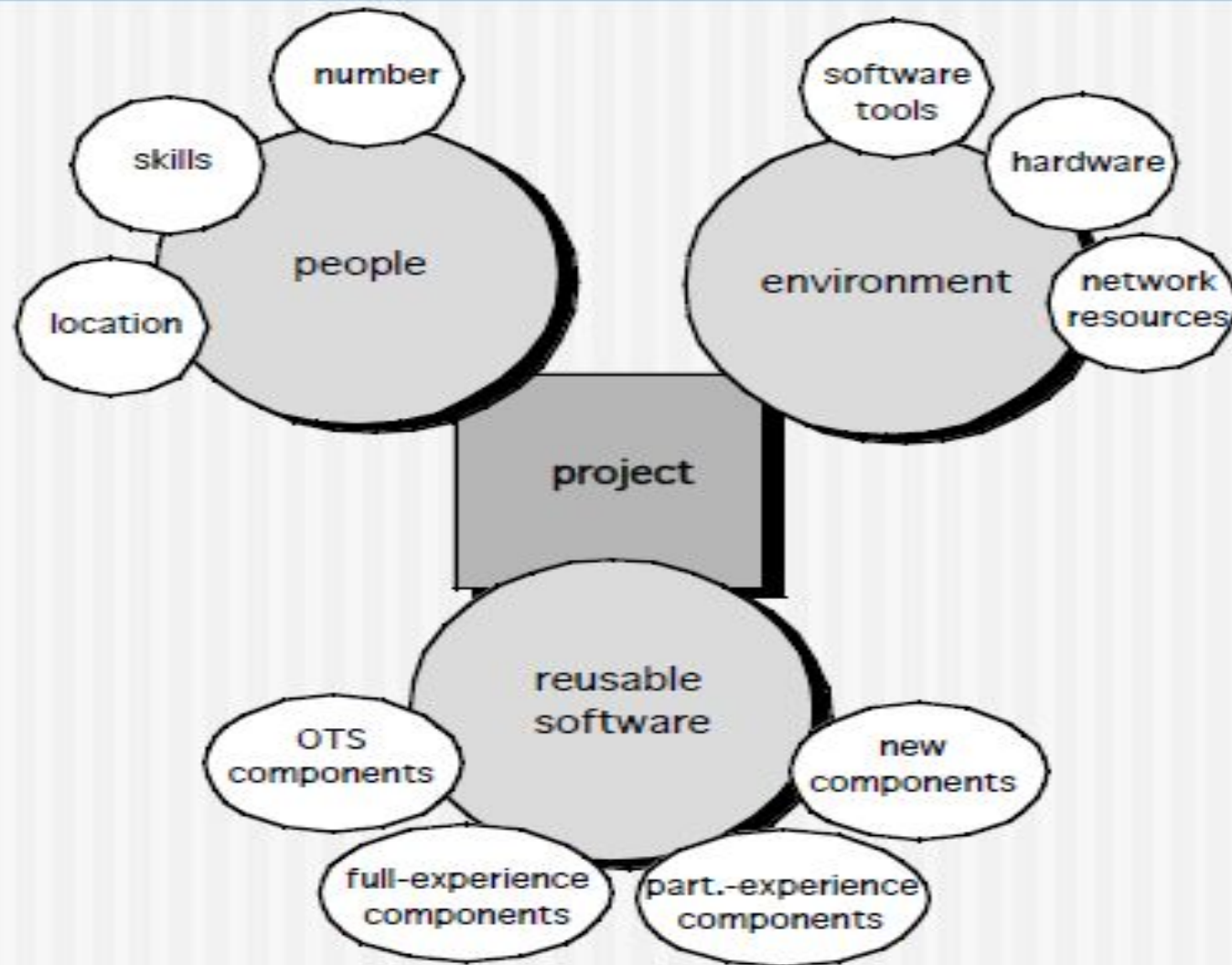
---

- *Software scope* describes
  - the functions and features that are to be delivered to end-users
  - the data that are input and output
  - the “content” that is presented to users as a consequence of using the software
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use-cases is developed by end-users.



# Resources

---



# Project Estimation

---



- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process



# Work Breakdown Structures

# Work Breakdown Structures



- The development of a work breakdown structure is dependent on the project management style, organizational culture, customer preference, financial constraints and several other hard-to-define parameters.
- A WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks.
  - A WBS provides the following information structure:
  - A delineation of all significant work
  - A clear task decomposition for assignment of responsibilities
  - A framework for scheduling, budgeting, and expenditure tracking



- Two simple planning guidelines should be considered when a project plan is being initiated or assessed.

FIRST-LEVEL WBS ELEMENT	DEFAULT BUDGET
Management	10%
Environment	10%
Requirements	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

*The first guideline prescribes a default allocation of costs among the first-level WBS elements*

DOMAIN	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

*The second guideline prescribes the allocation of effort and schedule across the life-cycle phases*



- Project plans need to be derived from two perspectives.

- *Forward-looking:*

1. The software project manager develops a characterization of the overall size, process, environment, people, and quality required for the project
2. A macro-level estimate of the total effort and schedule is developed using a software cost estimation model
3. The software project manager partitions the estimate for the effort into a top-level WBS, also partitions the schedule into major milestone dates and partitions the effort into a staffing profile
4. At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

- *Backward-looking:*

1. The lowest level WBS elements are elaborated into detailed tasks, for which budgets and schedules are estimated by the responsible WBS element manager.
2. Estimates are combined and integrated into higher level budgets and milestones.
3. Comparisons are made with the top-down budgets and schedule milestones. Gross differences are assessed and adjustments are made in order to converge on agreement between the top-down and the bottom-up estimates.



# Risk Management

## Definition of Risk

- A risk is a potential problem – it might happen and it might not

## Conceptual definition of risk

- Risk concerns future happenings
- Risk involves change in mind, opinion, actions, places, etc.
- Risk involves choice and the uncertainty that choice entails

## Two characteristics of risk

- **Uncertainty:** the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)
- **Loss:** the risk becomes a reality and unwanted consequences or losses occur

# **Risk Categorization: – Approach #1**

## **Project risks**

- They threaten the project plan
- If they become real, it is likely that the project schedule will slip and that costs will increase

## **Technical risks**

- They threaten the quality and timeliness of the software to be produced
- If they become real, implementation may become difficult or impossible

## **Business risks**

- They threaten the viability of the software to be built
- If they become real, they jeopardize the project or the product

# **Risk Categorization: – Approach #1 (Contd...)**

Sub-categories of Business risks

## **Market risk**

- building an excellent product or system that no one really wants

## **Strategic risk**

- building a product that no longer fits into the overall business strategy for the company

## **Sales risk**

- building a product that the sales force doesn't understand how to sell

## **Management risk**

- losing the support of senior management due to a change in focus or a change in people

## **Budget risk**

- losing budgetary or personnel commitment

# **Risk Categorization: – Approach #2 (Contd...)**

## **Known risks**

- Those risks that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date)

## **Predictable risks**

- Those risks that are extrapolated from past project experience (e.g., past turnover)

## **Unpredictable risks**

- Those risks that can and do occur, but are extremely difficult to identify in advance

# Reactive vs. Proactive Risk Strategies

## Reactive risk strategies

- "Don't worry, I'll think of something"
- The majority of software teams and managers rely on this approach
- Nothing is done about risks until something goes wrong
  - The team then flies into action in an attempt to correct the problem rapidly (fire fighting)
- Crisis management is the choice of management techniques

## Proactive risk strategies

- Steps for risk management are followed (see next slide)
- Primary objective is to avoid risk and to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner

## **Steps for Risk Management**

- 1) Identify possible risks; recognize what can go wrong
- 2) Analyze each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur
- 3) Rank the risks by probability and impact - Impact may be negligible, marginal, critical, and catastrophic
- 4) Develop a contingency plan to manage those risks having high probability and high impact

## **Risk Identification:**

Risk identification is a systematic attempt to specify threats to the project plan by identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary

### **Generic risks**

- Risks that are a potential threat to every software project

### **Product-specific risks**

- Risks that can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built
- This requires examination of the project plan and the statement of scope
- "What special characteristics of this product may threaten our project plan?"



## **Risk Item Checklist**

- Used as one way to identify risks
- Focuses on known and predictable risks in specific subcategories (see next slide)
- Can be organized in several ways
  - A list of characteristics relevant to each risk subcategory
  - Questionnaire that leads to an estimate on the impact of each risk
  - A list containing a set of risk component and drivers and their probability of occurrence

# Known and Predictable Risk Categories

- Product size
- Business impact
- Customer characteristics
- Process definition
- Development environment
- Technology to be built
- Staff size and experience

# Risk Projection

Risk projection (or estimation) attempts to rate each risk in two ways

- The probability that the risk is real
  - The consequence of the problems associated with the risk, should it occur
- 
- The project planner, managers, and technical staff perform four risk projection steps (see next slide)
  - The intent of these steps is to consider risks in a manner that leads to Prioritization
  - Be prioritizing risks, the software team can allocate limited resources where they will have the most impact

# **Risk Projection/Estimation Steps**

- 1) Establish a scale that reflects the perceived likelihood of a risk  
e.g., 1-low, 10- high
- 2) Delineate the consequences of the risk
- 3) Estimate the impact of the risk on the project and product
- 4) Note the overall accuracy of the risk projection so that there will be no misunderstandings

## Contents of a Risk Table

- A risk table provides a project manager with a simple technique for risk projection
- It consists of five columns
  - **Risk Summary** – short description of the risk
  - **Risk Category** – one of seven risk categories (slide 12)
  - **Probability** – estimation of risk occurrence based on group input
  - **Impact** : (1) catastrophic  
(2) critical  
(3) marginal  
(4) negligible
  - **RMMM** – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

## Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value (See next slide)
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

## Assessing Risk Impact

- Three factors affect the consequences that are likely if a risk does occur
  - Its nature – This indicates the problems that are likely if the risk occurs
  - Its scope – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)
  - Its timing – This considers when and for how long the impact will be felt
- The overall risk exposure formula is  $RE = P \times C$ 
  - $P$  = the probability of occurrence for a risk
  - $C$  = the cost to the project should the risk actually occur
- Example
  - $P$  = 80% probability that 18 of 60 software components will have to be developed
  - $C$  = Total cost of developing 18 components is \$25,000
  - $RE = .80 \times \$25,000 = \$20,000$



# Seven Principles of Risk Management

- **Maintain a global perspective**
  - View software risks within the context of a system and the business problem that is intended to solve
- **Take a forward-looking view**
  - Think about risks that may arise in the future; establish contingency plans
- **Encourage open communication**
  - Encourage all stakeholders and users to point out risks at any time
- **Integrate risk management**
  - Integrate the consideration of risk into the software process

## **Seven Principles of Risk Management (Contd...)**

- **Emphasize a continuous process of risk management**
  - Modify identified risks as more becomes known and add new risks as better insight is achieved
- **Develop a shared product vision**
  - A shared vision by all stakeholders facilitates better risk identification and Assessment
- **Encourage teamwork when managing risk**
  - Pool the skills and experience of all stakeholders when conducting risk management activities

# Configuration Management

## INTRODUCTION:

- Configuration Management helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- *In software engineering, Software configuration management* is the task of *tracking and controlling* changes in the software
  - SCM practices include revision control and the *establishment of baselines*.
  - If something goes wrong, SCM can determine what was changed and who changed it.
  - If a configuration is working well, SCM can determine how to replicate it across many hosts.
- The acronym "**SCM**" is also referred as **source configuration management process** and **software change and configuration management**.
- **Configuration** is generally understood to cover changes typically made by a system administrator.

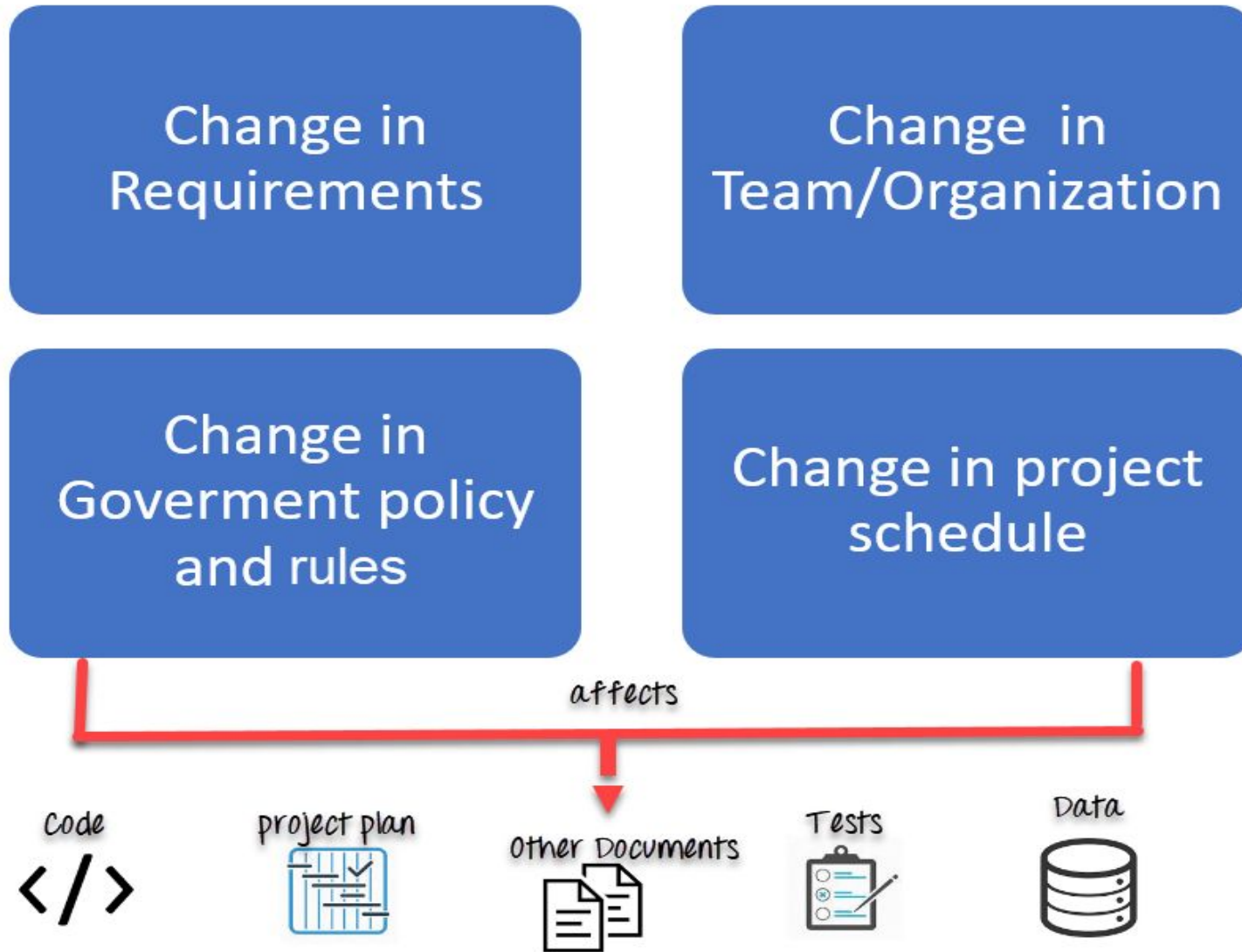
# Configuration Management

## NEED FOR SOFTWARE CONFIGURATION MANAGEMENT:

- There are multiple people working on software which is continually updating.
- It may be a case where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently.
- Changes in user requirement, policy, budget, and schedule need to be accommodated.
- Software should be able to run on various machines and Operating Systems helps to develop coordination among stakeholders.
- SCM process is also beneficial to control the costs involved in making changes to a system.

**NOTE: Stake holders** - A person or group of people who own a share in a business.

# Configuration Management



# Configuration Management

## IMPORTANCE OF SOFTWARE CONFIGURATION MANAGEMENT:

- **Configuration management (CM)** focuses on establishing and maintaining consistency of a product's performance, and its functional and physical attributes with its requirements, design, and operational information throughout its life.
- CM streamlines the delivery of software and applications by automating the build out of systems quickly and efficiently.
- It can be used by management and engineers to check which components have been changed and why, ensuring an audit trail of changes done to the system.
- This helps with quickly identifying bad configuration changes and allows for rollbacks to well-known working ones to ensure rapid restoration of service(s).
- This also helps developers with debugging to check if configuration changes impacts the product's functionality.

# Configuration Management

## The Four Basic Requirements for an SCM

**Identification:** Each software part is labeled so that it can be identified. Furthermore, there will be different versions of the software parts as they evolve over time

**Control:** Control in the context of configuration management, "control" means that proposed changes to a CI are reviewed and, if approved, incorporated into the software configuration.

**Auditing:** Auditing an SCM system means that approved requested changes have indeed been implemented.

**Status accounting:** Reports and documentation produced by the status accounting function are the auditable entries.



# Configuration Management

## The Four Basic Requirements for an SCM

**Identification:** Each software part is labeled so that it can be identified. Furthermore, there will be different versions of the software parts as they evolve over time

**Control:** Control in the context of configuration management, "control" means that proposed changes to a CI are reviewed and, if approved, incorporated into the software configuration.

**Auditing:** Auditing an SCM system means that approved requested changes have indeed been implemented.

**Status accounting:** Reports and documentation produced by the status accounting function are the auditable entries.

# Configuration Management

## THE SOFTWARE CONFIGURATION MANAGEMENT PLAN

Defines the *types of documents* to be managed and a document naming scheme.

Defines *who takes responsibility* for the CM procedures and creation of baselines.

Defines *policies for* change control and version management.

Describes the *tools* which should be used to assist the CM process and any limitations on their use.

Defines the *configuration management database* used to record configuration information.

# Configuration Management

## **Outline of a Software Configuration Management Plan (SCMP, IEEE 828-1990)**

**Introduction** : Describes purpose, scope of application, key terms and references

**Management (WHO?)** : Identifies the responsibilities and authorities for accomplishing the planned configuration management activities

**Activities (WHAT?)** : Identifies the activities to be performed in applying to the project.

**Schedule (WHEN?)** : Establishes the sequence and coordination of the SCM activities with project mile stones.

**Resources (HOW?)** : Identifies tools and techniques required for the implementation of the SCMP

**Maintenance** : Identifies activities and responsibilities on how the SCMP will be kept current during the life-cycle of the project.

# Configuration Management

## **SYSTEM MAINTENANCE:**

### **Maintenance is Inevitable:**

- System requirements are likely to change while the system is being developed because their environment is changing
- Systems are tightly coupled to their environment
- When a system is installed it changes the environment and that can change the system requirements
- The delivered system may not meet its requirements
- Systems must be maintained to remain useful in their environment

# Configuration Management

## **Types of Maintenance:**

### **Corrective Maintenance (21%)**

making changes to repair defects

### **Adaptive Maintenance (25%)**

making changes to adapt software to external environment changes (hardware, business rules, OS, etc.)

### **Perfective Maintenance (50%)**

extending system beyond its original functional requirements

### **Preventative Maintenance (4%)**

modifying work products so that they are more easily corrected, adapted, or enhanced

# Configuration Management

## Configuration Items

### Identifying Configuration items

- Large projects typically produce thousands of entities (files, documents, data ...) which must be uniquely identified.
- Any entity managed in the software engineering process can potentially be brought under configuration management control
- But not every entity needs to be under configuration management control all the time.

# Configuration Management

## Configuration Items:

- Requirements Analysis Document (RAD)
- System Design Document (SDD)
- Object Design Document (ODD)
- Unit tests
- Source code
- Input data and data bases
- Test data
- Project Data
- Manuals



# Configuration Management

## Auditing

- Extreme attention to detail
- Ability to see congruence
- Ability to perceive what is missing
- Extensive experience with technical aspects of system engineering or software engineering

## Status Accounting

- Ability to take notes and record data
- Ability to organize data
- Some technical familiarity
- System engineering orientation
- Programming

# Configuration Management

## Benefits :

- The major benefits of a configuration management solution for application infrastructure relate to reducing complexity and promoting compliance for IT.
- By transforming today's manual, error-prone tasks into a sequence of automated processes, your IT team will see benefits in:
  - Improving the productivity of your IT Infrastructure Team, by upwards of 50% and enabling them to spend more time on new initiatives rather than fire-fighting
  - Accelerating time-to-value for applications by 25%
  - Improving the quality and uptime of your applications by reducing the key elements that cause most of today's outages
  -

# REFERENCES:

- Roger S. Pressman, Software Engineering – A Practitioner Approach, 8th ed., McGraw Hill, 2019
- Ian Sommerville, Software Engineering, 8th ed., Pearson Education, 2010
- Walker Royce, Software Project Management, Pearson Education, 1999
- Jim Smith Agile Project Management: Creating Innovative Products, Pearson