



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

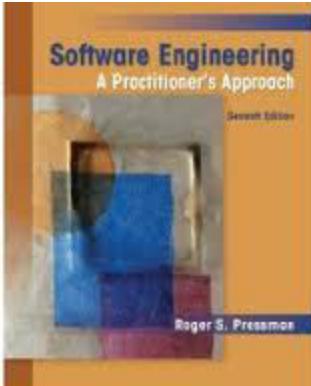
# SS ZG622:

## Software Project Management

### (Introduction, Concepts & Terminology)

Prof K G Krishna, BITS-Pilani Off-Campus Centre, Hyderabad

# Text Books



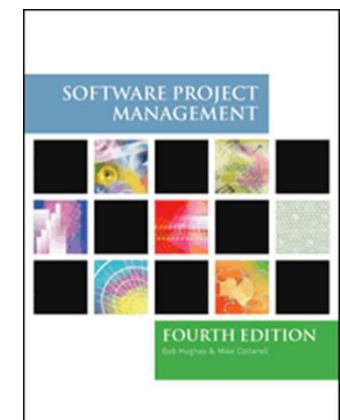
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Software Project Management:**

## **Introduction, Concepts & Terminology**

# TOPICS

---

- Project – a technical definition
- What makes Software Projects different?
- Project Life-cycle
- Project vs. Program
- Role of Project Manager
- Project Stakeholders
- Measuring success of a project

# What is a Project?

---

- Project Defined
  - A complex, non-routine, one-time effort limited by time, budget, resources, and performance specifications designed to meet customer needs.

# Project Definition / Attributes

---

- A project is unique, has specific time frame (not a continuing or on-going activity like automobile production, ...), requires resources, interdependent activities, and has a degree of uncertainty.

# Routine Work vs. Projects

## Routine, Repetitive Work

Taking class notes

Daily entering sales receipts into the accounting ledger

Responding to a customer call

Writing a program for tax computation

Manufacture of an Apple iPhone

## Projects

Writing a term paper

Setting up a sales desk for a professional accounting

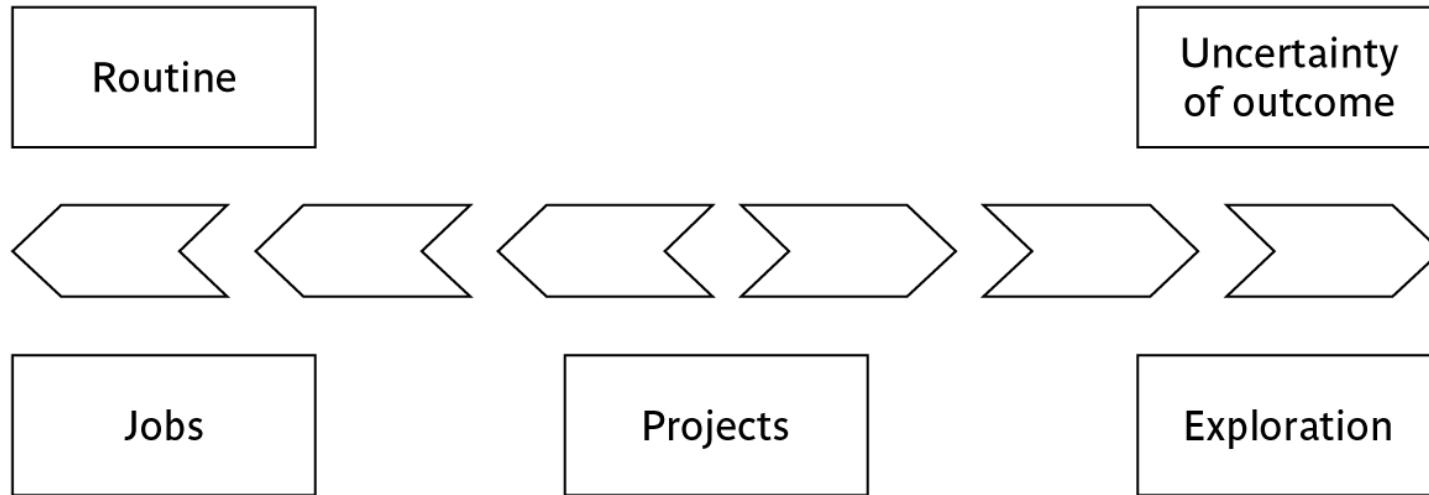
Developing a customer support system

Developing a tax management system

Designing an iPhone that is approximately 2 X 4 inches, interfaces with iWatch, and stores 10,000 songs

TABLE 1.1

# On Uncertainty scale: Jobs (routine work) vs. Projects



‘Jobs’ – repetition of very well-defined and well understood tasks with very little uncertainty

‘Exploration’ – e.g. finding a cure for cancer: the outcome is very uncertain

Projects – in the middle!

# Programs vs. Projects

---

- Program Defined
  - A series of coordinated, related, multiple projects that continue over an extended time and are intended to achieve a goal.
  - A higher level group of projects targeted at a common goal.
  - Example:
    - *Project*: completion of a required course in project management.
    - *Program*: completion of all courses required for a business major.

# Summary:

## Major Characteristics of a Project

---

- Non-routine
- Planned
- Has an established objective (Project Goal)
- Has a defined life span with a beginning and an end.
- Requires across-the-organizational participation.
- Involves doing something never been done before.
- Has specific time, cost, and performance requirements
- Made up of several different phases
- Large / complex
- Has a Structured Methodology for Planning & Execution
- ...



# Thank You

Next...>

*Activities/Phases in  
Software Project Management*

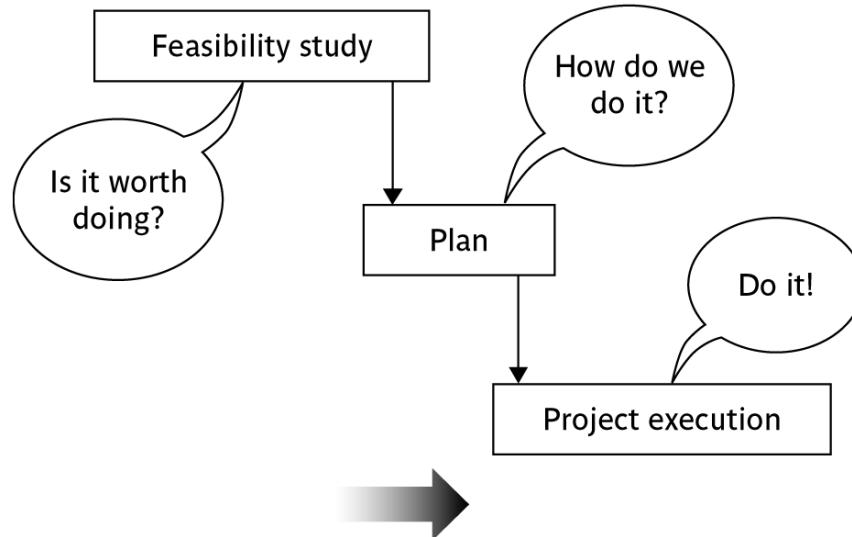
# What is management?

---

This involves the following activities:

- **Planning** – deciding what is to be done
- **Organizing** – making arrangements
- **Staffing** – selecting the right people for the job
- **Directing** – giving instructions
- **Measurements** – of activities, effort, schedules, work-products,...
- **Monitoring** – checking on progress
- **Controlling** – taking action to remedy hold-ups
- **Innovating** – coming up with solutions when problems emerge
- **Representing** – liaising with clients, users, developers and other stakeholders
- ...

# Activities covered by project management



## Feasibility study

Is project technically feasible and worthwhile from a business point of view?

## Planning

Only done if project is feasible

## Execution

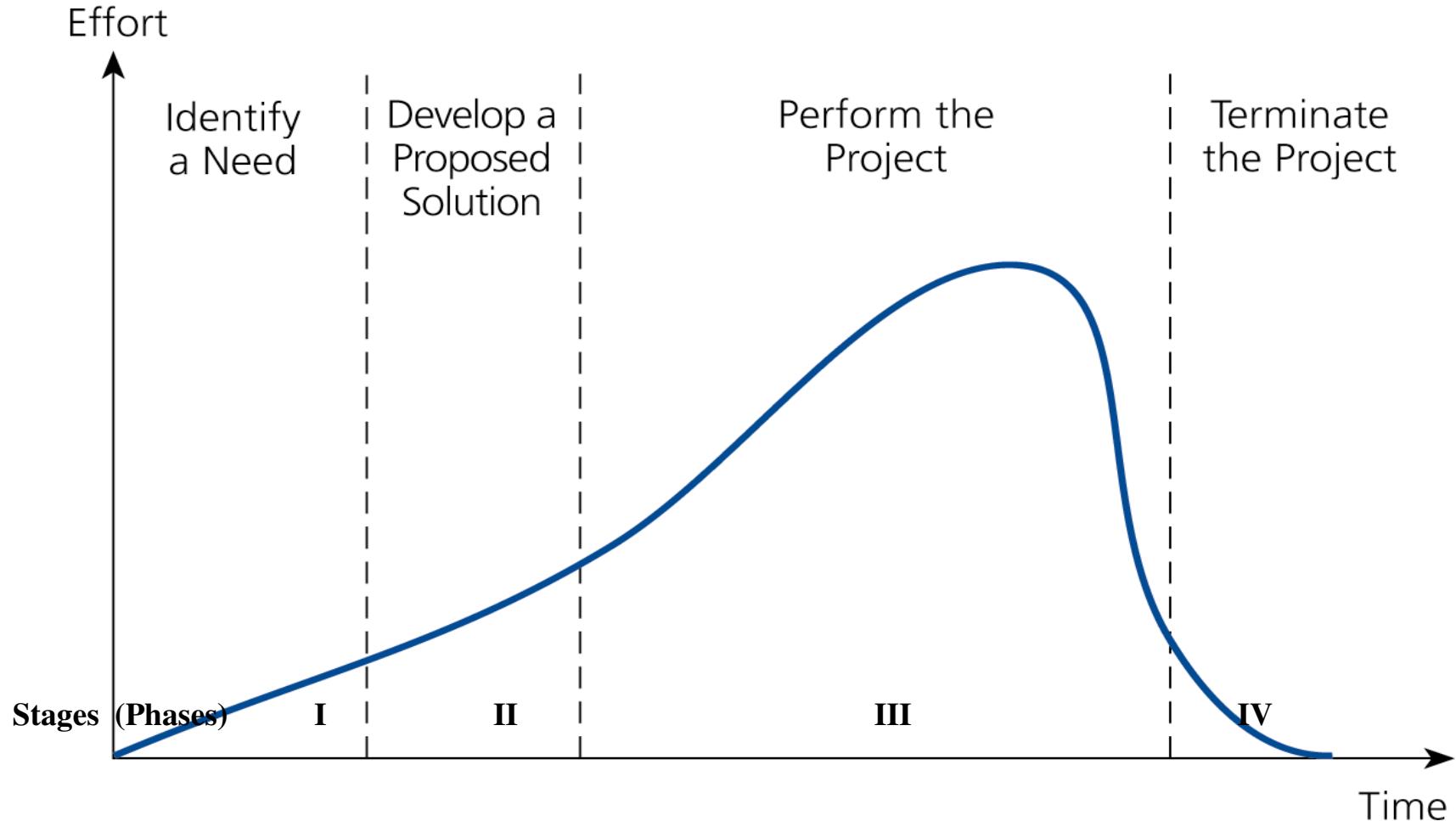
Implement plan, but plan may be changed as we go along

# In short, Project Management involves:

---

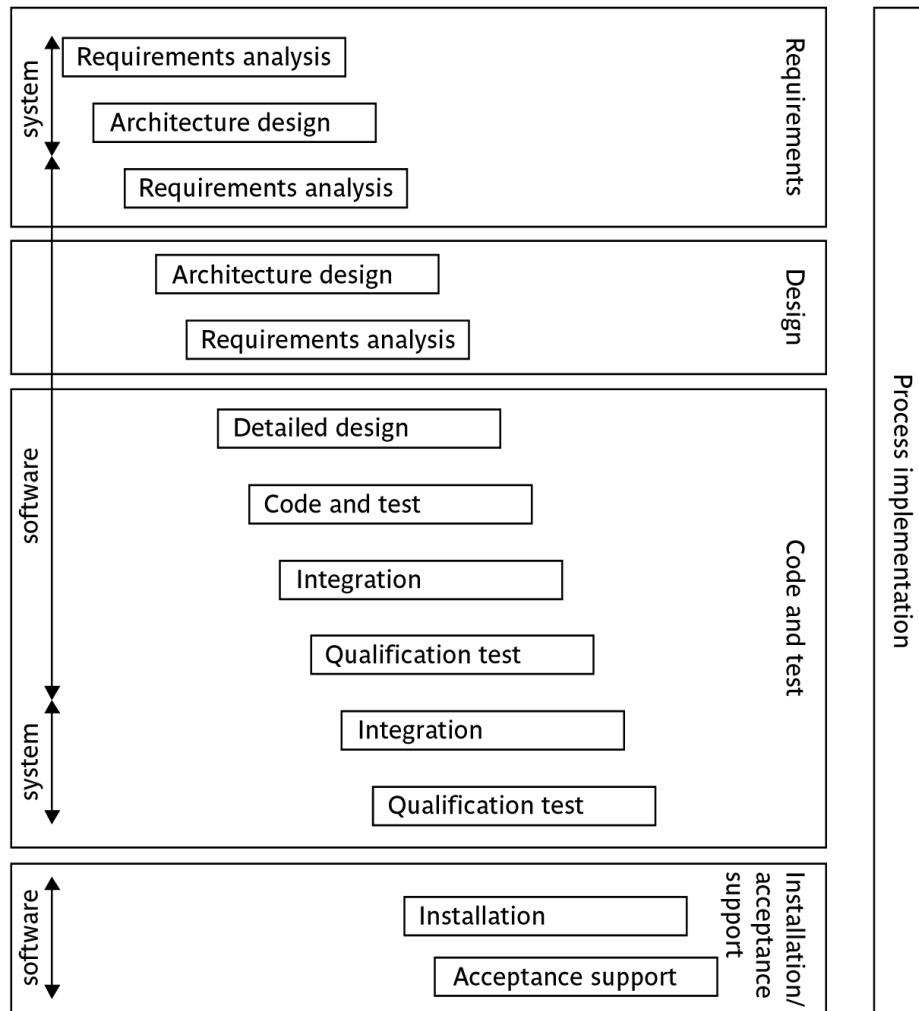
- Typical Project Life Cycle (Generic, 4 phases, )
  - Identifying need and inviting proposals
  - Receiving and evaluating proposals and project selection
  - Executing the project
  - Terminating the project

# Project Life-cycle (Phases)



# The software development life-cycle

(Ref: ISO 12207: Framework for Software Life-cycle Processes)



# Thank You

*Next...> Challenges in Software Project Management*

# The Challenge of Project Management

---

- The Project Manager
  - Manages temporary, non-repetitive activities and frequently acts independently of the formal organization.
    - Marshals resources for the project.
    - Has direct interface with customer and other stakeholders
    - Provides direction, coordination, and integration to the project team.
    - Is responsible for end-to-end performance and success of the project.
  - Manages teams: induct right people at the right time; motivate teams for performance; resolves issues and conflicts
  - Address issues by escalating at the right time and takes appropriate decisions
  - ...

# The Importance of Project Management

---

- Factors leading to the increased use of project management:
  - Compression of the product life cycle
  - Global competition
  - Knowledge explosion
  - Corporate downsizing (cost competitiveness)
  - Increased customer focus
  - Expansion of the economy and resultant opportunities
  - ...

# Software Project Management!

---

- What is software project management? How is it really different from management of other projects?
- How do you know when a project has been successful?

# Why is software project management important?

---

- Large amounts of money are spent on ICT
- Project often fail – Standish Group claim only a third of ICT projects are successful. 82% were late and 43% exceeded their budget.
- Poor project management a major factor in these failures

# Are *software* projects really different from other projects?

---

*Not really ...but*

- Invisibility
- Complexity
- Conformity
- Flexibility
- Changing Requirements
- Skill Requirements
- ...

make software more problematic to build than engineered artefacts in other fields

# Thank You

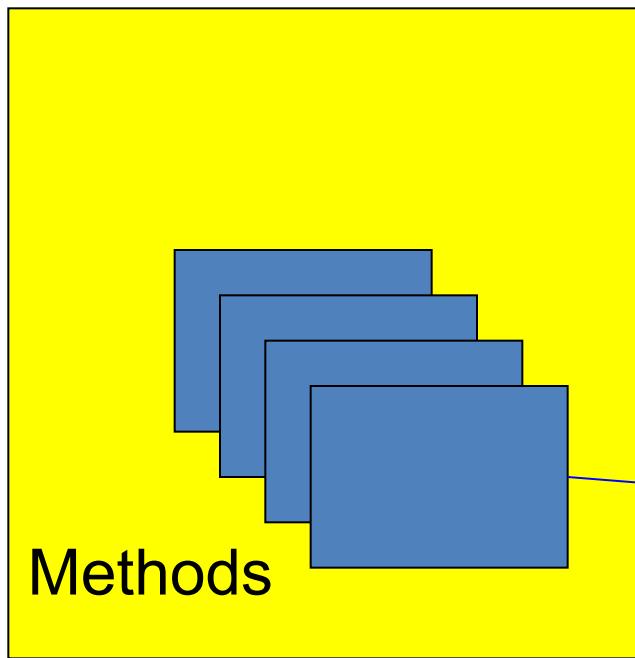
Next...>

*Project Stakeholders &  
Success Criteria*

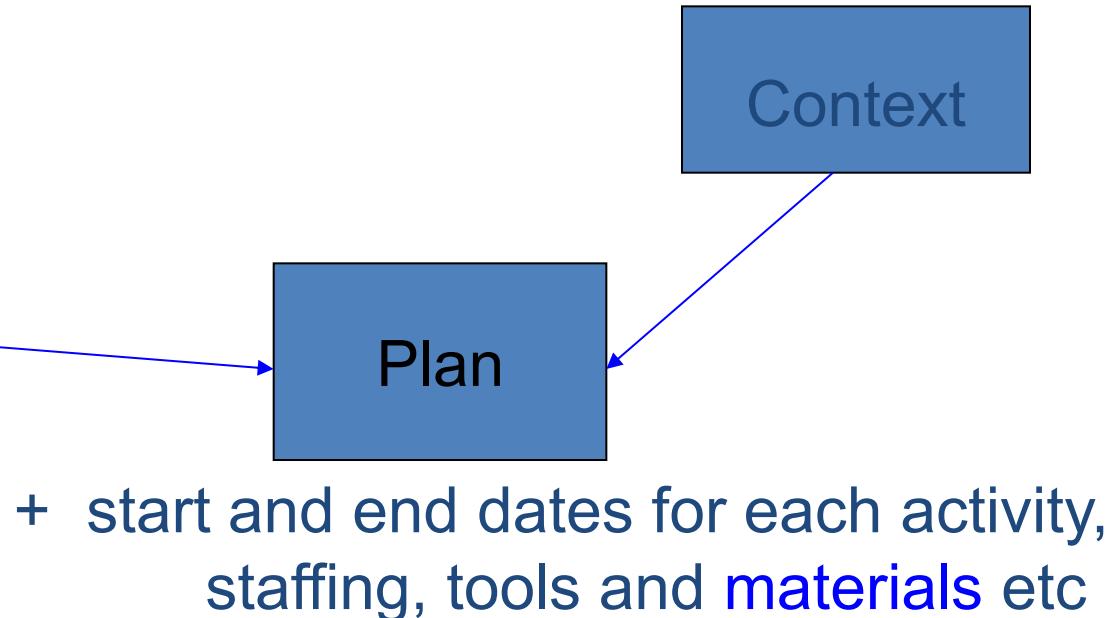
# Professional Software Development employs formal Methods:



Methodology = a set of methods



A way of working



# Project Stakeholders

---

These are people who have a stake or interest in the project  
In general, they could be *users/clients* or *developers/implementers*

They could be:

- Within the project team
- Outside the project team, but within the same organization
- Outside both the project team and the organization (society,...)

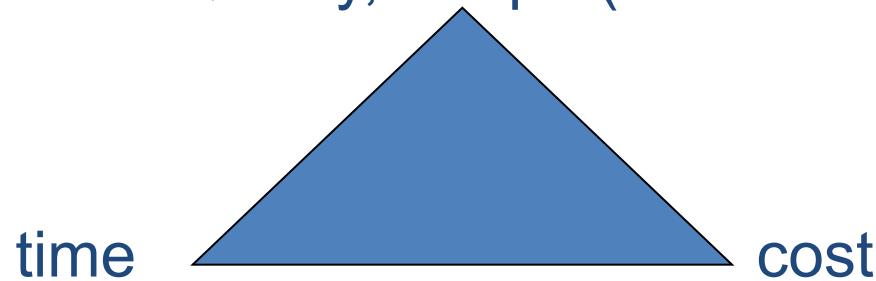
Different stakeholders may have different objectives – need to define common project objectives

# Project success/failure

---

- Degree to which objectives are met

Quality, Scope (of deliverables)



In general if, for example, project is running out of time, this can be recovered for by reducing scope or increasing costs. Similarly costs and scope can be protected by adjusting other corners of the 'project triangle'.

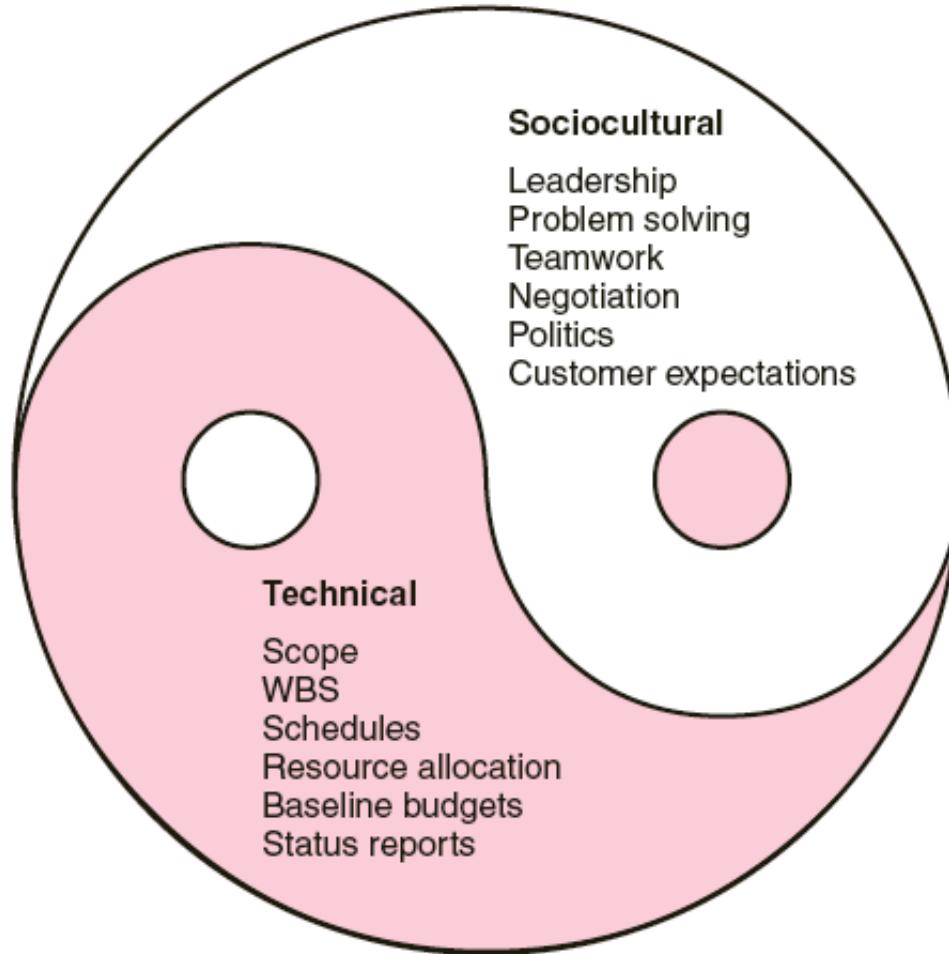
# Other success criteria

---

These can relate to longer term, less directly tangible assets

- Improved skill and knowledge
- Creation of assets that can be used on future projects e.g. software libraries
- Improved customer relationships that lead to repeat business
- ...

# Social & Technical aspects of Project Management



# Summary

---

- Projects are non-routine – thus there is an element of uncertainty
- In addition, managing software projects involve dealing with complexity, uncertain or changing requirements
- The role of a Project Manager is well-defined and he/she is responsible for the success of the project managing teams, stakeholders and resources
- Professional Software Development involves adoption of formal Software Engineering Models

---

# Thank You

## Any Questions?



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

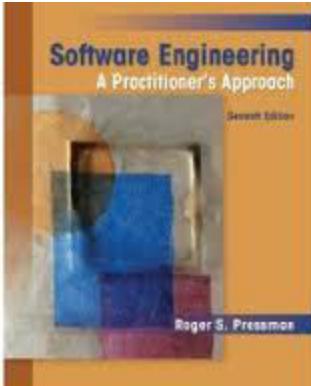
# SS ZG622:

## Software Project Management

### (Software Development Models)

K G Krishna, BITS-Pilani, Hyderabad Campus

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Software Process, Software Engineering Models**

## **– An Overview**

# Software Engineering—The Backbone of IT Project Management



# The software process

---

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

# Software process descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a **data model**, **designing a user interface**, etc. and the **sequencing** of these activities.
- Process descriptions may also include:
  - Products, which are the outcomes or intermediate deliverables of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

# Plan-driven and agile processes

---

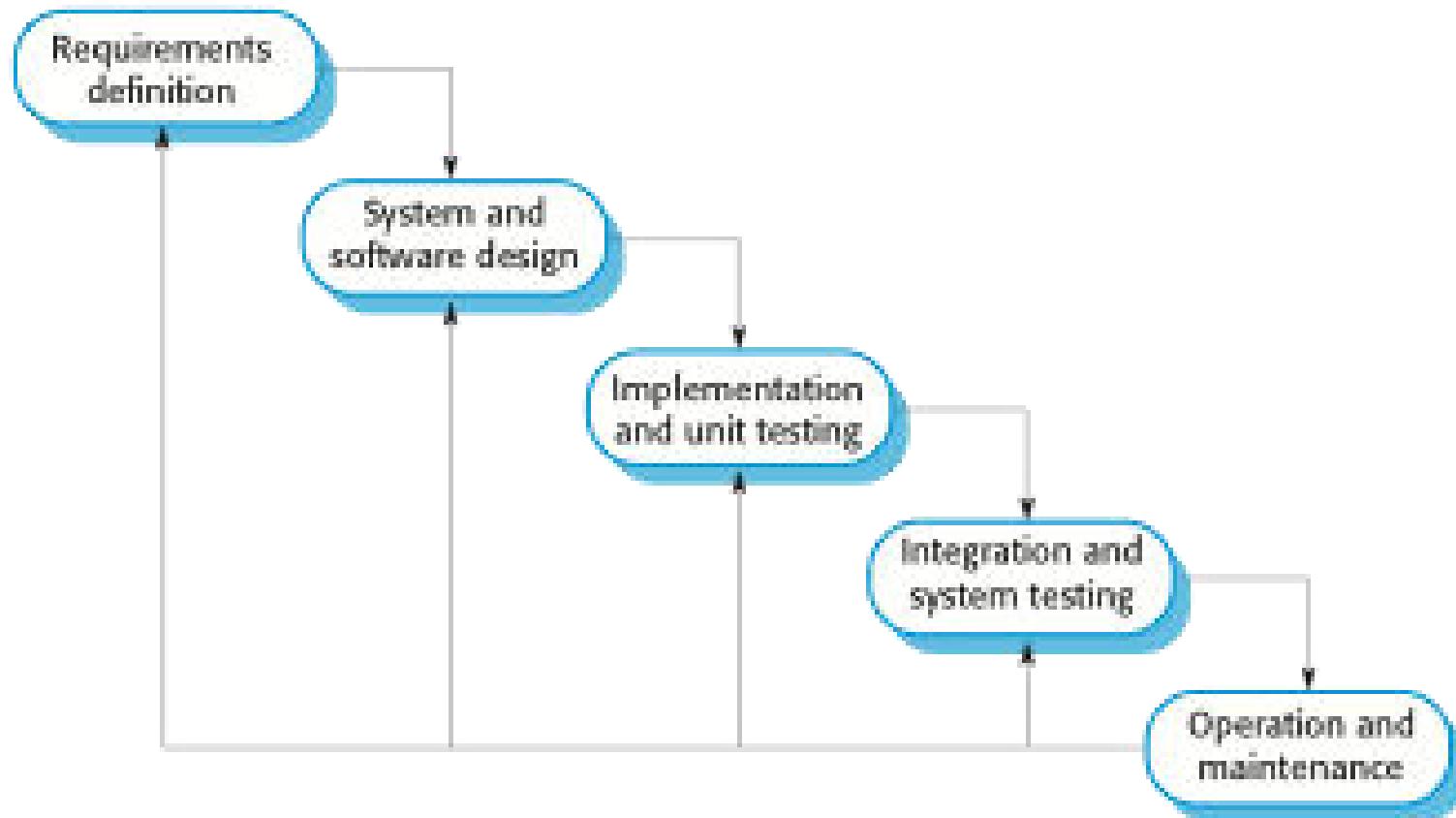
- Plan-driven processes are processes where all of the process activities are **planned in advance** and progress is **measured against this plan**.
- In agile processes, planning is **incremental** and it is easier to change the process to reflect changing customer requirements.
- In practice, most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

# Software process models

---

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Reuse-oriented software engineering
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

# The Waterfall model



# Waterfall model phases

---

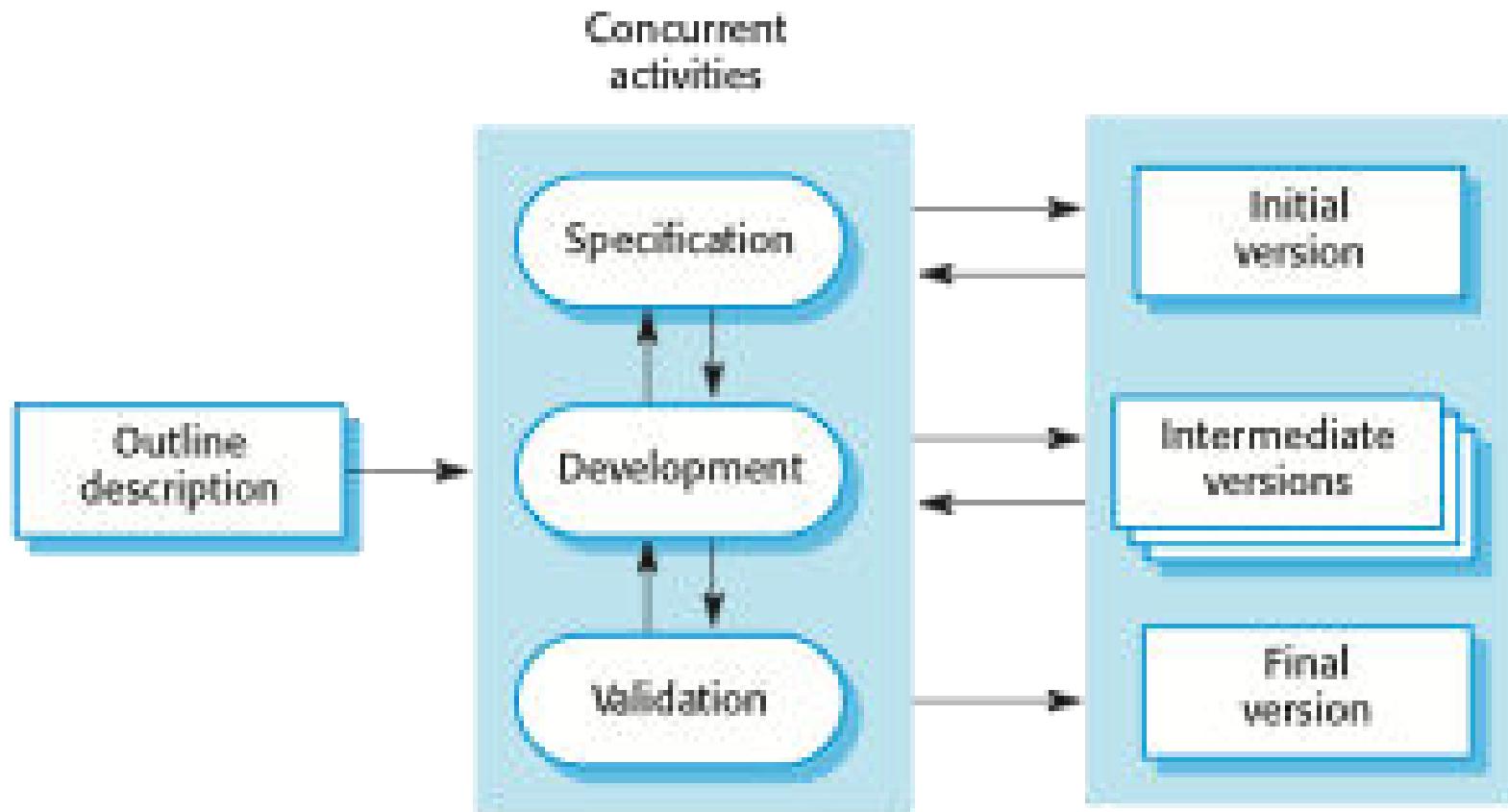
- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

# Waterfall model problems

---

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

# Incremental development



# Incremental development: Benefits

---

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

# Incremental development: Problems

---

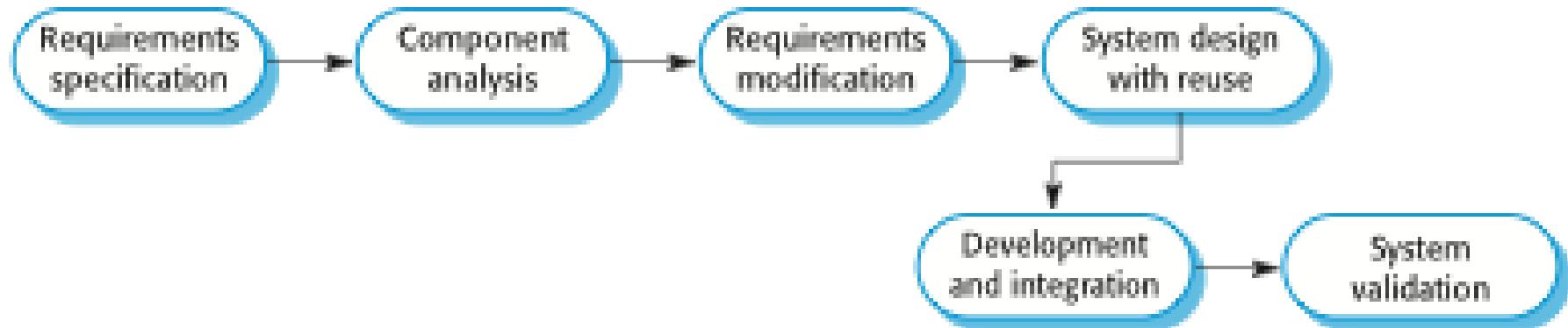
- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money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

# Reuse-oriented software engineering

---

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- Reuse is now the standard approach for building many types of business and industrial software systems today (Libraries, Frameworks, SaaS,...)

# Reuse-oriented software engineering



# Types of software component

---

- Web services that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Stand-alone software systems (COTS) that are configured for use in a particular environment.

# Process activities

---

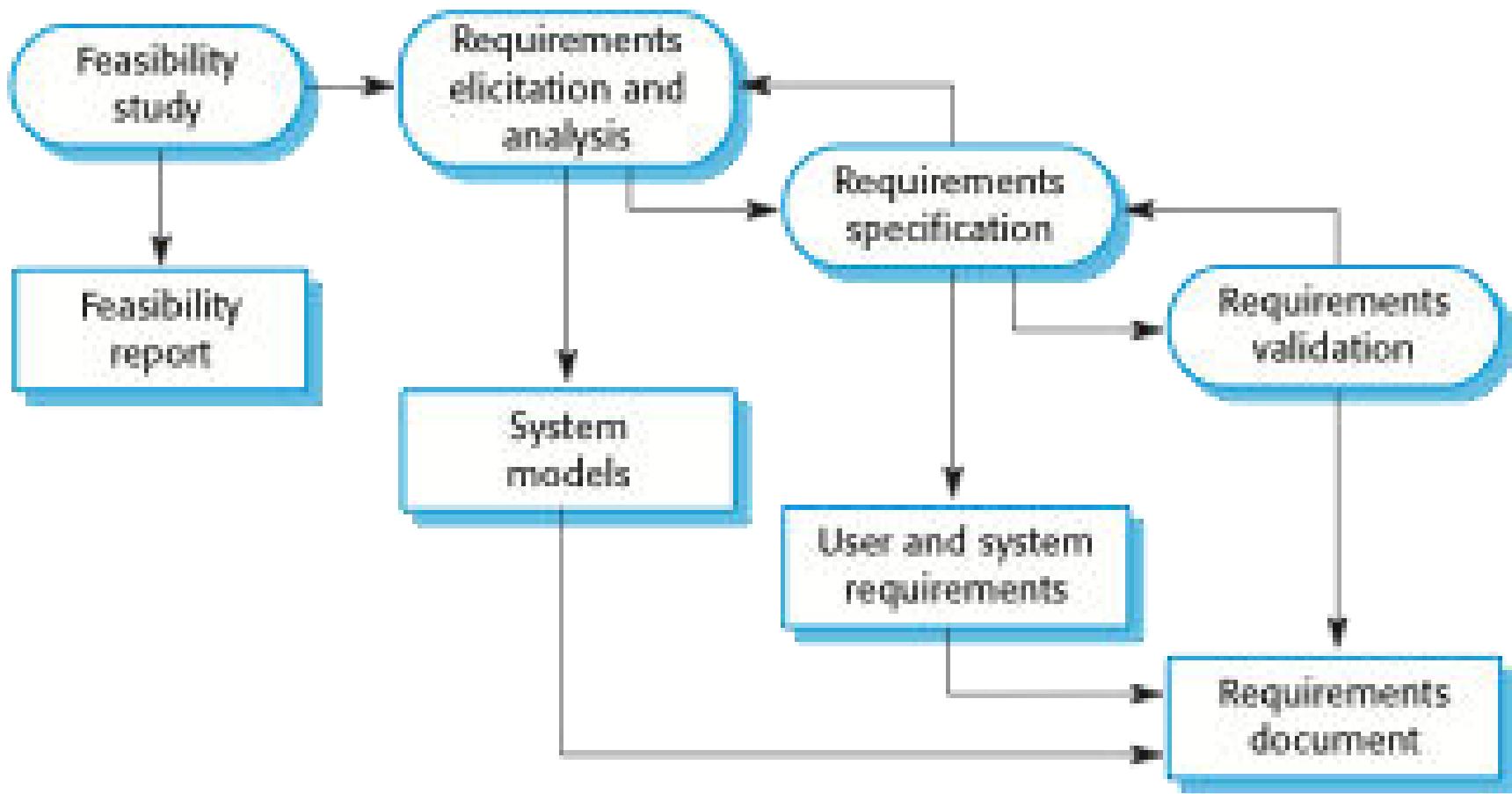
- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of **specifying, designing, implementing and testing** a software system.
- The four basic process activities of **specification, development, validation and evolution** are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.

# Software specification

---

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements

# The requirements engineering process

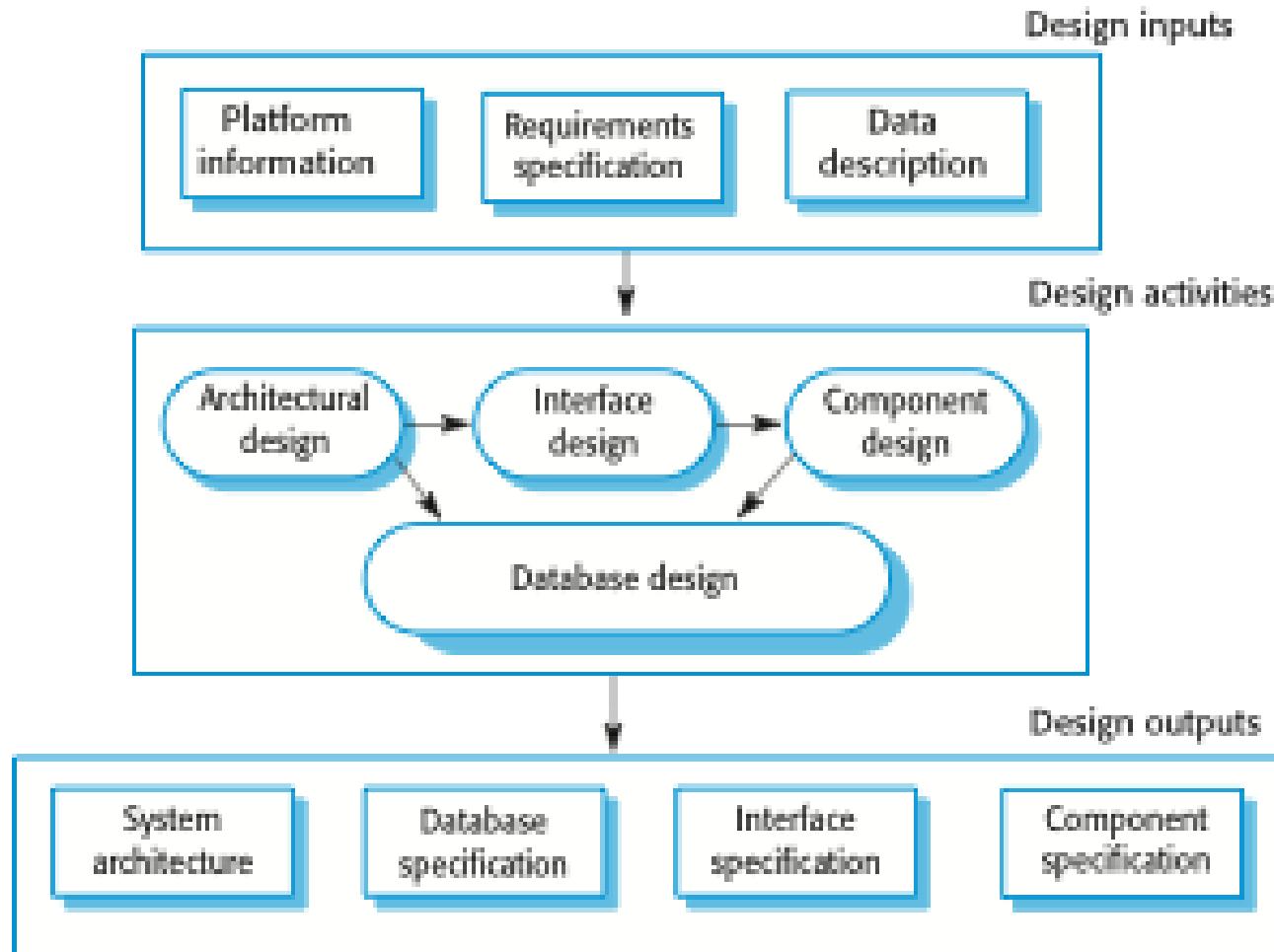


# Software design and implementation

---

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are closely related and may be inter-leaved.

# A general model of the design process



# Design activities

---

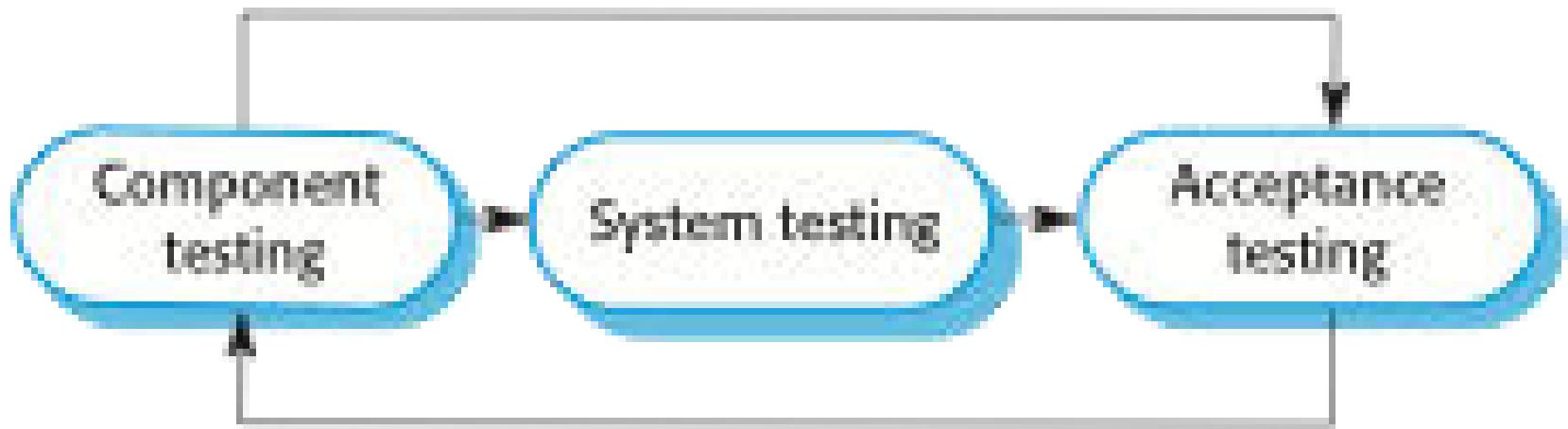
- *Architectural design*, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.
- *Interface design*, where you define the interfaces between system components.
- *Component design*, where you take each system component and design how it will operate.
- *Database design*, where you design the system data structures and how these are to be represented in a database.

# Software validation

---

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

# Stages of testing

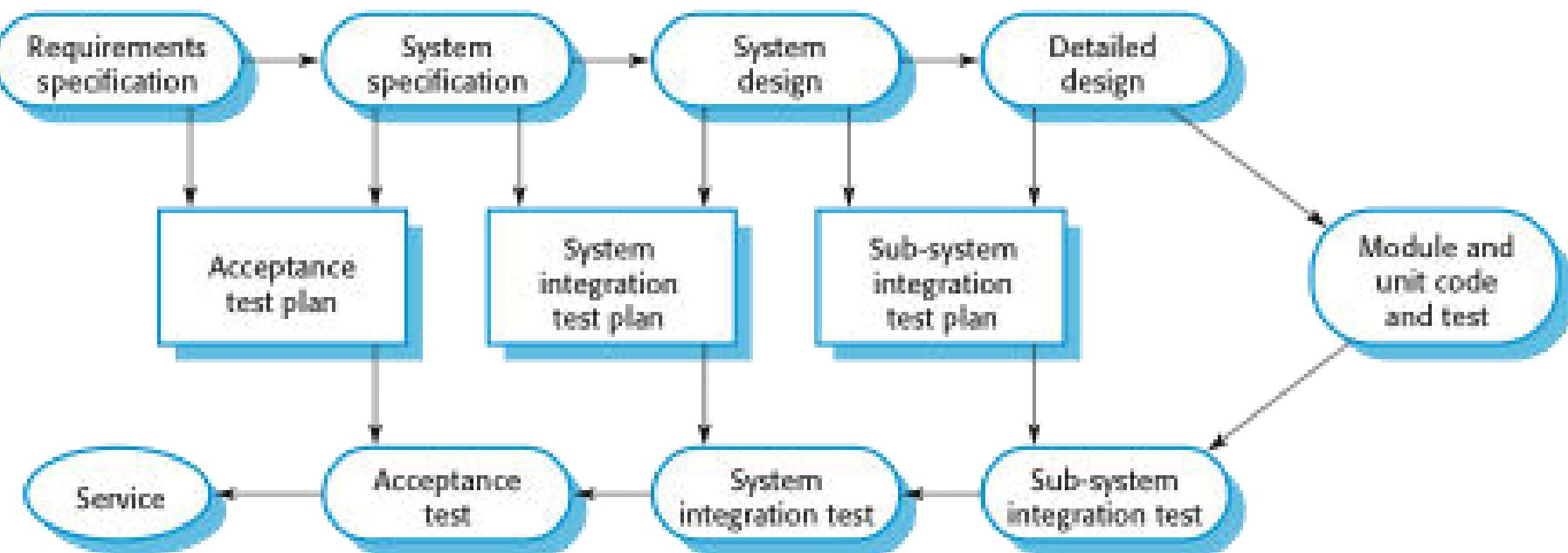


# Testing stages

---

- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs.

# Testing phases in a plan-driven software process

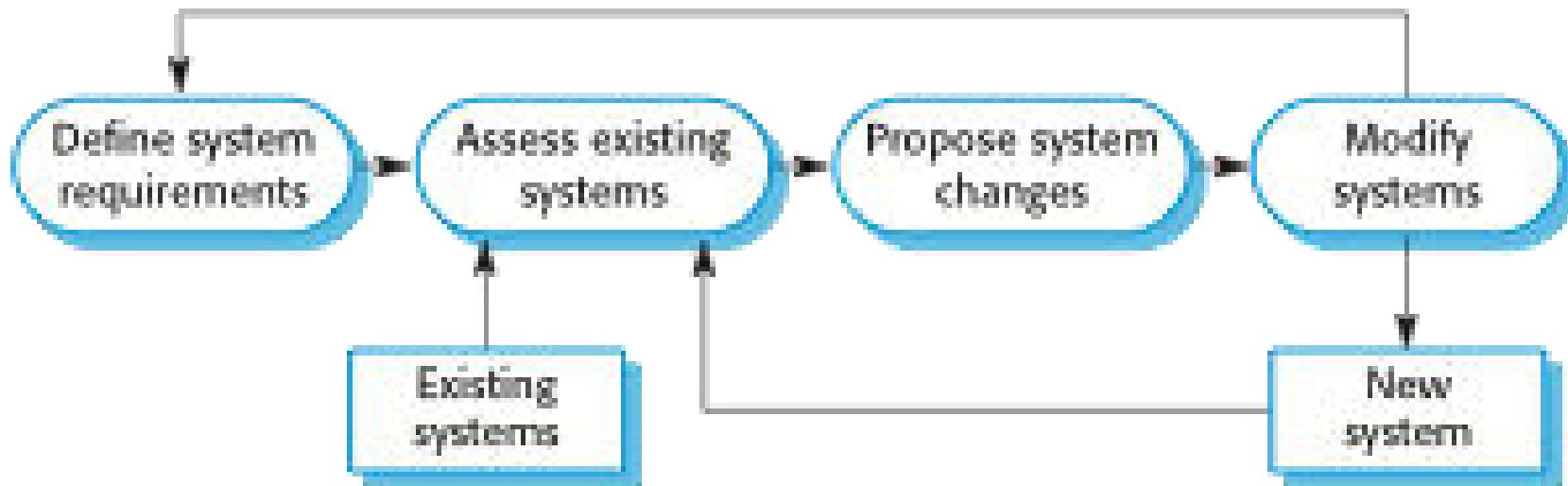


# Software evolution

---

- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

# System evolution



# Summary: Software Process

---

- Software processes are the activities involved in producing a software system. Software process models are **abstract representations** of these processes.
- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development, Agile Models.

# Summary: Software Process

---

- **Requirements engineering** is the process of developing a software specification.
- **Design and implementation** processes are concerned with transforming a requirements specification into an executable software system.
- **Software validation** is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- **Software evolution** takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

---

# Thank You



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

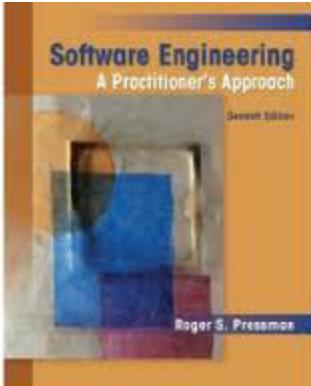
# SS ZG622:

# Software Project Management

(Selection of Development Process Models)

K G Krishna, Off-Campus Centre, BITS-Pilani, Hyderabad

# Text Books



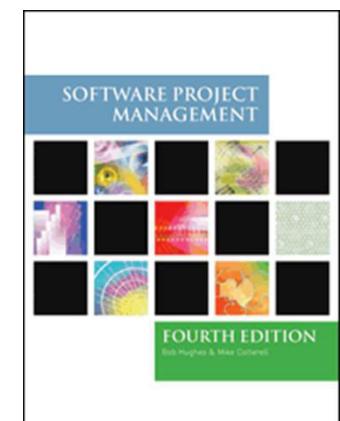
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# Selecting the Right Development Process Model for the Project

# Project Evaluation: Summary

---

- A project may fail not through poor management but because it should never have been started
- A project may make a profit, but it may be possible to do something else that makes even more profit
- A real problem is that it is often not possible to express benefits in accurate financial terms
- Projects with the highest potential returns are often the most risky

# Selection of Project Approach

## (Development Model)

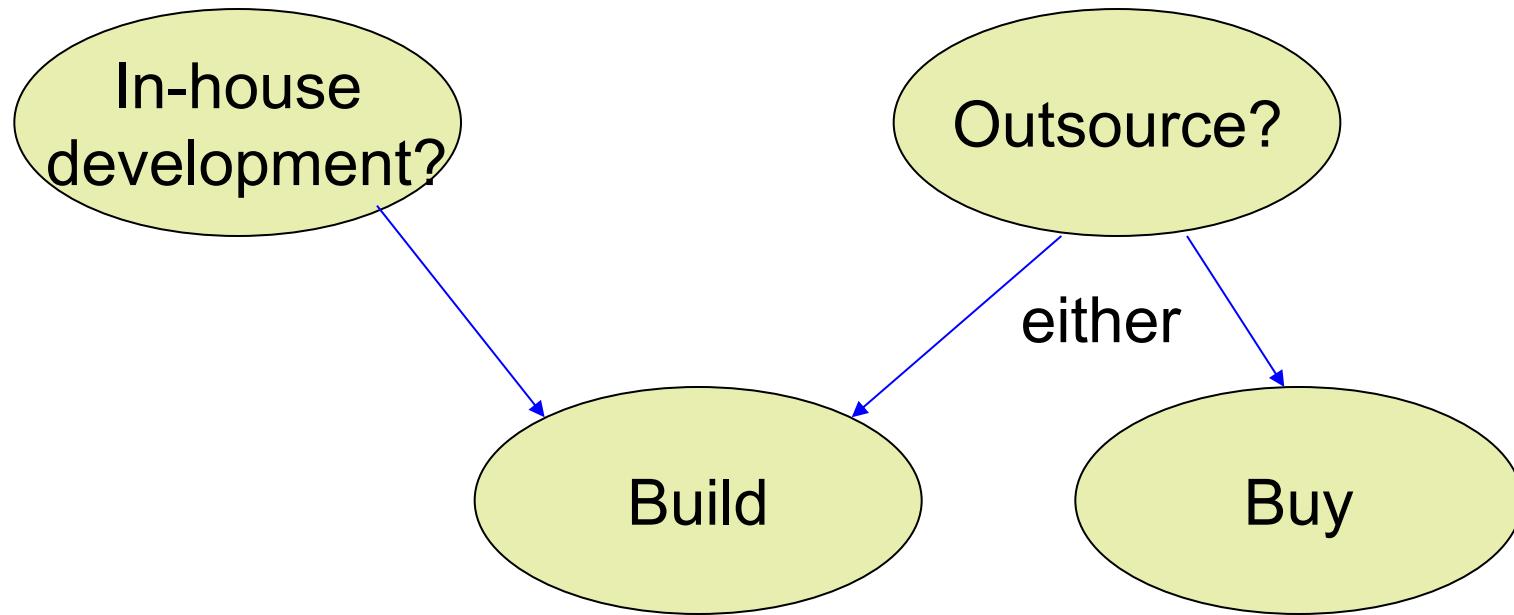
- Building versus buying software
- Taking account of the characteristics of the project
- Process models
  - Waterfall
  - Prototyping and iterative approaches
  - Incremental delivery
- Agile methods

# Selection the Right Model for Development

---

- Choosing the right approach to a particular project:
  - *technical planning, project analysis, methods engineering / custom models*
- In-house: often the methods to be used dictated by organizational standards
- Vendors: need for tailoring as different customers have different needs

# Build or Buy?



# Some advantages of off-the-shelf (OTS) software

---



- Cheaper as supplier can spread development costs over a large number of customers
- Software already exists
  - Can be used by potential customer
  - No delay while software being developed
- Where there have been existing users, bugs are likely to have been found and eradicated

# Some possible disadvantages of OTS software products

---



- Customer will have same application as everyone else: no competitive advantage, *but* competitive advantage may come from the *way* application is used
- Customer may need to change the way they work in order to fit in with OTS application
- Customer does not own the code and cannot change it
- Danger of over-reliance on a single supplier

# General Guidelines:



- Look at risks and uncertainties e.g.
  - are requirements well understood?
  - are technologies to be used well understood?
- Look at the type of application being built e.g.
  - information system? embedded system?
  - criticality? differences between target and development environments?
- Clients' own requirements
  - need to use a particular method

# Structured Development vs. Speed of Delivery



## Structured approach:

- Also called ‘heavyweight’ approaches
- Step-by-step methods where each step and intermediate product is carefully defined
- Emphasis on getting quality right first time
- Example: use of UML
- Future vision: Model-Driven Architecture (MDA) --UML supplemented with Object Constraint Language, press the button and application code generated from the UML/OCL model (?)

# Structure vs. Speed

---

- Agile methods (for speed)
  - Emphasis on speed of delivery rather than documentation
- RAD (Rapid application development)
  - emphasizes use of quickly developed prototypes

# Basic Choice of Process Models

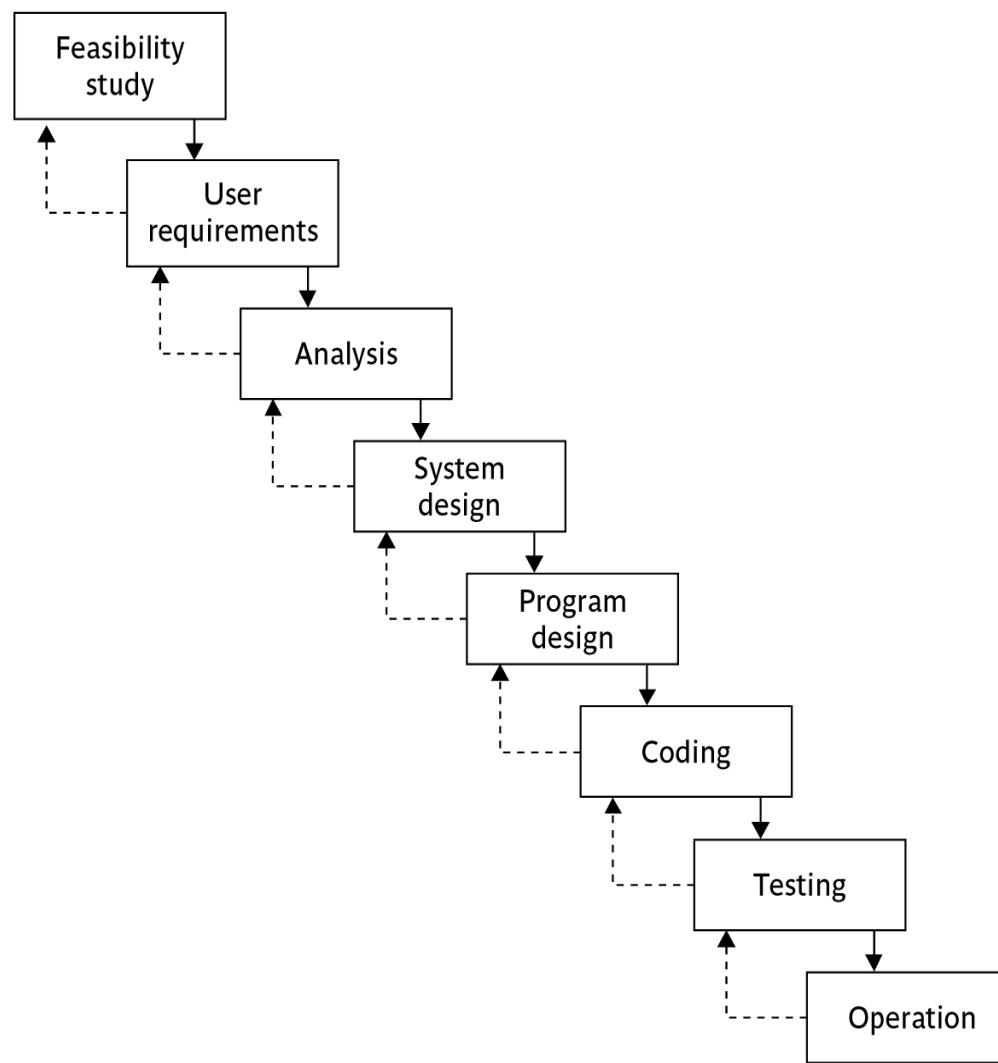
---

- Waterfall ('heavy-weight')
- Incremental delivery
- Evolutionary development

Or

use of 'agile methods' e.g. extreme programming  
/ Scrum Frameworks

# Waterfall Model



# Waterfall

---

- the 'classical' model
- imposes structure on the project
- every stage needs to be checked and signed off
- BUT
  - limited scope for iteration
- V model approach is an extension of waterfall where different testing phases are identified which check the quality of different development phases

# Evolutionary delivery: prototyping

- ‘An iterative process of creating quickly and inexpensively live and working models to test out requirements and assumptions’
  - ‘throw away’ prototypes
  - evolutionary prototypes

what is being prototyped?

- human-computer interface
- functionality

# Reasons for prototyping

---

- learning by doing
- improved communication
- improved user involvement
- a feedback loop is established
- reduces the need for documentation
- reduces maintenance costs i.e. changes after the application goes live
- prototype can be used for producing expected results

# Prototyping: some dangers

---

- users may misunderstand the role of the prototype
- lack of project control and standards possible
- additional expense of building prototype
- focus on user-friendly interface could be at expense of machine efficiency

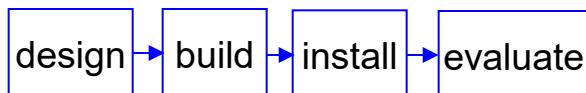
# Categories of Prototypes

---

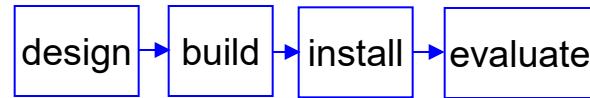
- Throw-away Prototypes
  - ‘experimental’ (HW/SW prototype)
  - ‘exploratory’ (application prototype)
- Re-usable Prototype
  - ‘evolutionary’ build (for continuous feedback)
  - Final prototype = Deliverable!

# Incremental delivery

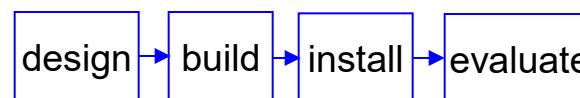
delivered system



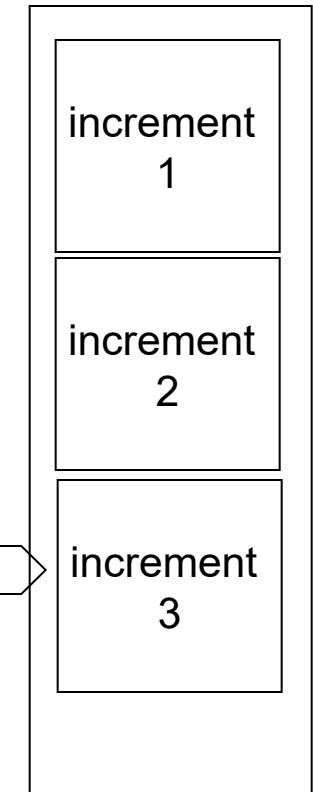
first incremental delivery



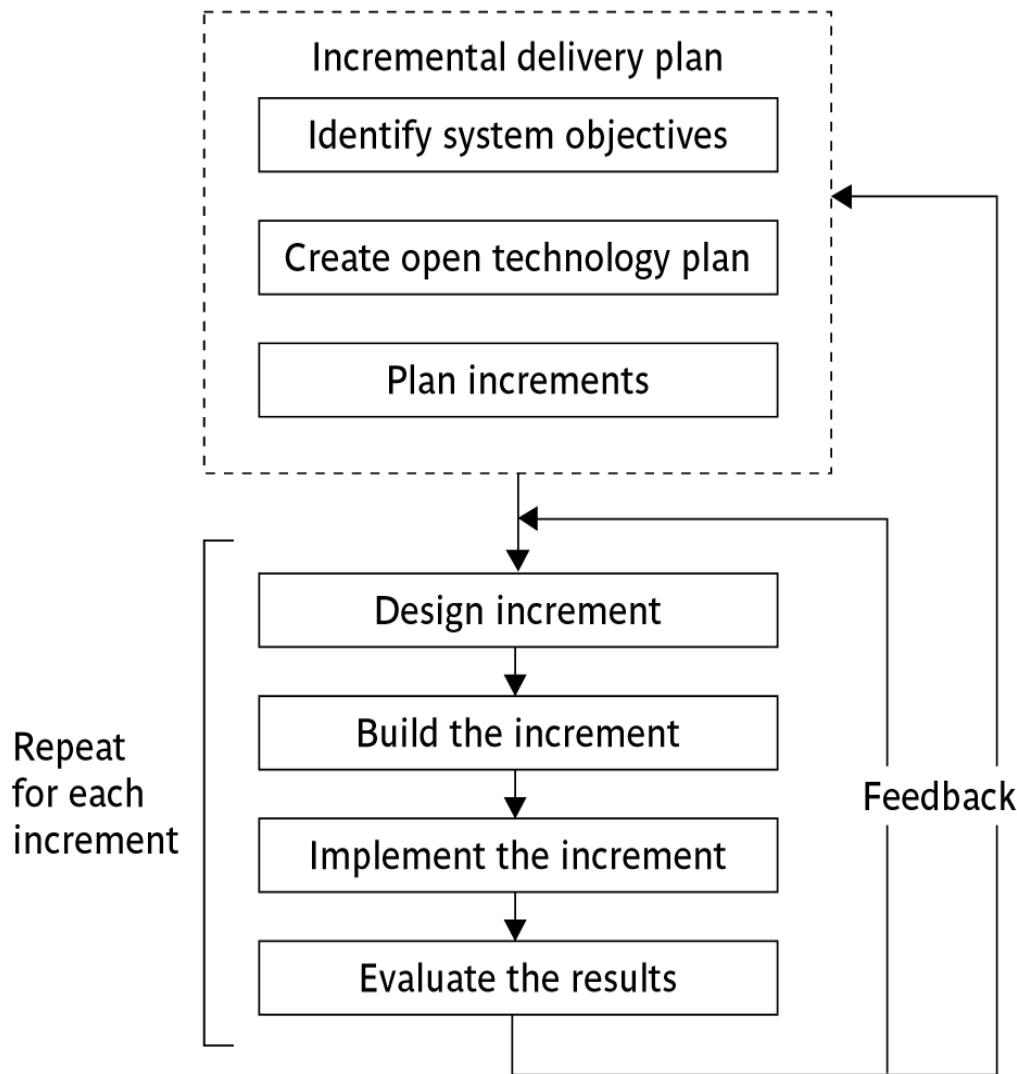
second incremental delivery



third incremental delivery



# The incremental process



# Incremental approach: benefits

---

- feedback from early stages used in developing latter stages
- shorter development thresholds
- user gets some benefits earlier
- project may be put aside temporarily
- reduces 'gold-plating':

BUT there are some possible disadvantages

- loss of economy of scale
- 'software breakage'

# Incremental planning

---

- steps ideally 1% to 5% of the total project
- ideal if a step takes one month or less:
  - not more than three months
- each step should deliver some benefit to the user
- some steps will be physically dependent on others

# ‘Agile’ methods

---

structured development methods have some perceived advantages

- produce large amounts of documentation which can be largely unread
- documentation has to be kept up to date
- division into specialist groups and need to follow procedures stifles communication
- users can be excluded from decision process
- long lead times to deliver anything etc. etc

The answer? ‘Agile’ methods!

# Time-boxing

---

- *time-box* fixed deadline by which *something* has to be delivered
- typically two to six weeks
- MOSCOW priorities
  - Must have - essential
  - Should have - very important, but system could operate without
  - Could have
  - Want - but probably won't get!

# Extreme Programming (XP)

---

- increments of one to three weeks
  - customer can suggest improvement at any point
- argued that distinction between design and building of software are artificial
- code to be developed to meet current needs only
- frequent re-factoring to keep code structured
- developers work in pairs
- test cases and expected results devised before software design
- after testing of increment, test cases added to a consolidated set of test cases

# Limitations of XP

---

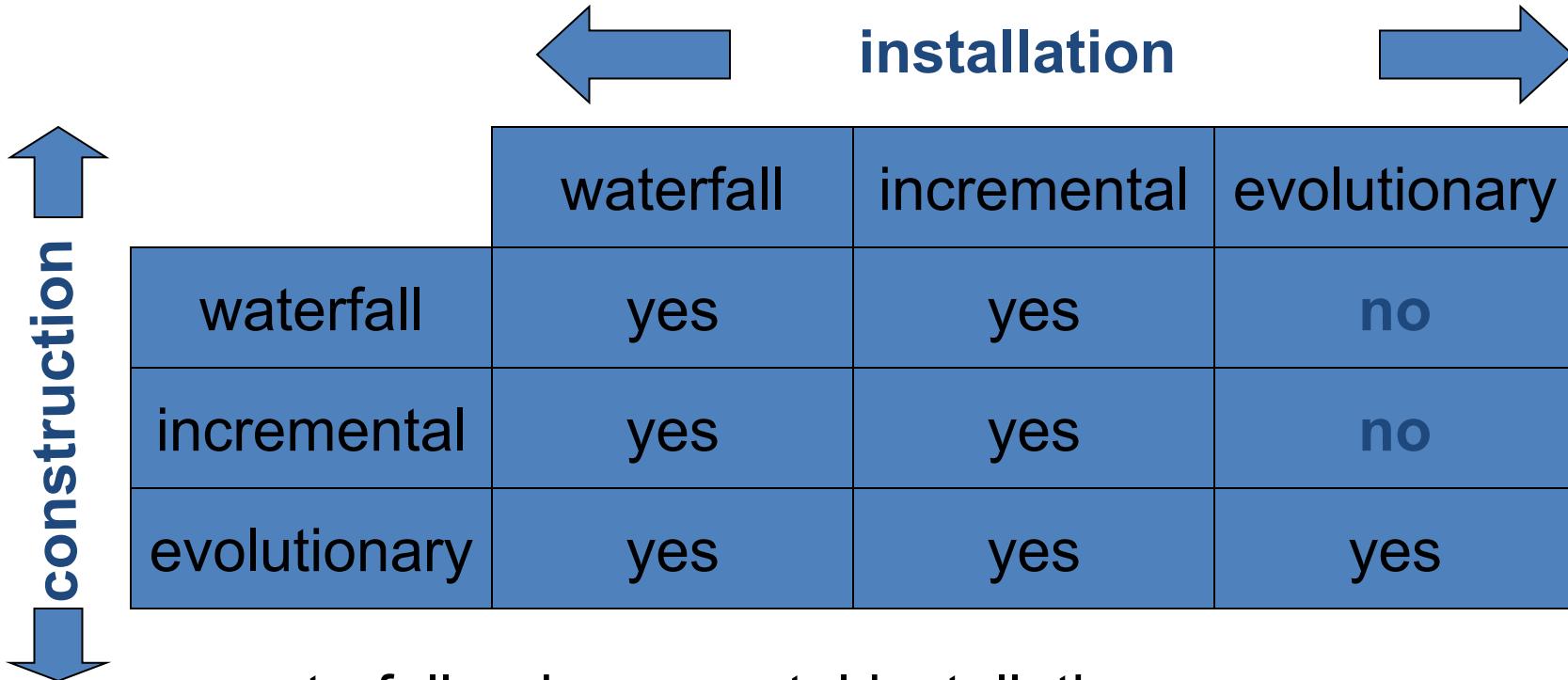
- Reliance on availability of high quality developers
- Dependence on personal knowledge – after development knowledge of software may decay making future development less easy
- Rationale for decisions may be lost e.g. which test case checks a particular requirement
- Reuse of existing code less likely

---

*'Conceptual integrity sometimes suffers because there is little motivation to deal with scalability, extensibility, portability, or reusability beyond what any vague requirement might imply'*

- Grady Booch (Authority on OO)

# A Combination of approaches?



- waterfall or incremental installation – any construction approach possible
- evolutionary installation implies evolutionary construction

# ***Summary: 'rules of thumb' about which approach to be used***

---

IF uncertainty is high

THEN use evolutionary approach

IF complexity is high but uncertainty is not

THEN use incremental approach

IF uncertainty and complexity both low

THEN use Water-fall model

IF schedule is tight

THEN use evolutionary or incremental

---

# Thank You



**BITS Pilani**

Pilani | Dubai | Goa | Hyderabad

# **Software Project Management – Caselet 1- SDLC methodology choice.**

**S Subramanian ( Subbu)**

# Caselet – Scenario

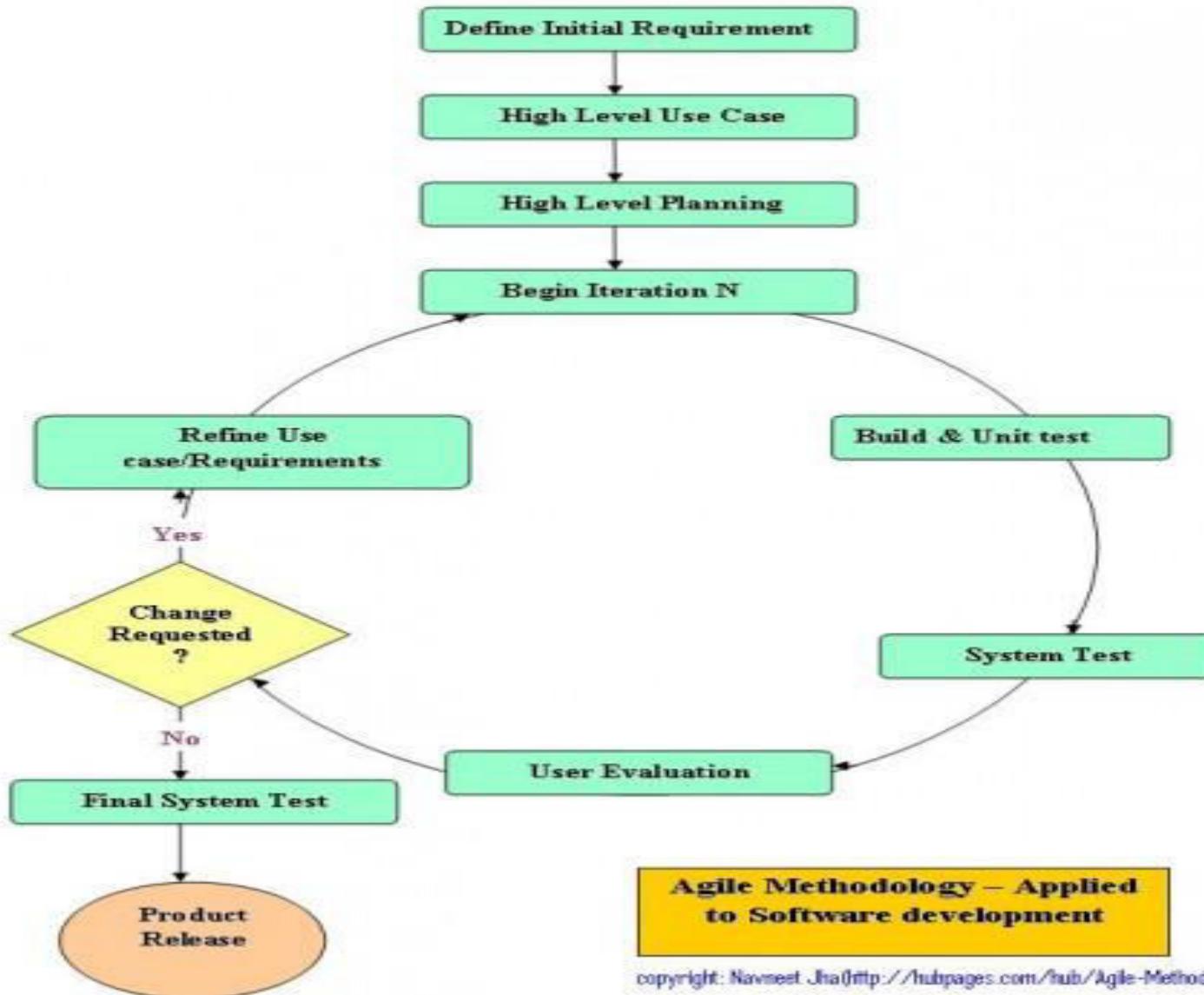
---

- Customer is demanding a particular SDLC execution methodology – AGILE
- Your company's experience in AGILE is limited
- You (PM) are not sure of client's maturity in AGILE process
- Your Sales team has already sold AGILE expertise to the client (*so you can't back out*)
- Either you follow AGILE or you don't lead the project – choice is yours.

# Cost of Change

- Agile- governance
  - The word “steering” implies active management involvement and frequent course-correction to produce better results. All stakeholders must collaborate to converge on moving targets, and the principles above delineate the economic foundations necessary to achieve good steering mechanisms
- The cost of change in a real-world enterprise system is never that small. You must plan for change, and understand its costs.

# AGILE Process



**Agile Methodology – Applied to Software development**

# Strong Architecture

---

- You must deliver **an architecture which can accommodate likely change in the best way for the enterprise**
  - It reduces the cost of change,
  - It provides a reference against which the sillier requirements can be challenged,
  - It can provide mechanisms which allow incorrect requirements to be rapidly reversed out

# AGILE – Project management

---

- Client must embrace Agile process
    - Continuous and serious involvement from client.
  - Daily stand up meeting
  - Empowered development teams
    - Strong PM is mandated
  - Estimation & Costing
    - Prefer T&M contracts over Fixed Price
  - Strong Change Control Board
-

# Summary

---

- For Agile process to succeed:
  - Get high level business requirements in as much detail as possible
  - Track Changes rigorously
  - Build strong Architecture upfront ( CBD)
  - Make sure at least 70% of your team size is experienced
  - Use good estimation process ( UCP/ Story Board)
  - Prefer T&M contract
  - It is a two way street, make sure client is actively involved throughout SDLC.
  - Follow SCRUM process.





**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

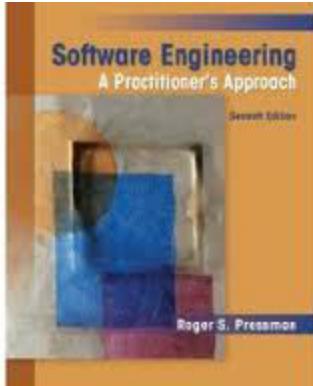
# SS ZG622: Software Project Management (Project Goals, Metrics)

K G Krishna, BITS-Pilani, Off-campus Centre, Hyderabad



# **Project Goal Setting (Objectives) & Metrics**

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations



# Software Project Categories

---

Distinguishing different types of project is important as different types need different project approaches/development models e.g.

- Commercial systems (ERP Systems)
- Consumer systems (video games, mobile apps)
- Embedded Systems
- Real-Time Systems
- Mission-critical Systems
- Custom developed (bespoke) Systems
- Software Products (off-the-shelf)

# Project Stakeholders

---

These are people who have a stake or interest in the project  
In general, they could be *users/customers* or  
*developers/implementers*

- Customer: Key Stakeholder
- Others could be:
  - Within the project team (Team members, PM)
  - Outside the project team, but within the same organization (Finance, Business,...)
  - Outside both the project team and the organization (society)

Different stakeholders may have different objectives – need to define common project objectives

# Setting Project Goals (Objectives)

---

- Answering the question '*What do we have to do to have a success?*'
- Set by a *project authority*
  - Sets the project scope
  - Allocates/approves costs
  - Could be one person - or a group
    - Project Board
    - Project Management Board
    - Steering committee

# Objectives

---

*Informally, the objective of a project can be defined by completing the statement:*

*The project will be regarded as a success  
if.....*

.....

Focus on what will be put in place, rather than  
*how* activities will be carried out

# Objectives should be SMART

---

S – specific, that is, concrete and well-defined

M – measurable, that is, satisfaction of the objective can be objectively judged

A – achievable, that is, it is within the power of the individual or group concerned to meet the target

R – relevant, the objective must relevant to the true purpose of the project

T – time constrained: there is defined point in time by which the objective should be achieved

# Goals/sub-objectives

---

These are steps along the way to achieving the objective

Informally, these can be defined by completing the sentence

To reach objective X, the following must be in place

A.....

B.....

C..... etc

# Goals/sub-objectives ...continued

---

- Often a goal can be allocated to an individual
- Individual might have the capability of achieving goal on their own, but not the overall objective e.g.

*Overall objective* – user satisfaction with software product

*Analyst goal* – accurate requirements

*Developer goal* – reliable software

# Measures of effectiveness

---

- How do we know that the goal or objective has been achieved?
- By a practical test, that can be objectively assessed.
  - e.g. for user satisfaction with software product:
  - Repeat business – they buy further products from us
  - Number of complaints – if low etc etc

# The Business Case (Cost vs. Benefit)

Benefits



Costs



Benefits of delivered project must outweigh costs

Costs include:

- Development
- Operation

Benefits

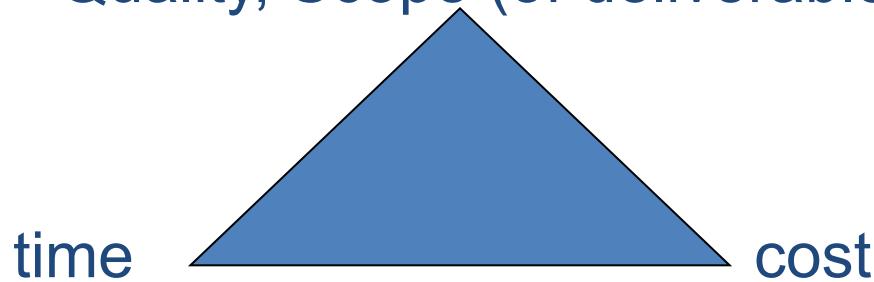
Quantifiable

Non-quantifiable

# Project Success/Failure

---

- Degree to which objectives are met  
Quality, Scope (of deliverables)



In general if, for example, project is running out of time, this can be recovered for by reducing scope or increasing costs. Similarly costs and scope can be protected by adjusting other corners of the 'project triangle'.

# Other success criteria

---

These can relate to longer term, less directly tangible assets

- Improved skill and knowledge
- Creation of assets that can be used on future projects e.g. software libraries
- Improved customer relationships that lead to repeat business

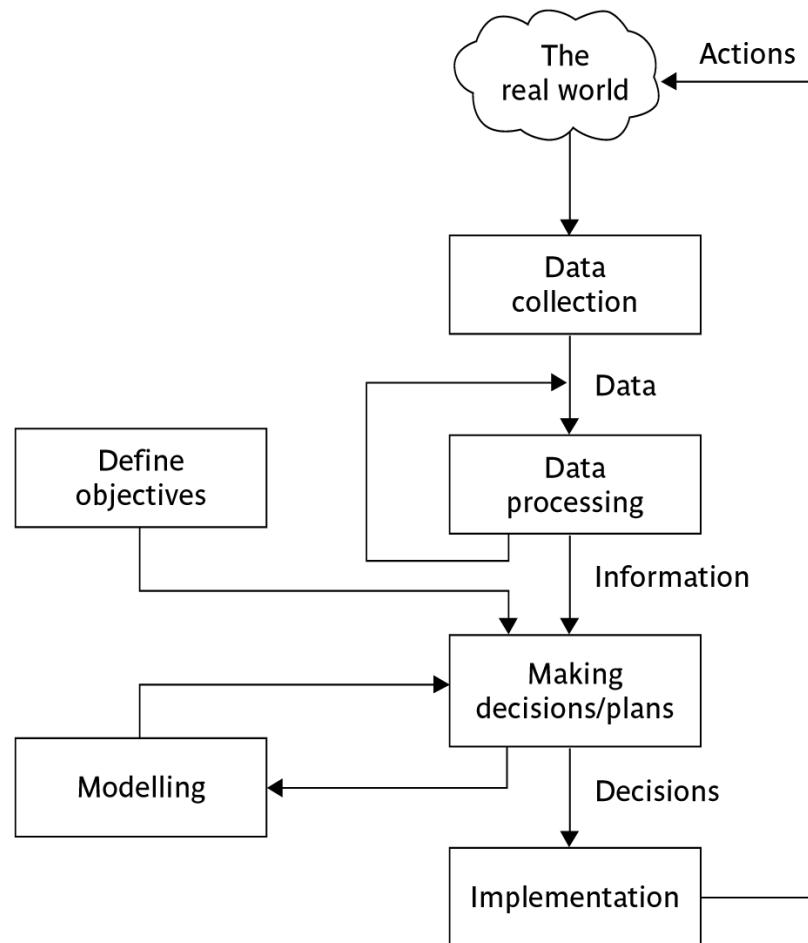
# Managing for Success

---

## Management is all about:

- **Planning** – deciding what is to be done
- **Organizing** – making arrangements
- **Staffing** – selecting the right people for the job (motivating for performance)
- **Directing** – giving instructions
- **Measuring** – Defining Metrics and establishing baselines
- **Monitoring** – checking on progress
- **Controlling** – taking action to remedy bottle-necks
- **Innovating** – coming up with solutions when problems emerge
- **Representing** – liaising with customers, users, developers and other stakeholders
- ...

# Data, Measurements & Management control



# Management control

---

**Data** – the raw details

e.g. '*6,000 documents processed at location X*'

**Information** – the data is processed to produce something that is meaningful and useful

e.g. '*productivity is 100 documents a day*'

**Comparison** with objectives/goals

e.g. *we will not meet target of processing all documents by 31<sup>st</sup> March*

**Modelling** – working out the probable outcomes of various decisions

e.g. if we employ two more staff at location X how quickly can we get the documents processed?

**Implementation** – carrying out the remedial actions that have been decided upon

# Objectives, Project Control: Summary



- ❖ Projects are non-routine - thus uncertain
- ❖ Clear objectives which can be objectively assessed are essential
- ❖ Things work! Not usually possible to keep precisely plan – need for control
- ❖ Communicate, communicate, communicate!

---

# Thank You



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# SS ZG622:

## Software Project Management

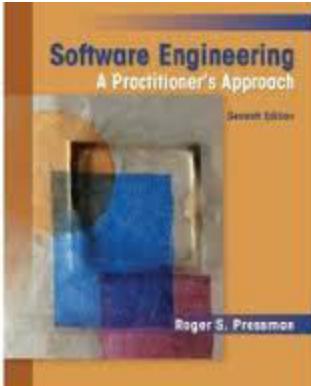
### (Software Measurements, Metrics Baseling)

K G Krishna, BITS-Pilani, Off-campus Centre, Hyderabad



# **Software Measurements & Metrics Baseling**

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations



# Categories of Software Measurement

---

- Two categories of software measurement
  - Direct measures of the
    - Software process (cost, effort, etc.)
    - Software product (lines of code produced, execution speed, defects reported over time, etc.)
  - Indirect measures of the
    - Software product (functionality, quality, complexity, efficiency, reliability, maintainability, etc.)
- Project metrics can be consolidated to create process metrics for an organization

# Size-oriented Metrics

---

- Derived by normalizing quality and/or productivity measures by considering the size of the software produced
- Thousand lines of code (KLOC) are often chosen as the normalization value
- Metrics include
  - Errors per KLOC
  - Defects per KLOC
  - Dollars per KLOC
  - Pages of documentation per KLOC
  - Errors per person-month
  - KLOC per person-month
  - Dollars per page of documentation

# Size-oriented Metrics (continued)

---

- Size-oriented metrics are not universally accepted as the best way to measure the software process
- Opponents argue that KLOC measurements
  - Are dependent on the programming language
  - Penalize well-designed but short programs
  - Cannot easily accommodate nonprocedural languages
  - Require a level of detail that may be difficult to achieve

# Function-oriented Metrics

---

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value
- Most widely used metric of this type is the function point:

$$FP = \text{count total} * [0.65 + 0.01 * \text{sum (value adj. factors)}]$$

- Function point values on past projects can be used to compute, for example, the average number of lines of code per function point (e.g., 60)

# Typical Function-Oriented Metrics

---

- errors per FP
- defects per FP
- Rs. per FP
- pages of documentation per FP
- FP per person-month

# Why Opt for FP?

---

- Programming language independent
- Used readily countable characteristics that are determined early in the software process
- Does not “penalize” inventive (short) implementations that use fewer LOC than other more clumsy versions
- Makes it easier to measure the impact of reusable components

# Function Point Controversy

---

- Proponents claim that
  - FP is programming language independent
  - FP is based on data that are more likely to be known in the early stages of a project, making it more attractive as an estimation approach
- Opponents claim that
  - FP requires some “sleight of hand” because the computation is based on subjective data
  - FP has no direct physical meaning...it’s just a number

# Reconciling LOC and FP Metrics

---

- Relationship between LOC and FP depends upon
  - The programming language that is used to implement the software
  - The quality of the design
- FP and LOC have been found to be relatively accurate predictors of software development effort and cost
  - However, a historical baseline of information must first be established
- LOC and FP can be used to estimate object-oriented software projects
  - However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process
- The table on the next slide provides a rough estimate of the average LOC to one FP in various programming languages

# LOC Per Function Point

Language	Average	Median	Low	High
Ada	154	--	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
PL/1	78	67	22	263
Visual Basic	47	42	16	158

Source: [www.qsm.com/?q=resources/function-point-languages-table/index.html](http://www.qsm.com/?q=resources/function-point-languages-table/index.html)

# Object-oriented Metrics

- Number of scenario scripts (i.e., use cases)
  - This number is directly related to the size of an application and to the number of test cases required to test the system
- Number of key classes (the highly independent components)
  - Key classes are defined early in object-oriented analysis and are central to the problem domain
  - This number indicates the amount of effort required to develop the software
  - It also indicates the potential amount of reuse to be applied during development
- Number of support classes
  - Support classes are required to implement the system but are not immediately related to the problem domain (e.g., user interface, database, computation)
  - This number indicates the amount of effort and potential reuse

— Lorenz & Kidd

# Object-oriented Metrics (continued)

- Average number of support classes per key class
  - Key classes are identified early in a project (e.g., at requirements analysis)
  - Estimation of the number of support classes can be made from the number of key classes
  - GUI applications have between two and three times more support classes as key classes
  - Non-GUI applications have between one and two times more support classes as key classes
- Number of subsystems
  - A subsystem is an aggregation of classes that support a function that is visible to the end user of a system
    - Lorenz & Kidd

# WebApp Project Metrics

---

- Number of **static Web pages** (the end-user has no control over the content displayed on the page)
- Number of **dynamic Web pages** (end-user actions result in customized content displayed on the page)
- Number of **internal page links** (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp)
- Number of **persistent data objects**
- Number of **external systems interfaced**
- Number of **static content objects**
- Number of **dynamic content objects**
- Number of **executable functions**

# Measuring Quality

---

- **Correctness** — the degree to which a program operates according to specification
- **Maintainability**—the degree to which a program is amenable to change
- **Integrity**—the degree to which a program is impervious to outside attack
- **Usability**—the degree to which a program is easy to use

— Gilb

# Example Metrics for Quality

---

- Correctness
  - Number of defects per KLOC, where a defect is a verified lack of conformance to requirements
  - Defects are those problems reported by a program user after the program is released for general use
- Maintainability
  - Ease with which a program can be corrected if an error is found, adapted if the environment changes, or enhanced if the customer has changed requirements
  - Mean time to change (MTTC) : the time to analyze, design, implement, test, and distribute a change to all users
    - Maintainable programs on average have a lower MTTC
- ...

# Defect Removal Efficiency

---

- Defect removal efficiency provides benefits at both the project and process level
- It is a measure of the filtering ability of QA activities as they are applied throughout all process framework activities
  - It indicates the percentage of software errors found before software release
- It is defined as  $DRE = E / (E + D)$ 
  - E is the number of errors found before delivery of the software to the end user
  - D is the number of defects found after delivery
- As D increases, DRE decreases (i.e. becomes a smaller and smaller fraction)
- The ideal value of DRE is 1, which means no defects are found after delivery
- DRE encourages a software team to institute techniques for finding as many errors as possible before delivery

---

# Integrating Metrics within the Software Process

# Arguments for Software Metrics

---

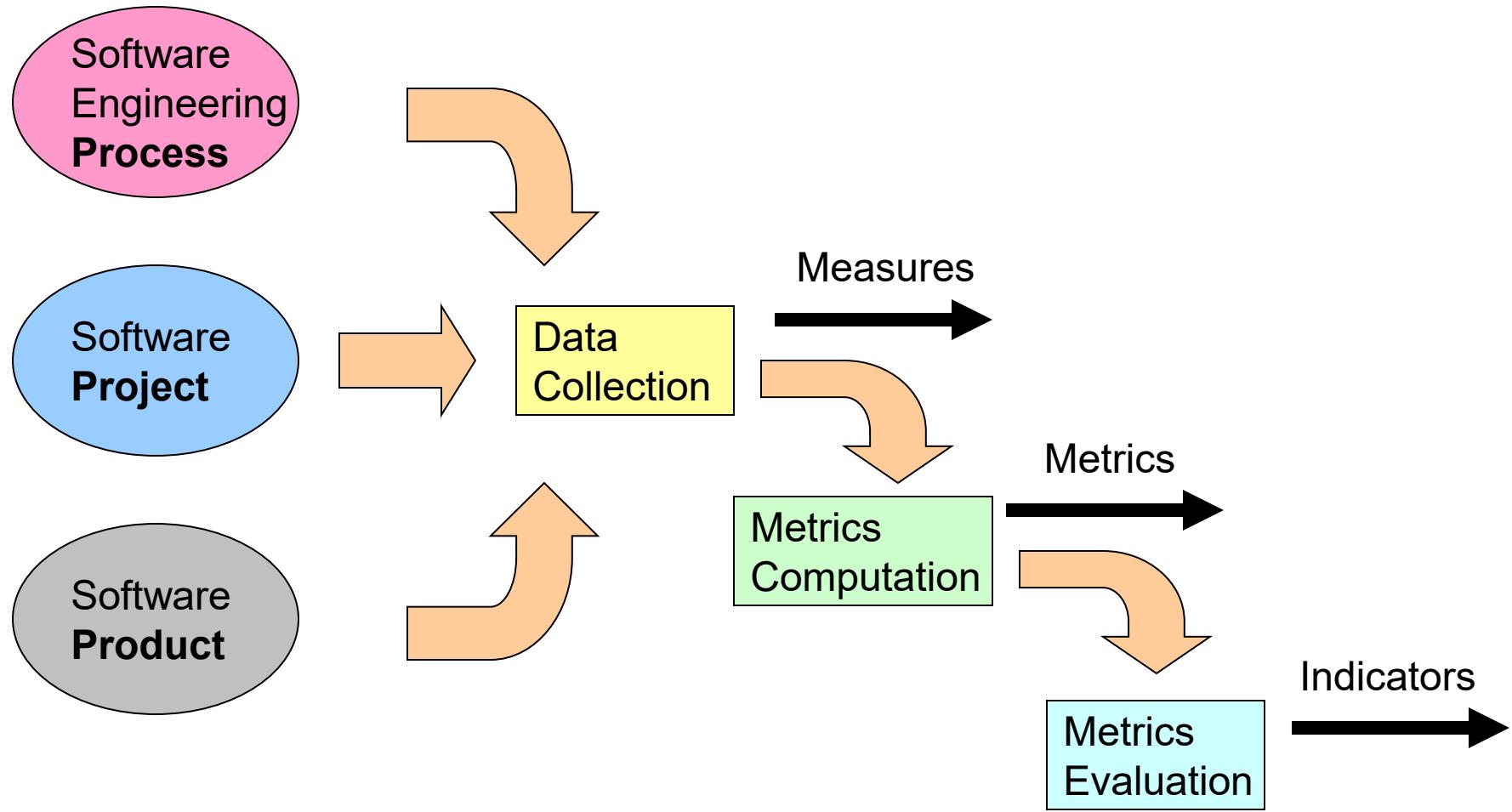
- Most software developers do not measure, and most have little desire to begin
- Establishing a successful company-wide software metrics program can be a multi-year effort
- But if we do not measure, there is no real way of determining whether we are improving
- Measurement is used to establish a process baseline from which improvements can be assessed
- Software metrics help people to develop better project estimates, produce higher-quality systems, and get products out the door on time

# Establishing a Metrics Baseline

---

- By establishing a metrics baseline, benefits can be obtained at the software process, product, and project levels
- The same metrics can serve many masters
- The baseline consists of data collected from past projects
- Baseline data must have the following attributes
  - Data must be reasonably accurate (guesses should be avoided)
  - Data should be collected for as many projects as possible
  - Measures must be consistent (e.g., a line of code must be interpreted consistently across all projects)
  - Past applications should be similar to the work that is to be estimated
- After data is collected and metrics are computed, the metrics should be evaluated and applied during estimation, technical work, project control, and process improvement

# Software Metrics Baseline Process



# Establishing a Metrics Program

---

- Identify your business goals.
- Identify what you want to know or learn.
- Identify your subgoals.
- Identify the entities and attributes related to your subgoals.
- Formalize your measurement goals.
- Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals.
- Identify the data elements that you will collect to construct the indicators that help answer your questions.
- Define the measures to be used, and make these definitions operational.
- Identify the actions that you will take to implement the measures.
- Prepare a plan for implementing the measures.

– SEI Guidebook

# Getting Started with Metrics

(as per Software Productivity Center)



## 1) Understand your existing process

- Identify Framework activities
- Input information for each activity
- Tasks associated with each activity
- Quality assurance functions
- Work products

## 2) Define the goals to be achieved by establishing a metrics program

- Improve accuracy of estimation
- Improve product quality

## 3) Identify metrics to achieve those goals

- Keep the metrics simple
- Be sure the metrics add value to your process and product

## 4) Identify the measures to be collected to support those metrics

(contd.)

# Getting Started with Metrics

(continued)

---

- 5) Establish a measurement collection process
  - a) What is the source of the data?
  - b) Can tools be used to collect the data?
  - c) Who is responsible for collecting the data?
  - d) When are the data collected and recorded?
  - e) How are the data stored?
  - f) What validation mechanisms are used to ensure the data are correct?
- 6) Acquire appropriate tools to assist in collection and assessment
- 7) Establish a metrics database
- 8) Define appropriate feedback mechanisms on what the metrics indicate about your process so that the process and the metrics program can be improved

# Metrics for Small Organizations

---

- time (hours or days) elapsed from the time a request is made until evaluation is complete,  $t_{queue}$ .
- effort (person-hours) to perform the evaluation,  $W_{eval}$ .
- time (hours or days) elapsed from completion of evaluation to assignment of change order to personnel,  $t_{eval}$ .
- effort (person-hours) required to make the change,  $W_{change}$ .
- time required (hours or days) to make the change,  $t_{change}$ .
- errors uncovered during work to make change,  $E_{change}$ .
- defects uncovered after change is released to the customer base,  $D_{change}$ .

# Summary of Process & Project Metrics

---

- Process and Project metrics are quantitative measures to gain insight into the efficiency of the software process and the projects conducted using the process framework.
- Software project management is primarily concerned with productivity and quality metrics.
- There are four reasons for measuring software processes, products, and resources
  - to characterize, to evaluate, to predict, and to improve
- *Process metrics* used to provide indicators that lead to long term process improvement
- *Project metrics* enable project manager to
  - Assess status of ongoing project
  - Track potential risks
  - Uncover problems before they go critical
  - Adjust work flow or tasks
  - Evaluate the project team's ability to control quality of software work products

---

# Thank You



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Introduction to Software Estimation

BITS Pilani  
Viswanathan Hariharan  
2015



# Agenda

---

- Purpose of estimation
- Importance of accurate estimation

# Example situations of software estimation

---

- A Marketing manager wants to know whether his budget for developing a Customer Management system is sufficient or not
- A company wants to compare the estimates provided by different vendors

# State of software project execution

---



- IT industry's track record of software project execution is very poor
- According to Standish group more than 70% projects fail – that is, they either exceed estimates significantly or are abandoned
- One of the factors contributing to project failures is poor estimation

# Under-estimation

---

- Generally IT professionals under-estimate the effort
- This leads to many issues
  - Team stress
  - More defects
  - Schedule slippage
  - Fire fighting
  - Loss in business benefit (eg. Productivity)

# Over-estimation

---

- Over-estimation is not the answer for these issues
- Over-estimation also has de-merits
  - Loss of business to competition
  - Parkinson's Law comes into effect – “Work expands to fill available time”

# Why do estimates go wrong?

---

- In order to estimate accurately we need to understand the causes that lead to inaccurate estimates
- Main causes are:
  - Lack of detailed requirements
  - Omitted activities such as performance testing, data migration, etc.
  - Optimistic thinking, such as, we have learnt from previous projects
  - Lack of systematic approach to estimation

# Benefits of accurate estimates

---

- Accurate estimates can lead to:
  - Better quality software
  - Ability to identify risks when project slips
  - Higher chance of completing project on time
  - All leading to increased customer satisfaction



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Software Estimation Techniques

BITS Pilani  
Viswanathan Hariharan  
2015



# Estimation techniques

---

## Agenda

- Introduction
- Top down estimation
- Bottom up estimation

# Introduction

---

- There are many estimation techniques such as
  - Expert judgement,
  - Estimation by analogy,
  - Delphi method,
  - etc.
- However the most popular ones are
  - Top down &
  - Bottom up approaches

# Bottom up estimation

---

- For small projects (eg. 3 month project with 5 team members) a bottom up approach is generally used
- The project is decomposed into its activities, also known as Work Breakdown Structure (WBS)
- For each activity, the effort is estimated

# Example of Work Breakdown Structure (WBS)

Activity	Effort (person days)
User interface development	
Design	5
Coding	15
Unit testing	3
Business components development	
Architecture design	5
Detailed design	
Inventory	2
Purchase order	2
Shipment	2
Invoice	2
Payment	2
Coding & Unit testing	
Inventory	6
Purchase order	6
Shipment	6
Invoice	6
Payment	6
Integration testing	10
System testing	10
Deployment	3
Total	91

# Bottom up estimation...

---

While estimating the effort for each activity, decompose the activity into smaller pieces

- Instead of estimating a large activity, decompose it into smaller activities and estimate each small activity
- All estimates have some errors - some are +ve errors and some –ve errors
- When we have a large number of activities, these errors tend to cancel each other
- This is called the “Law of large numbers”
- Typically we need 5-10 numbers for the Law to apply

# Top down estimation

---

- In this technique, we first determine the size of the software
  - Example: Let us say an application has 25 screens and 10 reports. This gives a sense of the size of the software
- We then convert the size into effort, based on historical data
  - Example:
  - Let us say, we have collected historical data on the effort per screen and report
  - Based on historical data, let us say , the average effort for developing one screen is 5 days
  - And let us say, the average effort for developing one report is 3 days
  - Then, the Total effort = 25 screens x 5 days + 10 reports x 3 days = 105 person days

# Determining the size

---

- Size can be determined in many ways, such as:
  - Use cases
  - Functions & data files
  - Objects – Screens, reports, etc.
  - Web pages
  - Test cases
  - Defects to be fixed
  - Configuration settings

# Example of historical data

**Table 7-1 Examples of Quantities That Can Be Counted for Estimation Purposes**

Quantity to Count	Historical Data Needed to Convert the Count to an Estimate
Features	<ul style="list-style-type: none"> <li>■ Average effort hours per feature for development and/or testing</li> </ul>
Use cases	<ul style="list-style-type: none"> <li>■ Average total effort hours per use case</li> <li>■ Average number of use cases that can be delivered in a particular amount of calendar time</li> </ul>
Stories	<ul style="list-style-type: none"> <li>■ Average total effort hours per story</li> <li>■ Average number of stories that can be delivered in a particular amount of calendar time</li> </ul>
Engineering requirements	<ul style="list-style-type: none"> <li>■ Average number of engineering requirements that can be formally inspected per hour</li> <li>■ Average effort hours per requirement for development/test/documentation</li> </ul>
Function Points	<ul style="list-style-type: none"> <li>■ Average development/test/documentation effort per Function Point</li> <li>■ Average lines of code in the target language per Function Point</li> </ul>
Change requests	<ul style="list-style-type: none"> <li>■ Average development/test/documentation effort per change request (depending on variability of the change requests, the data might be decomposed into average effort per small, medium, and large change request)</li> </ul>
Web pages	<ul style="list-style-type: none"> <li>■ Average effort per Web page for user interface work</li> <li>■ Average whole-project effort per Web page (less reliable, but can be an interesting data point)</li> </ul>

# Data collection

- Collect effort for all activities and all items (screen, report, etc.) including
  - Requirement gathering
  - Architecture
  - Detailed design
  - Coding & Unit testing
  - Design of test cases
  - Integration testing
  - System testing
  - User acceptance testing
  - Project Management
  - User documentation
  - Reviews of requirement, design, code, test cases, user manual, etc.
- Collect data while project is underway. It is difficult to go back six months in the project to try to collect data
- If you do not have historical data, use data from current project, as a basis to estimate the remainder of the project

# Calibration

---

- Based on data collected, we can calibrate our productivity, ie. Effort for unit size
- Calibration examples
  - Our team takes 8 hours for design & 24 hours to code one screen
  - Our testers take 6 hours to create a test case
  - Our maintenance team takes 4 hours to fix a defect

# Factors influencing effort

---

## 1. Project size:

- Project size has the biggest influence on effort estimation
- Effort increases exponentially (not linearly) as project size increases because there is much more coordination in larger projects (this is opposite of 'Economies of scale' concept)

## 2. Kind of software being developed

- Avionics, embedded systems and System software are much more complex than business systems and internet systems

## 3. Personal factors impact estimation

- Capability of requirements analyst, capability of programmer, change of team members, level of domain knowledge, platform experience, team cohesion

# Summary

---

- We looked at 2 primary ways of estimating effort of software projects, namely, top down & bottom up
- In top down approach
  - We count the number of items to determine the size of software
  - We then convert the size into effort based on historical data
- Bottom up approach is used for small projects
  - Here we use work breakdown structure (WBS) to identify different activities
  - We then estimate the effort for each activity



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Function point estimation technique

BITS Pilani  
Viswanathan Hariharan  
2015



# Function point (FP) estimation technique

---

- FP technique is used in the early stages of the project to determine the size of the project based on software requirements
- The size of software is measured in Function Point units
- Once the size is estimated in FPs, we can convert FPs to Effort, based on historical data

# Function point (FP) technique

---

- The calculation of size is based on functions supported by the software and the data files used by the software
- Functions are of 3 types
  - External input:
  - External output
  - External inquiry
- Data files are of 2 types
  - Internal logical files
  - External logical files

# Types of Functions

---

- **External Input (EI):** These are functions that modify the data of the system.
  - Examples: Create Purchase order, Withdraw money from ATM, Update price of a product
- **External Output (EO):** These are functions that send data to external systems or do some processing of data and provide to end users.
  - Examples: Exporting data to other systems via files or messages, Generating reports involving analysis such as average
- **External Query (EQ):** These are simple inquiry functions that extract data from the system and present to user.
  - Examples: Show availability of seats in a flight, generating a list of all orders to be shipped tomorrow

# Types of Data

---

- Internal Logical Files: These are data files (database tables or flat files) maintained by the system and that reside within the system.
  - Example: Employee file, Orders file, Payment file, Inventory issue file
- External Interface Files: These are data files (database tables or flat files) maintained outside the system but needed by our system.
  - Example: Schedule of flights of other airlines, Interest rate offered by different banks

# Functions Points

---

## Steps to calculate Function points

1. Assign FPs to each Function & each File based on its complexity
2. Total the FPs. This is called Unadjusted FP or Raw FPs
3. Multiply the Unadjusted FPs by an adjustment factor. The adjustment factor is determined / based on 14 characteristics of the system

# FPs for Functions & Files

Here are the FPs for different types of functions and files, based on their complexity

Program Characteristic	Low Complexity	Medium Complexity	High Complexity
External Inputs	— × 3	— × 4	— × 6
External Outputs	— × 4	— × 5	— × 7
External Queries	— × 3	— × 4	— × 6
Internal Logical Files	— × 4	— × 10	— × 15
External Interface Files	— × 5	— × 7	— × 10

**Unadjusted Function Point (UPFs) = Sum of EI FP, EO FP, EQ FP, ILF FP, EIF FP**

# Determining complexity of EI, EO & EQ

---



- Complexity of functions is determined based on number of Data Element Types referenced and File Types referenced, by the function
  - Example #1: The function “Create purchase order” (EI), updates the fields – PO#, Date, Description, PO amount, Customer ID, Customer name, etc. These are the DETs maintained by the function
  - Example #2: The report “Generate Sales analysis report” (EO) displays the following fields on the report – Product ID, Product name, City, Month, Total quantity sold, Total Sales revenue, Name of Salesman. These fields are the DETs of this function. In order to perform this function, it uses 3 files namely – Sales file, Product file and Salesman file. These are the File Types referenced

# Determining complexity of EI, EO & EQ

- Here is the matrix to determine the complexity

**Data Element Types (DETs)** – user-recognizable data elements that are maintained as ILFs by the EI, appear in the EO, EQs (data fields, buttons, search).

**File Types Referenced (FTRs)** – all ILFS and EIFs referenced or maintained during the processing of the EI/EO/EQs (tables, flat files).

EI	1-4 DETs	5-15 DETs	16 + DETs
0-1 FTR	Low	Low	Average
2 FTRs	Low	Average	Average
3+ FTRs	Average	High	High

EO/EQ	1-5 DETs	6-19 DETs	19+ DETs
0-1 FTR	Low	Low	Average
2-3 FTRs	Low	Average	High
4+ FTRs	Average	High	High

# Determining complexity of files

---



- Complexity of files is determined based on 2 characteristics of the file namely - Data Element Types (DET) and Record Element Types (RET)
- The meaning of DET & RET is explained below:
- Example:
  - Employee file can have the following logical groupings
    - Basic data of employee such as Name, Date of birth, Address, etc.
    - Past employment info such as Name of company, Start date, end date, Last designation, etc.
    - Past project info such as Name of project, From date, To date, Role in project, etc.
  - In the above example,
    - the DETs are Name, date of birth, Address, Name of company, From date, etc.
    - We have 3 types of logical grouping of data namely - Basic data group, Past employment info group, Past project info group. Hence we have 3 RETs

# Determining complexity of LIF & EIF



- Here is the matrix to determine the complexity

**Data Element Types (DETs)** – Unique, user-recognizable, non-repeating fields or attributes, including foreign key attributes that enter the boundary of the subsystem or application (fields).

**Record Element Types (RETs)** – Logical sub-groupings based on the user's view of the data (0 RET = 1 RET).

ILF/EIF	1-19 DETs	20-50 DETs	51 + DETs
1 RET	Low	Low	Average
2-5 RETs	Low	Average	High
6+ RETs	Average	High	High

# Adjustment factor

- Adjustment factor depends on the rating of 14 system characteristics
- We need to rate each characteristic on a scale of 0-5
- The adjustment factor is calculated using a formula shown below

F1	Reliable back-up and recovery	0 – 5
F2	Data communications	0 – 5
F3	Distributed functions	0 – 5
F4	Performance	0 – 5
F5	Heavily used configuration	0 – 5
F6	Online data entry	0 – 5
F7	Operational ease	0 – 5
F8	Online update	0 – 5
F9	Complex interface	0 – 5
F10	Complex processing	0 – 5
F11	Reusability	0 – 5
F12	Installation ease	0 – 5
F13	Multiple sites	0 – 5
F14	Facilitate change	0 – 5
<b>Value adjustment factor (VAF) = <math>0.65 + 0.01 * \text{Sum}(F1, F14)</math></b>		
<b>Adjusted function point (AFP) = UFPs * VAF</b>		

# Example of FP calculation

Functions	Name of Function	Type	Complexity	FP
	Inquire balance	EQ	Low	3
	Withdraw money	EI	Medium	4
	Generate statement	EO	High	7
	...			
	...			

Files	Name of File	Type	Complexity	FP
	Account file	ILF	Medium	10
	Transaction file	ILF	Low	4
	TDS rates	EIF	Medium	7

Total FPs (unadjusted) 35

Adjustment factor 1.1

Adjusted FP 38.5



# Object based estimation technique

BITS Pilani  
Viswanathan Hariharan  
2015

**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Object based estimation

---

- This is a flexible estimation approach that can be adapted to different types of projects
- For example, some projects have screens & reports as objects. Others have configurations settings as objects (ex, ERP package implementation), others have change requests as objects
- This technique determines the size of project based on number of objects and their complexity
- Please note that this is not the same as Object Point technique

# Object based estimation

---

## Steps:

1. List all objects of different categories
2. Assign points to each object based on its complexity (Low, Medium, High)
3. Determine the total points
4. Finally convert the Points to effort based on Calibrated Historical data

**Note:** The classification of objects and the way to determine their complexity needs to be done by each organization depending on the nature of projects executed

# Assigning points, based on complexity of objects: Example

Point based on complexity				
Object type	Complexity ->	Low	Medium	High
Screen		2	4	8
Report		1	2	3
Interface		2	4	8

Points will differ from organization to organization. It has to be based on historical data of effort needed to develop & test these objects

# Object based estimation: Example

Screens	Name of screen	Complexity	Points
	Account maintenance	High	8
	Funds transfer	Medium	4
	Pay bills	Low	2

Reports	Name of report	Complexity	Points
	Generate monthly statement	High	3
	Print last 10 transactions	Medium	2

Interfaces	Name of interface	Complexity	Points
	Send statement via email	Low	2
	Receive government rules	Medium	4

<b>Total Object points</b>	<b>25</b>
----------------------------	-----------

# Parameters that determine complexity of objects:



## Example

---

- Given below are some parameters that can be used to determine complexity of objects:
  - Screen complexity: can be determined based on number of fields on the screen, number of command buttons,
  - Report complexity: can be determined based on number of fields in the report, number of data files needed
  - Interface complexity: can be determined by the number of data items sent or received, number of data files needed

# Determining complexity of objects: Example

## Screen complexity determination: Example

		Number of fields		
		<5	6 - 10	> 10
Number of command buttons	1	Simple	Simple	Medium
	2 - 3	Simple	Medium	Medium
	> 3	Simple	Complex	Complex

# Summary

---

- Object based estimation is a flexible approach that can be adapted to different types of projects
- Projects need to identify suitable types of objects based on the type of project
- The size of projects are estimated based on number of objects and complexity of objects



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Planning for different phases of a project

BITS Pilani  
Viswanathan Hariharan  
2015



# Estimating effort for different phases

---

- Once the size of the project is estimated using FPs or Object points, we can convert the size into effort, based on calibrated historical data
- We can then distribute the total effort across different phases of the project, again based on historical data
- For example:
  - 7% for Requirement gathering
  - 8% for Requirement analysis
  - 20% for architecture & design
  - 45% for construction
  - 15% for System testing
  - 5% for Project management

# Estimating effort: Other considerations

---

There may be other factors to consider:

- Configuration management: 2-5% of technical effort
- Requirement change: 1-4% effort per elapse month
- First time development with new language & tool: 20- 40% effort
- Contingency depending on risk severity & probability: 5-10%

**Note: All percentages are illustrative only. Actual percentages should be based on historical data**

# Use of multiple approaches

---

- In case of large projects, we should not depend on one approach to estimation, because there could be some aspects which might have been overlooked
- Hence it is recommended to use multiple estimation techniques and compare the results
- If the difference is large then it is likely that some aspects have been overlooked
- We should try to understand the reason for the difference
- Then we should re-estimate until results converge to within 10%

# Estimation negotiation techniques

---

Project managers encounter situations where stakeholders ask for changes to estimation. Here are some ways to handle them:

1. If Marketing team tells you to reduce the cost or effort
  - Use historical data to justify the effort calculations
2. If more features are requested without adding more effort
  - Move some functions to version 2, or
  - Cut features altogether, if the business value is low
3. When you are asked to complete the project faster
  - Ask for more developers & testers & team leads, or
  - Add higher skilled technical staff, or
  - Ask what features are a 'must have' & what are not required



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# SS ZG622:

## Software Project Management

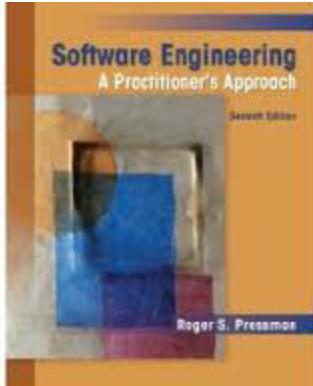
### (Software Project Planning)

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad



# Software Project Planning

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations



---

# Software Project planning

# Software Project Planning

*involves...*



- Proposal Planning
  - Estimation
  - Software Pricing
- Project Planning Process
  - Plan-driven Development
  - Agile Planning (XP/Scrum)

# Any Project Planning involves:

---

- **Work Breakdown Structure (WBS):** Breaking down the work into parts and assign these to project team members, anticipate problems that might arise and prepare tentative solutions to those problems.
- The **Project Plan**, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

# Planning Stages

---

- At the **proposal stage**, when you are bidding for a contract to develop or provide a software system.
- During the **project startup** phase, when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
- Periodically throughout the project, when you modify your plan in the light of experience gained and information from monitoring the progress of the work.

# Proposal Planning

---

- Planning may be necessary with only **outline** software requirements.
- The aim of planning at this stage is to provide information that will be used in setting a price for the system to customers.

# Software Pricing

---

- Estimates are made to discover the cost, to the developer, of producing a software system.
  - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

# Factors affecting software pricing

Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.

# Factors affecting software pricing

Factor	Description
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.

# Plan-driven development

---

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
  - Plan-driven development is based on engineering project management techniques and is the ‘traditional’ way of managing large software development projects.
- A project plan is created that records the work to be done, who will do it, the development schedule and the work products.
- Managers use the plan to support project decision making and as a way of measuring progress.

# Plan-driven development – pros and cons

---

- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
- The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

# Project Plan Structure

---

- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.
- Plan sections
  - Introduction
  - Project organization
  - Risk analysis
  - Hardware and software resource requirements
  - Work breakdown
  - Project schedule
  - Monitoring and reporting mechanisms

# Project Plan includes...

---

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

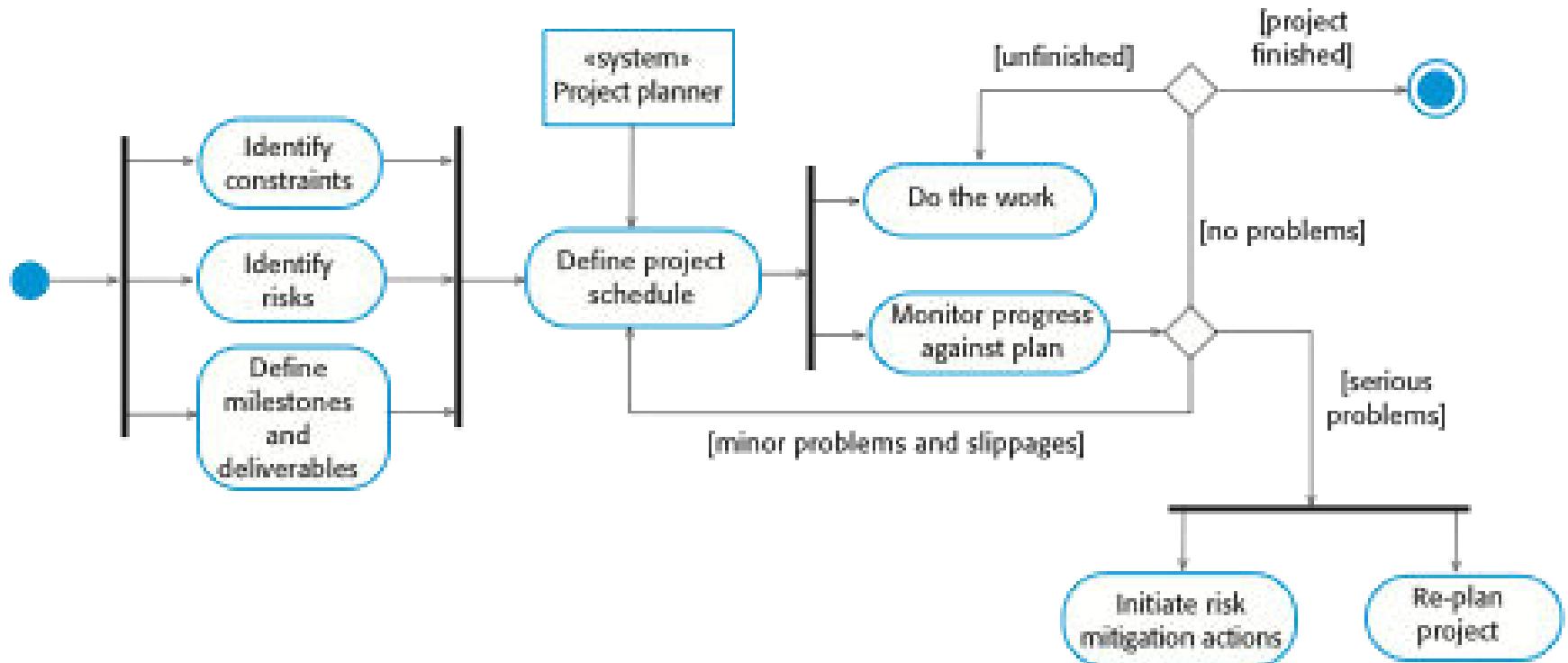
---

# The Planning Process

---

- Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.
- Plan changes are inevitable.
  - As more information about the system and the project team becomes available during the project, you should regularly revise the plan to reflect requirements, schedule and risk changes.
  - Changing business goals also leads to changes in project plans. As business goals change, this could affect all projects, which may then have to be re-planned.

# The Project Planning / Execution Process



---

# Agile Planning

Extreme Programming (XP) / Pair Programming / Scrum

# Agile Planning

---

- Agile methods of software development are **iterative** approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, the functionality of these increments is **not planned in advance** but is decided during the development.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a **flexible plan** that can accommodate these changes.

# Agile Planning Stages

---

- **Release planning**, which looks ahead for several months and decides on the features that should be included in a release of a system.
- **Iteration planning**, which has a shorter term outlook, and focuses on planning the next increment of a system. This is typically 2-4 weeks of work for the team.

# Planning in XP



# Story-based Planning

---

- The system specification in XP is based on user stories that reflect the features that should be included in the system.
- The project team read and discuss the stories and rank them in order of the amount of time they think it will take to implement the story.
- Release planning involves selecting and refining the stories that will reflect the features to be implemented in a release of a system and the order in which the stories should be implemented.
- Stories to be implemented in each iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks).

# Software Project Planning: Summary

---

- The price charged for a system does not just depend on its estimated development costs; it may be adjusted depending on the market and organizational priorities.
- Plan-driven development is organized around a complete project plan that defines the project activities, the planned effort, the activity schedule and who is responsible for each activity.
- Project scheduling involves the creation of graphical representations the project plan. Bar charts show the activity duration and staffing timelines, are the most commonly used schedule representations.
- The XP planning game involves the whole team in project planning. The plan is developed incrementally and, if problems arise, is adjusted. Software functionality is reduced instead of delaying delivery of an increment.

---

# Thank You



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

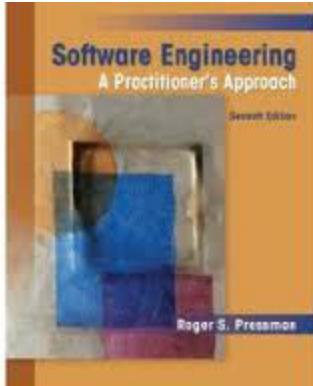
# SS ZG622: Software Project Management (Project Scheduling)

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad



# Project Scheduling

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations



---

# Project Scheduling

# Project Scheduling

---

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and **when and how** these tasks will be executed.
- You estimate the **calendar time** needed to complete each task, the **effort** required and who will work on the tasks that have been identified.
- You also have to estimate the **resources** needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

# Project Scheduling Activities

---

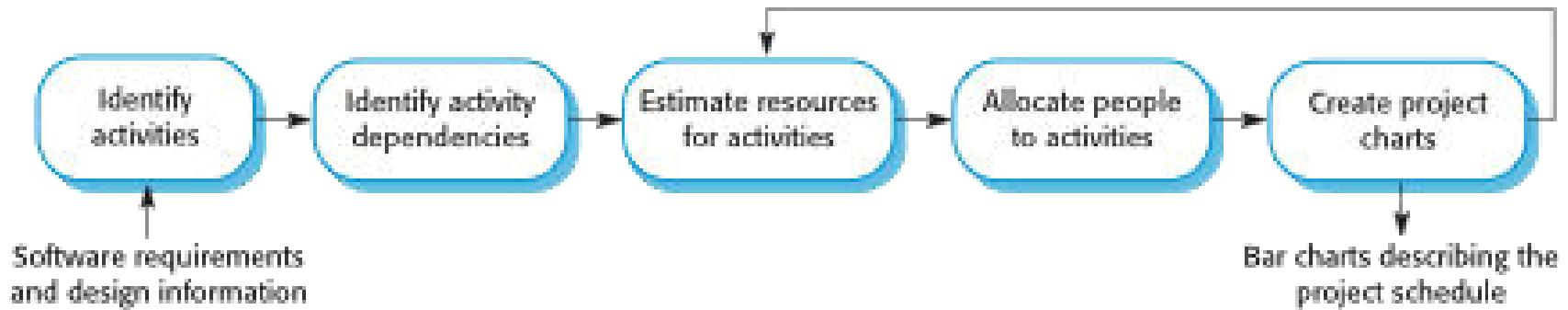
- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks **concurrently** to make **optimal** use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

# Milestones and Deliverables

---

- **Milestones** are points in the schedule against which you can assess progress, for example, the handover of the system for testing.
- **Deliverables** are work products that are delivered to the customer, e.g. a requirements document for the system.

# The Project Scheduling Process



# Scheduling problems

---

- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

# Schedule representation

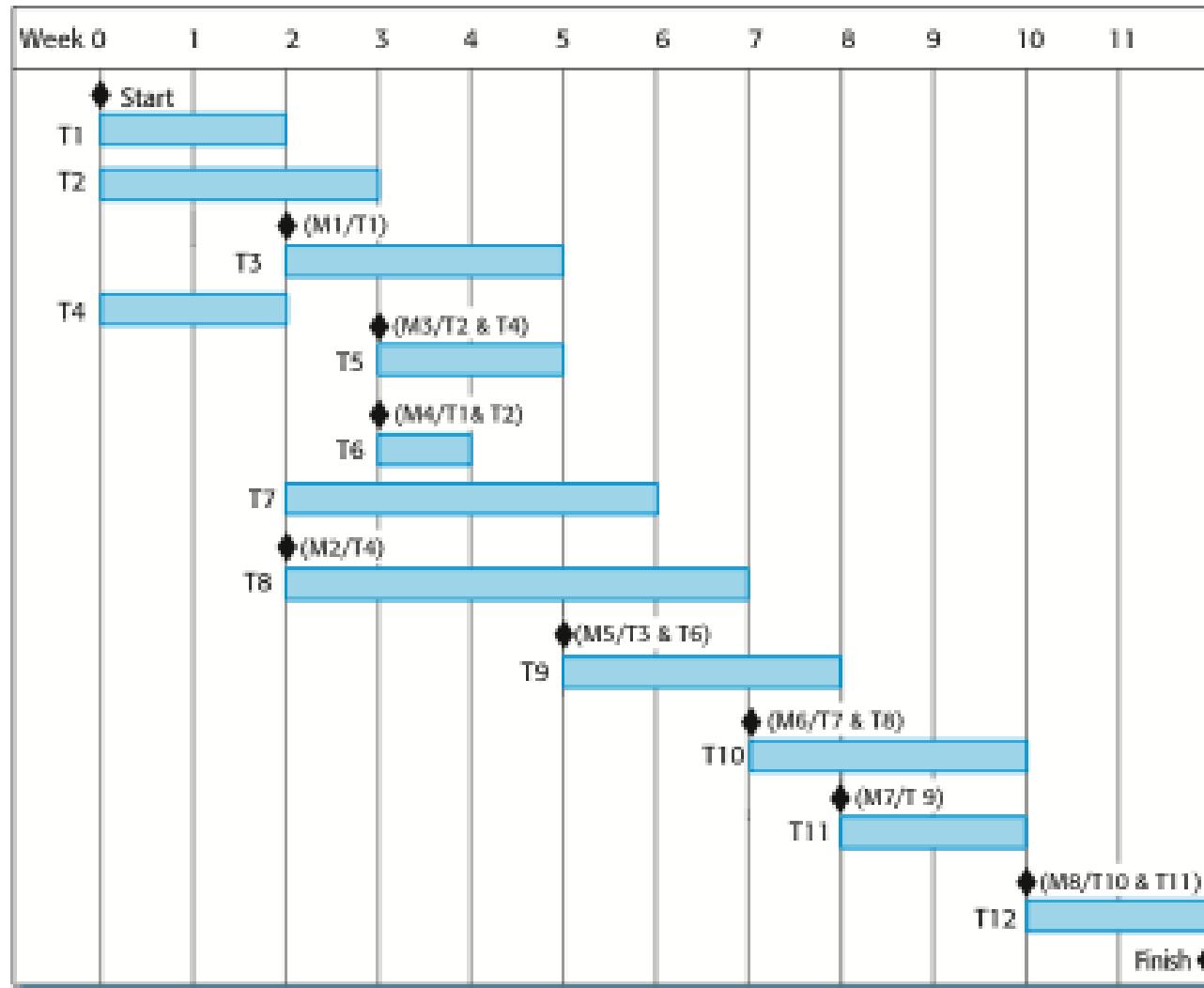
---

- Graphical notations are normally used to illustrate the project schedule.
- These show the project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- **Bar charts** are the most commonly used representation for project schedules. They show the schedule as activities or resources against time.

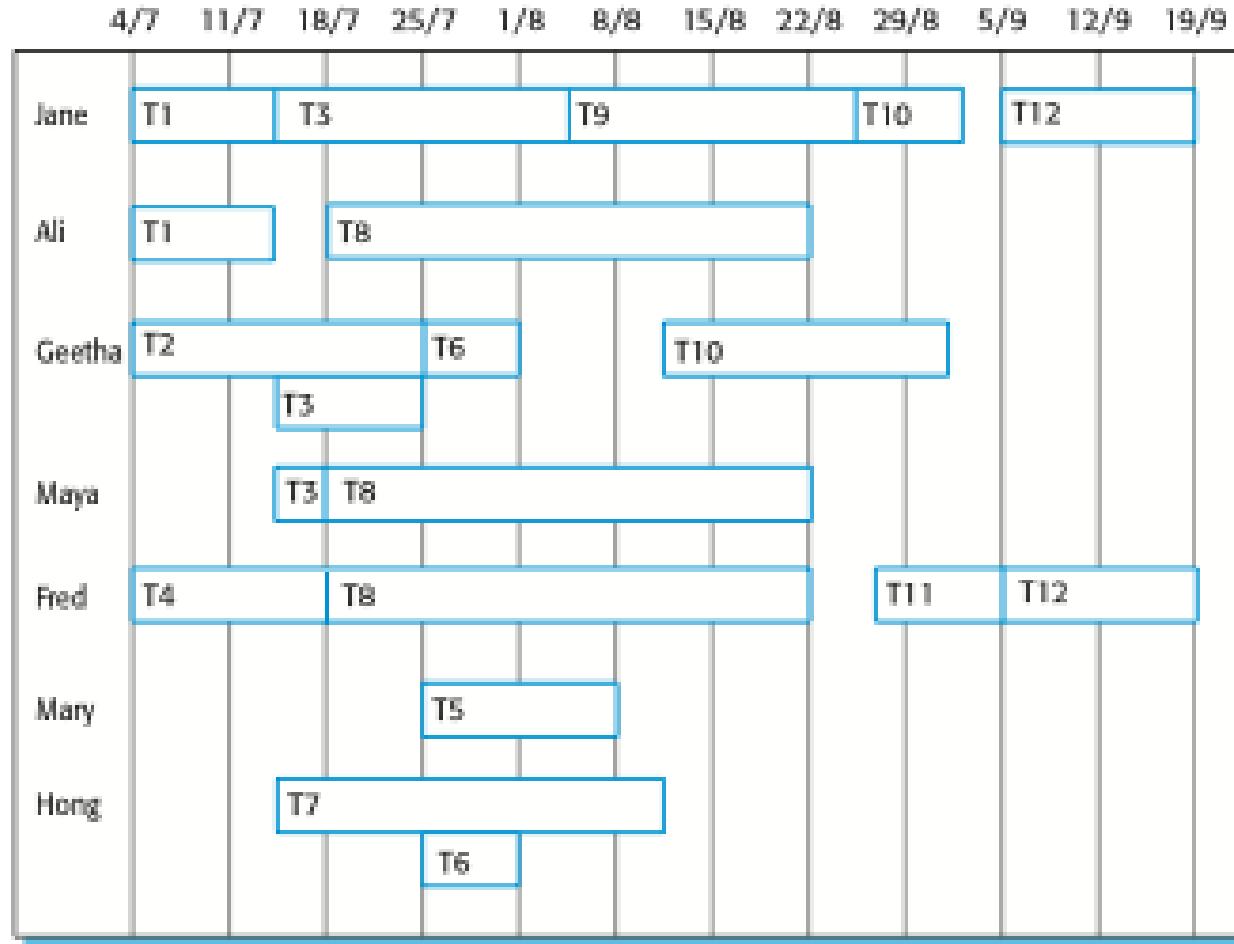
# Tasks, durations, and dependencies

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

# Activity bar chart



# Staff (Resource) allocation chart



---

# Thank You

## Any Questions?



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# SS ZG622:

## Software Project Management

(Project Scheduling using Activity Network Diagram)

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad



# **Project Scheduling using Activity-on-Network (AoN) Diagram**

# Scheduling

---

‘Time is nature’s way of stopping everything happening at once’

Having

- worked out a method of doing the project
- identified the tasks to be carried
- assessed the time needed to do each task

need to allocate dates/times for the start and end of each activity

# Activity networks

---

- Assess the feasibility of the planned project completion date
- Identify when resources will need to be deployed to activities
- Calculate when costs will be incurred
- This helps the co-ordination and motivation of the project team

# Defining activities

---

Activity networks are based on some assumptions:

- A project is:
  - Composed of a number of **activities**
  - May start when at least one of its activities is ready to start
  - Completed when all its activities are completed

# Defining activities -continued

---

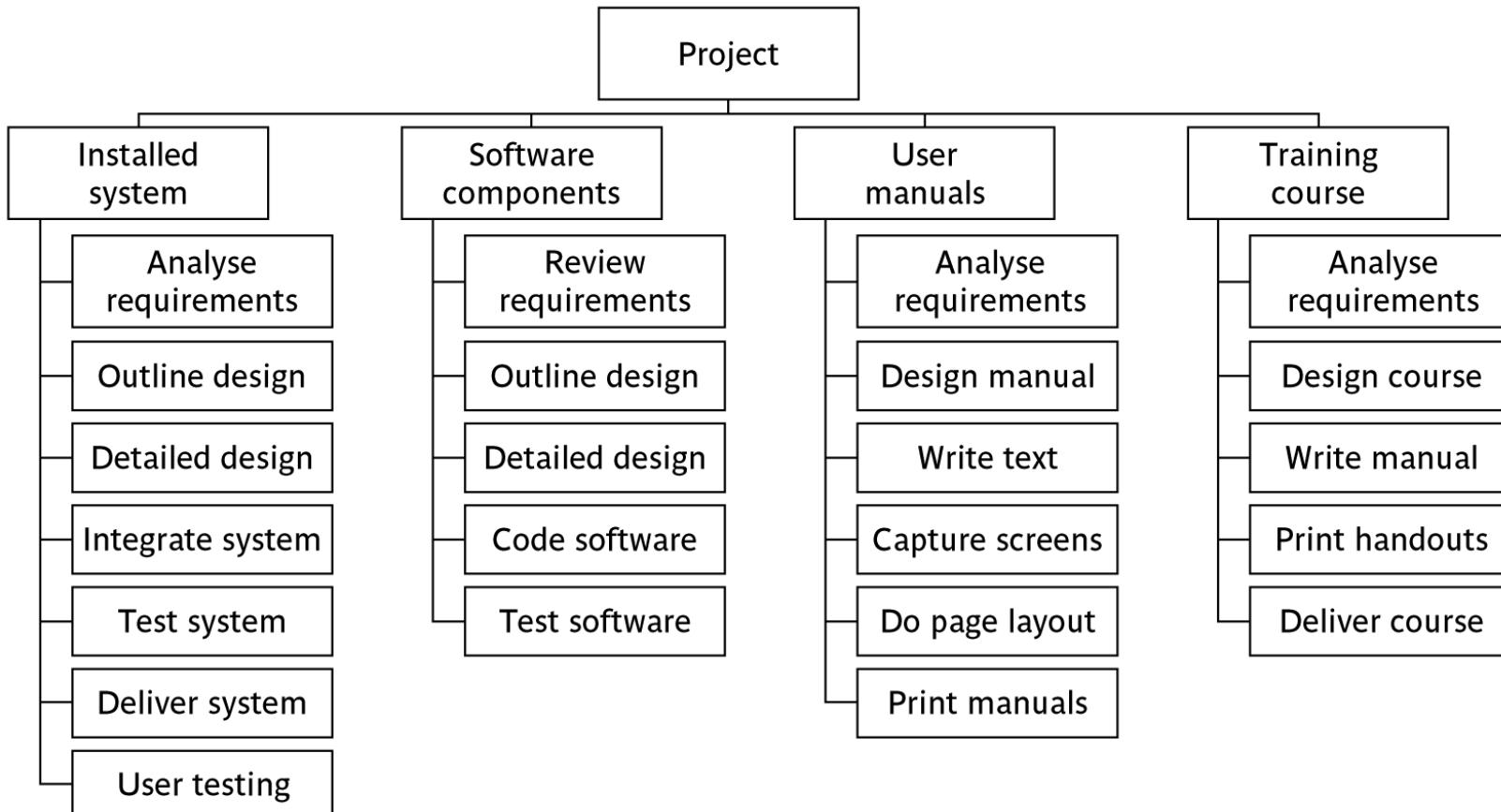
- An activity
  - Must have clearly defined start and end-points
  - Must have resource requirements that can be forecast: these are assumed to be constant throughout the project
  - Must have a duration that can be forecast
  - May be dependent on other activities being completed first (precedence networks)

# Identifying activities

---

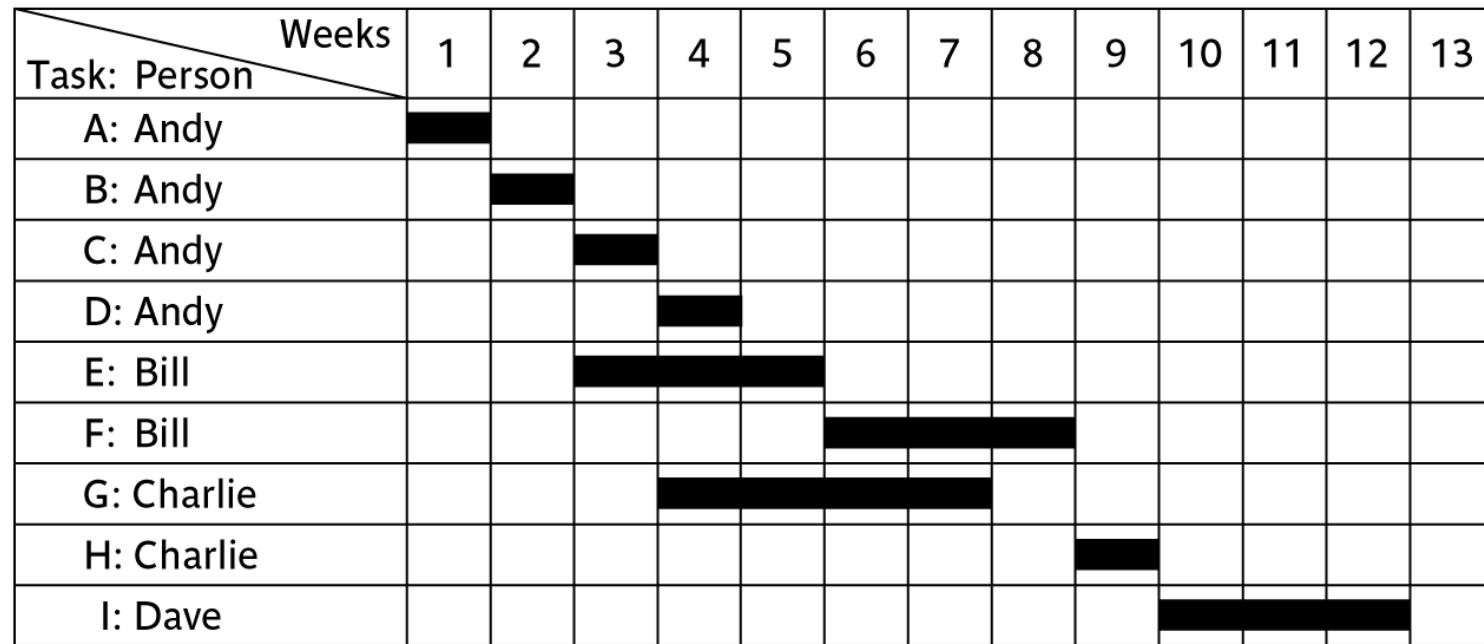
- Work-based: draw-up a Work Breakdown Structure listing the work items needed
- Product-based approach
  - list the deliverable and intermediate products of project – product breakdown structure (PBS)
  - Identify the order in which products have to be created
  - work out the activities needed to create the products

# Hybrid approach



# The final outcome of the planning process

A project plan as a bar chart

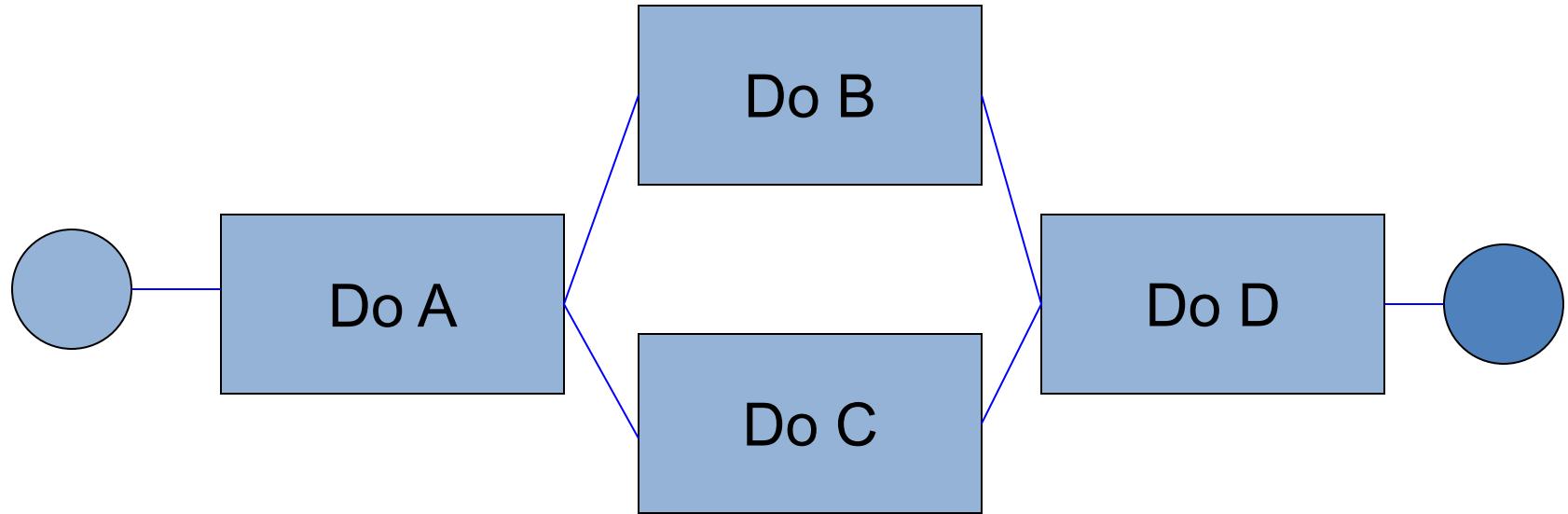


Activity key

A: Overall design  
 B: Specify module 1  
 C: Specify module 2  
 D: Specify module 3  
 E: Code module 1

F : Code module 3  
 G: Code module 2  
 H: Integration testing  
 I : System tesing

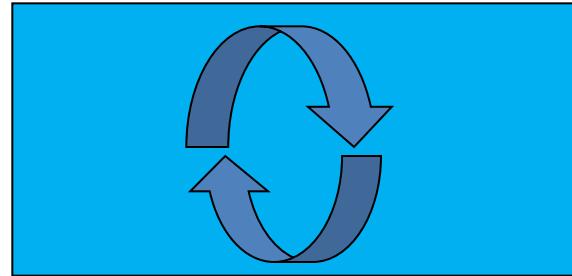
# Activity-on-Node (AoN) Diagram



# Drawing up a AoN diagram

---

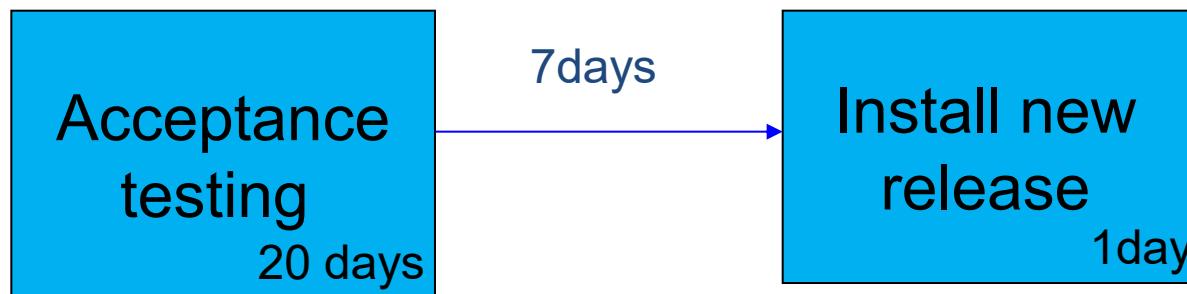
- No looping back is allowed – deal with iterations by hiding them within single activities



- *milestones* – ‘activities’, such as the start and end of the project, which indicate transition points. They have zero duration.

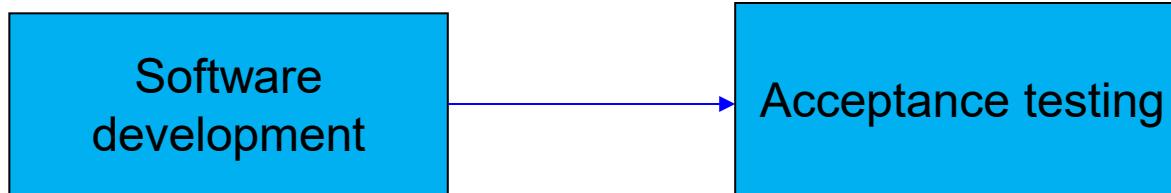
# Lagged activities

- where there is a fixed delay between activities e.g. seven days notice has to be given to users that a new release has been signed off and is to be installed

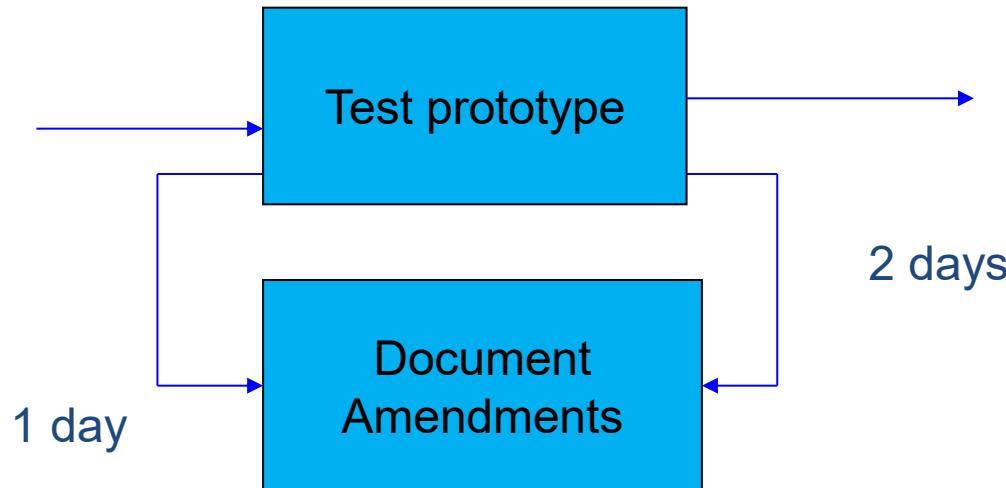


# Types of links between activities

- Finish to start

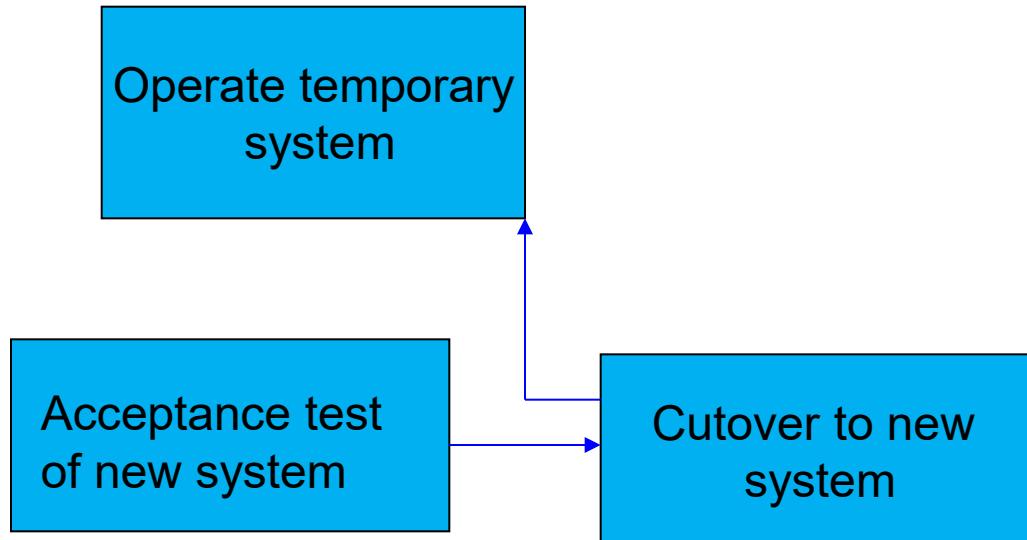


- Start to start/ Finish to finish

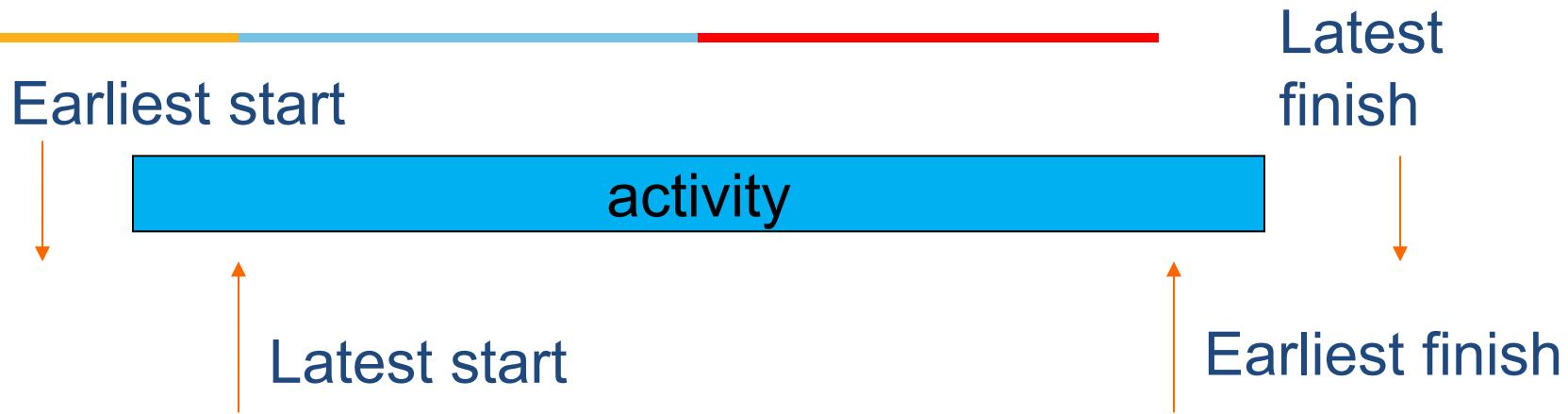


# Types of links between activities

- Start to finish



# Start and finish times



- Activity 'write report software'
- Earliest start (ES)
- Earliest finish (EF) = ES + duration
- Latest finish (LF) = latest task can be completed without affecting project end Latest start = LF - duration

# Example

---

- earliest start = day 5
- latest finish = day 30
- duration = 10 days
- earliest finish = ?
- latest start = ?

Float = LF - ES – duration = ?

# AoN Notation: 'Day 0'

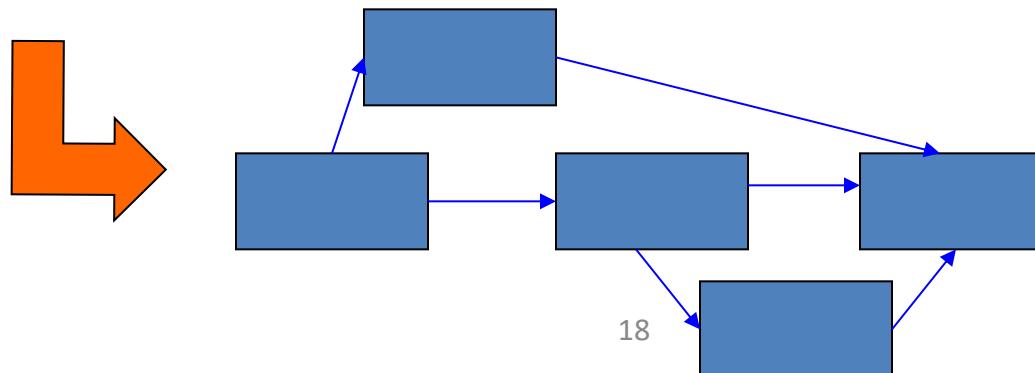
---



- Day numbers used rather than actual dates
- Makes initial calculations easier – not concerned with week-ends and public holidays
- For **finish** date/times Day 1 means at the END of Day 1.
- For a **start** date/time Day 1 also means at the END of Day 1.
- The first activity therefore begin at Day 0 i.e. the end of Day 0 i.e. the start of Day 1

# AoN: notation

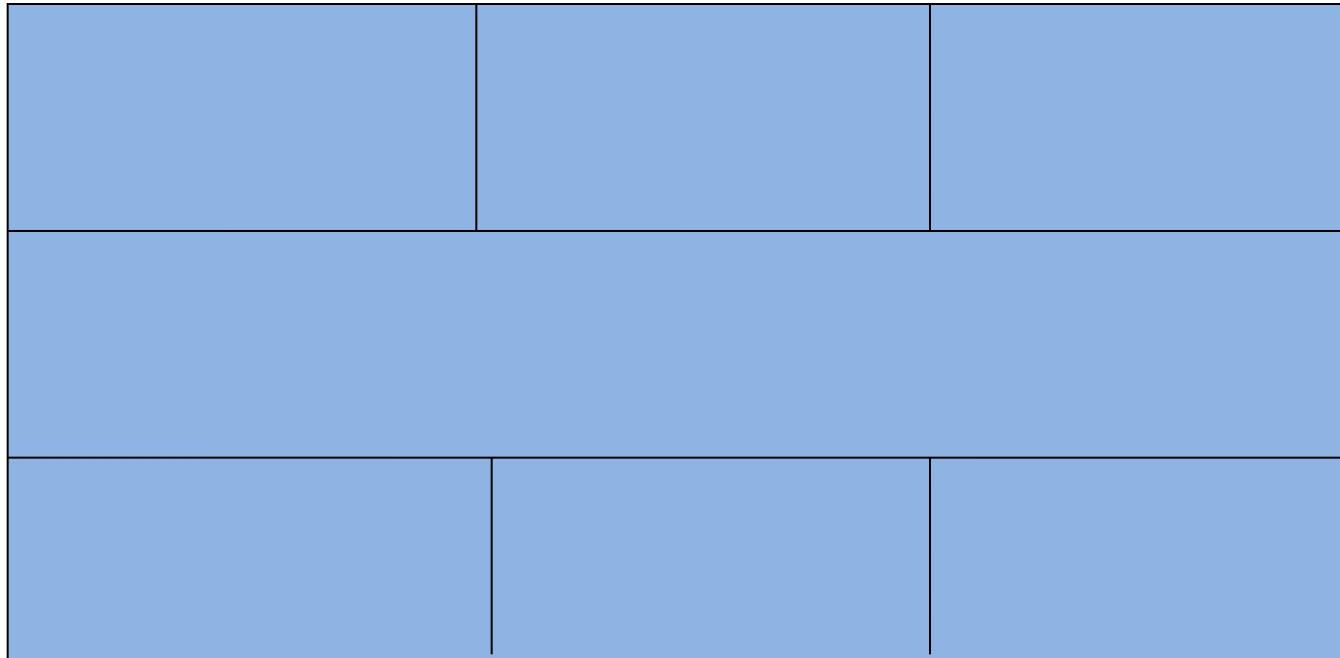
Earliest start	Duration	Earliest finish
Activity label, activity description		
Latest start	Float	Latest finish



# Complete for the previous example

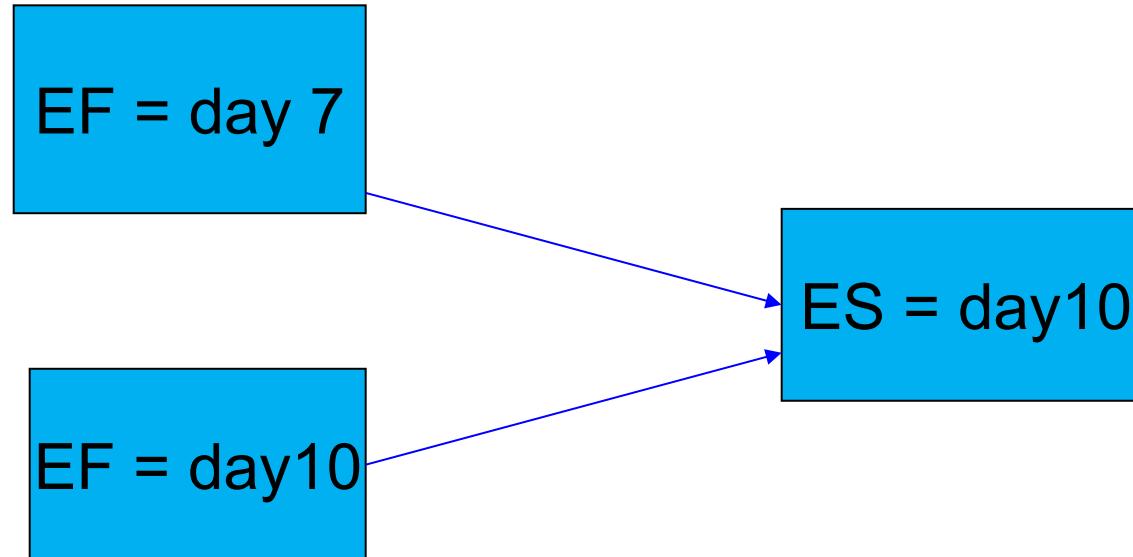
---

- earliest start = day 5 / earliest finish = ?
- latest finish = day 30 / latest start = ?
- duration = 10 days

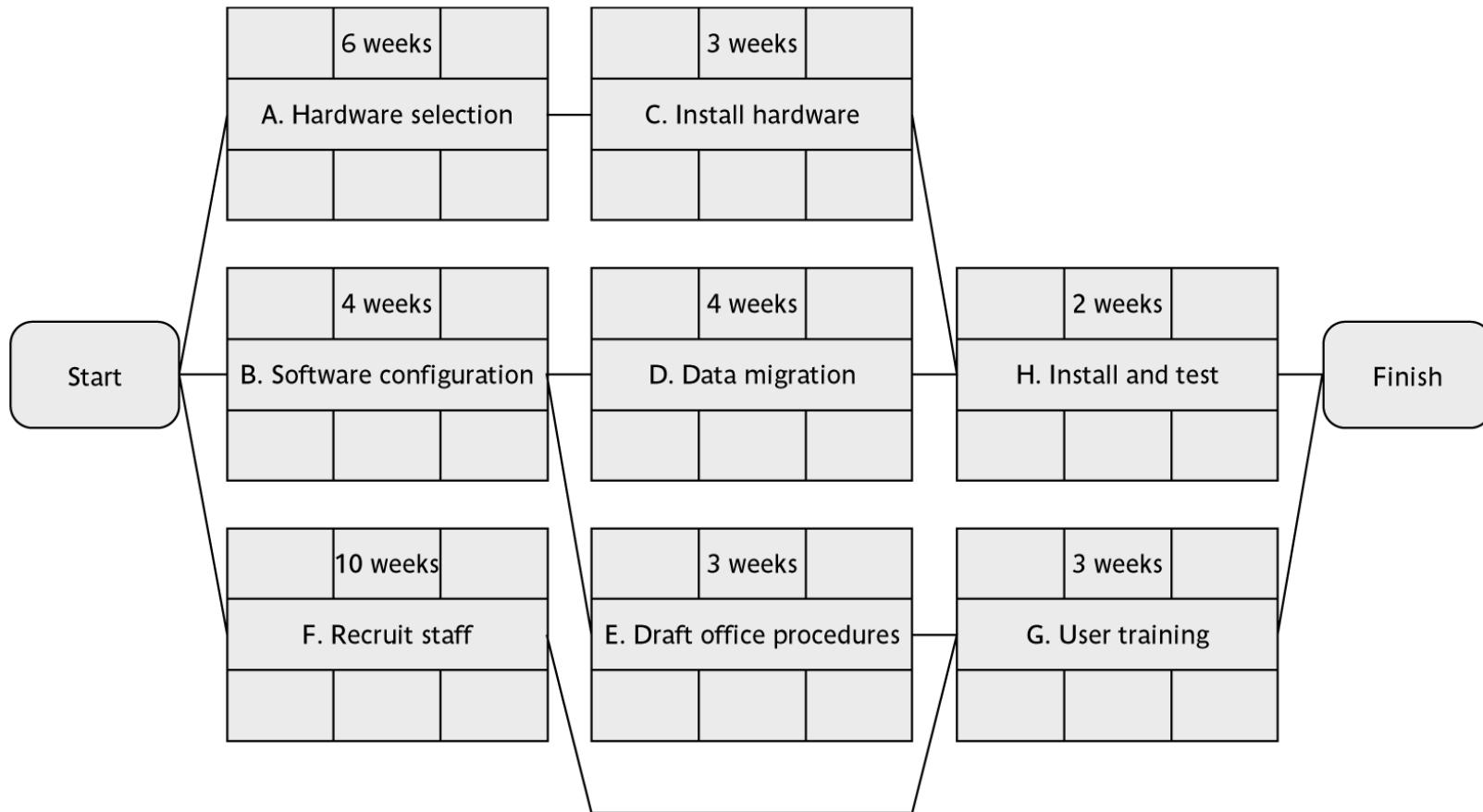


# Forward pass

- Start at beginning (Day 0) and work forward following chains.
- Earliest start date for the *current* activity = earliest finish date for the *previous*
- When there is more than one previous activity, take the *latest* earliest finish



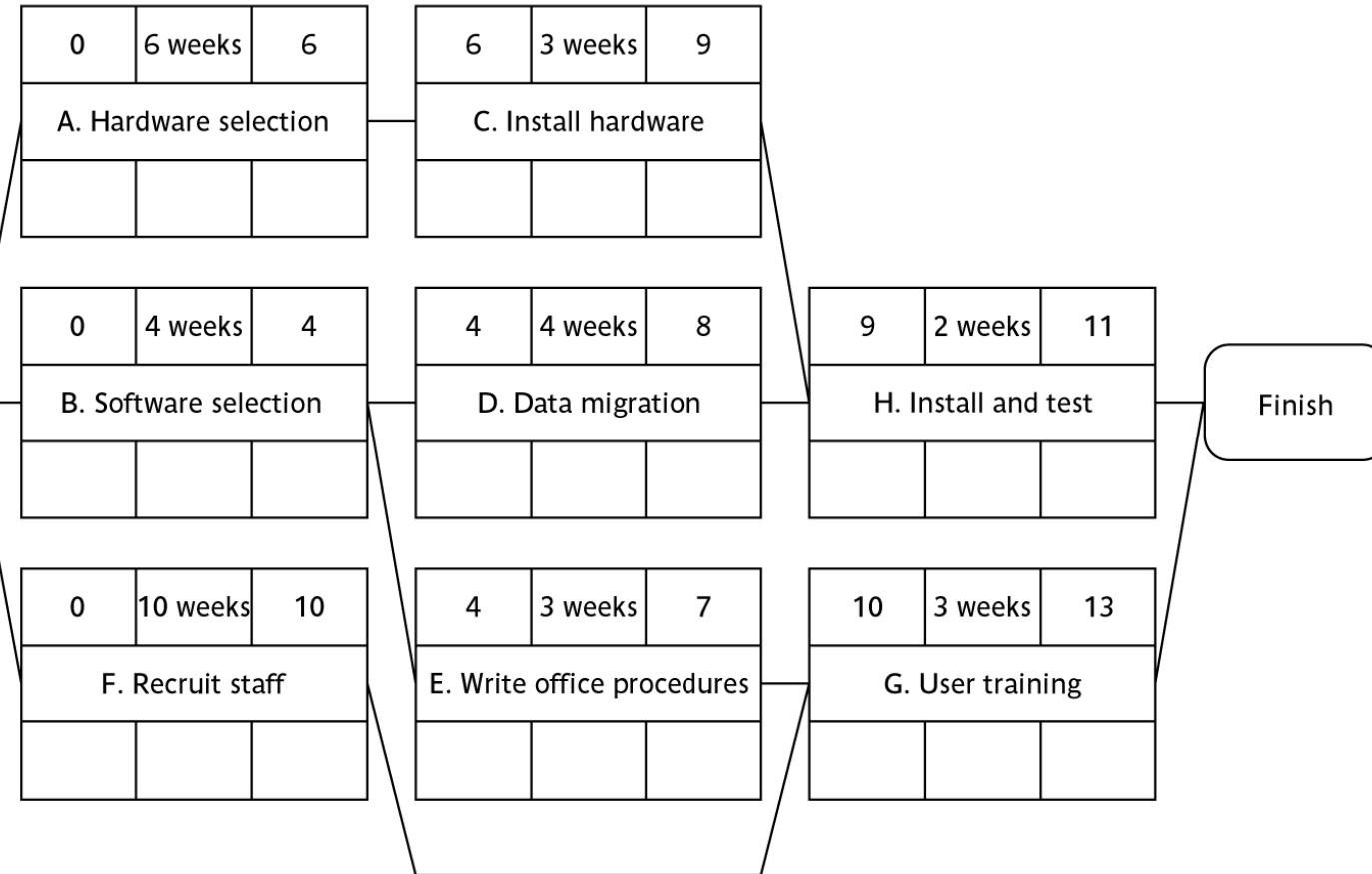
# Example of an activity network



# Forward Pass →

Activity	ES	Dur	Pre	EF	LS	LF
A (HW selection)	0	6	-	6		
B (System Config)	0	4	-	4		
C (Instal HW)	6	3	A	9		
D (Data Migration)	4	4	B	8		
E (Draft office process)	4	3	B	7		
F (Recruit Staff)	0	10	-	10		
G (User Training)	10	3	E,F	14		
H (Instal and Test System)	9	2	C,D	11		

# Example: LS for all activities



# Backward pass

- Start from the *last* activity
- Latest finish (LF) for last activity = earliest finish (EF)
- work backwards
- Latest finish for *current* activity = Latest start for the *following*
- More than one following activity - take the *earliest* LS
- Latest start (LS) = LF for activity - duration

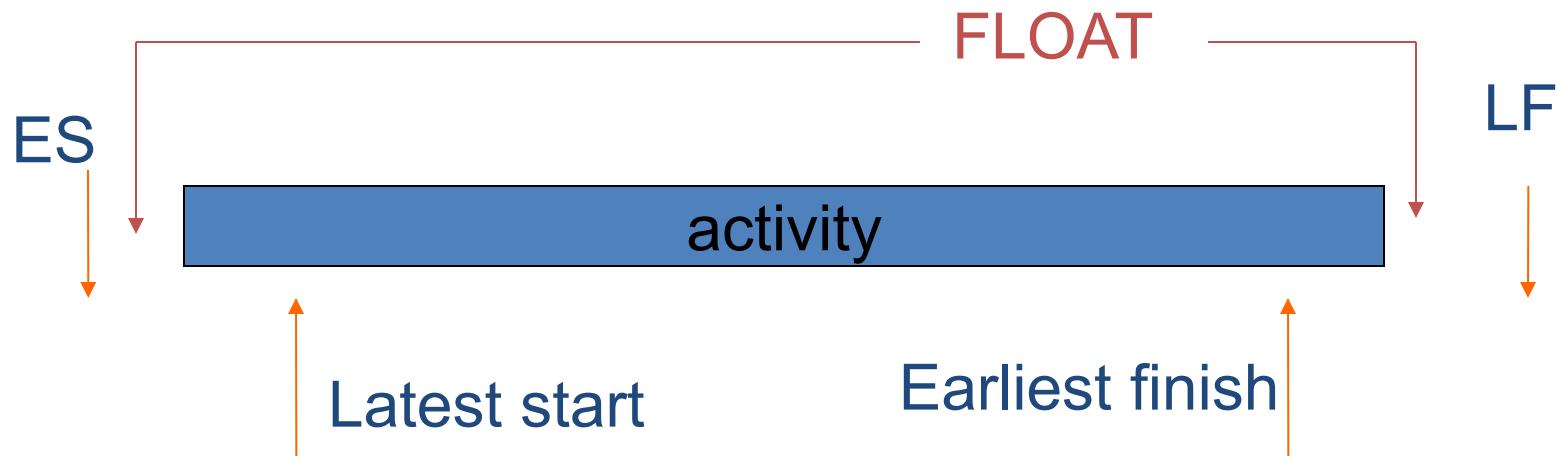
# Let's Complete the table

Activity	ES	Dur	Pre	EF	LS	LF
A (HW selection)	0	6	-	6	2	8
B (System Config)	0	4	-	4	3	7
C (Instal HW)	6	3	A	9	8	11
D (Data Migration)	4	4	B	8	7	11
E (Draft office process)	4	3	B	7	7	10
F (Recruit Staff)	0	10	-	10	0	10
G (User Training)	10	3	E,F	14	10	13
H (Instal and Test System)	9	2	C,D	11	11	13

# Float



Float = Latest finish - Earliest start - Duration

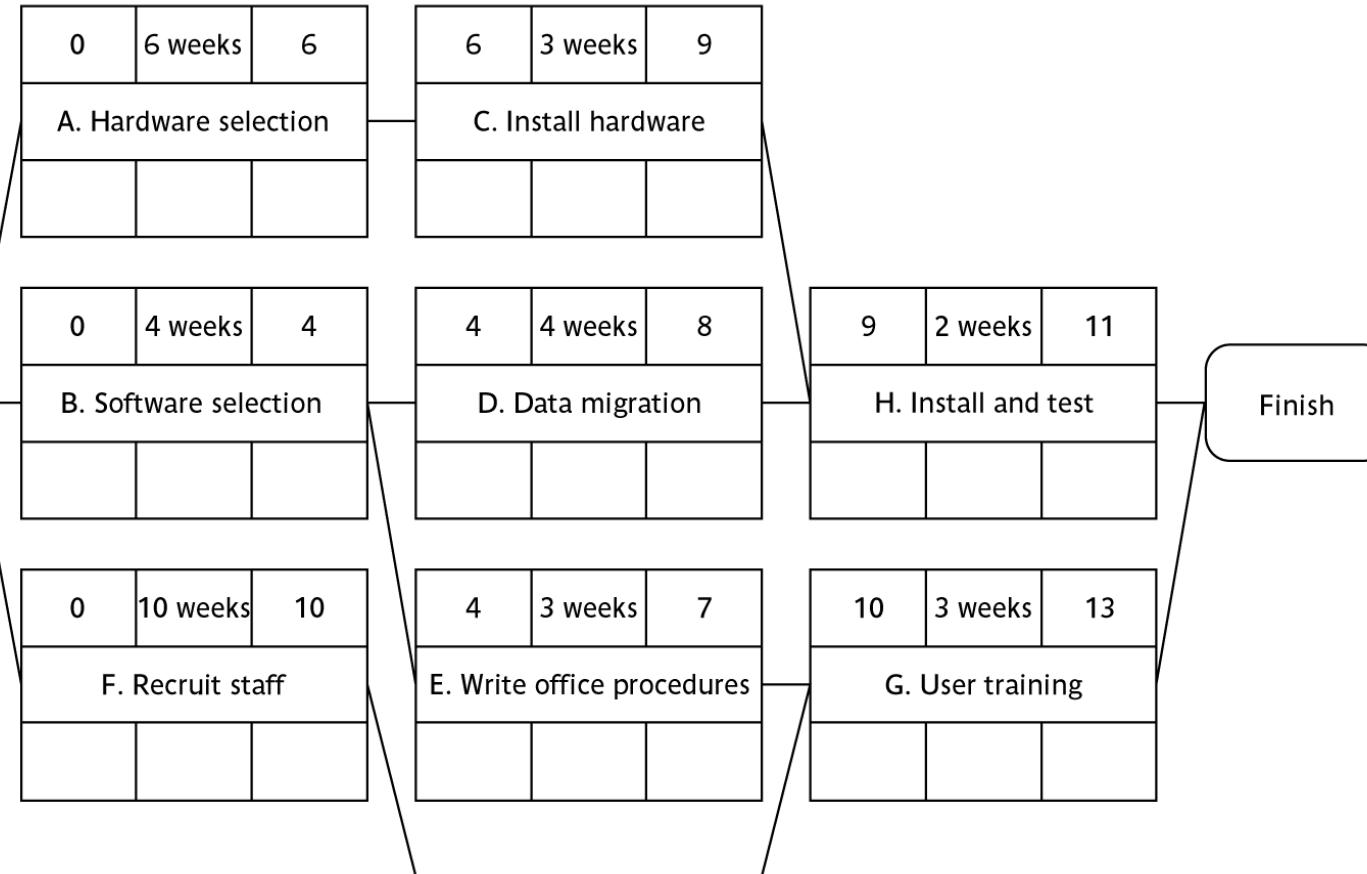


# Compute Activity Float



Activity	ES	Dur	Pre	EF	LS	LF	Float
A (HW selection)	0	6	-	6	2	8	2
B (System Config)	0	4	-	4	3	7	3
C (Instal HW)	6	3	A	9	8	11	2
D (Data Migration)	4	4	B	8	7	11	3
E (Draft office process)	4	3	B	7	7	10	3
F (Recruit Staff)	0	10	-	10	0	10	0
G (User Training)	10	3	E,F	14	10	13	0
H (Instal and Test System)	9	2	C,D	11	11	13	2

# Example: LS/LF for all activities



# Critical path

---

- Note the path through network with **zero** floats
- **Critical path**: any delay in an activity on this path will delay whole project
- There will be **at least ONE** critical path

# ‘Free’ and ‘interfering’ float

0	7w	7
A		
2	2	9

0	4w	4
B		
5	5	9

0	10w	10
A		
0	0	10

B can be up to 3 days late  
and not affect any  
other activity = **free float**

7	1w	8
D		
9	2	10

10	2w	12
E		
10	0	12

B can be a further 2 days late – affects  
D but not the project end date =  
**interfering float**

# Project Scheduling: Summary

---

- Scheduling is arranging activities on a time-scale
- Activity dependencies are charted using AoN diagrams
- Critical Path (CP) is the path of activities (from start to finish) whose float is zero
- CP determines the duration of the project
- Managing activities on CP is the priority of Project Manager as delay of any activity on CP delays the whole project
- Shortening schedule of any project involves reducing duration of CP activities (which might result in new CP!)
- Commercial Tools (e.g., MS Project) have built-in diagramming and scheduling features

---

*Please do some simple exercises given in the text-book*

# Thank You



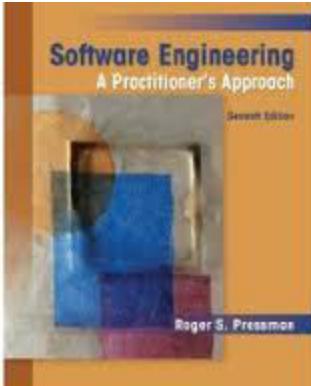
**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# SS ZG622: Software Project Management (Resource/Cost Scheduling)

Prof K G Krishna, BITS-Pilani Off-campus centre, Hyderabad

# Text Books



**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Resource Allocation/Loading Cost Scheduling**

**Source Courtesy:** Some of the contents of this PPT are sourced from materials provided by Publishers of T1 & T2

# Schedules *in general...*

---

- *Activity schedule* - indicating start and completion dates for each activity
- *Resource schedule* - indicating dates when resources needed + level of resources
- *Cost schedule* showing accumulative expenditure

# Resources

---

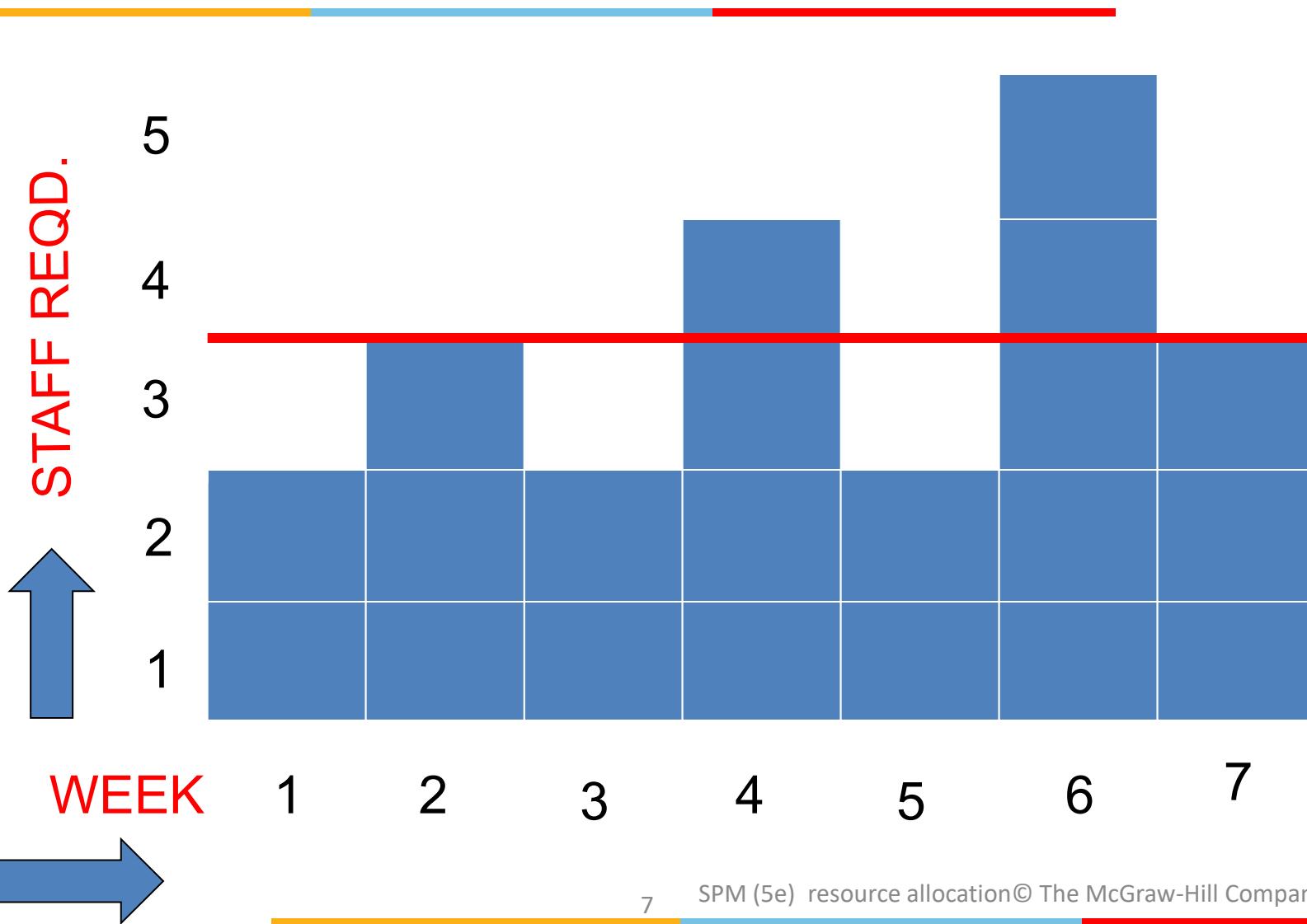
- These include
  - labour
  - equipment (e.g. workstations)
  - materials
  - space
  - services
- Time: elapsed time can often be reduced by adding more staff
- Money: used to buy the other resources

# Resource allocation

---

- Identify the resources needed for each activity and create a *resource requirement list*
- Identify *resource types* - individuals are interchangeable within the group (e.g. 'VB programmers' as opposed to 'software developers')
- Allocate resource types to activities and examine the *resource histogram*

# Resource histogram:

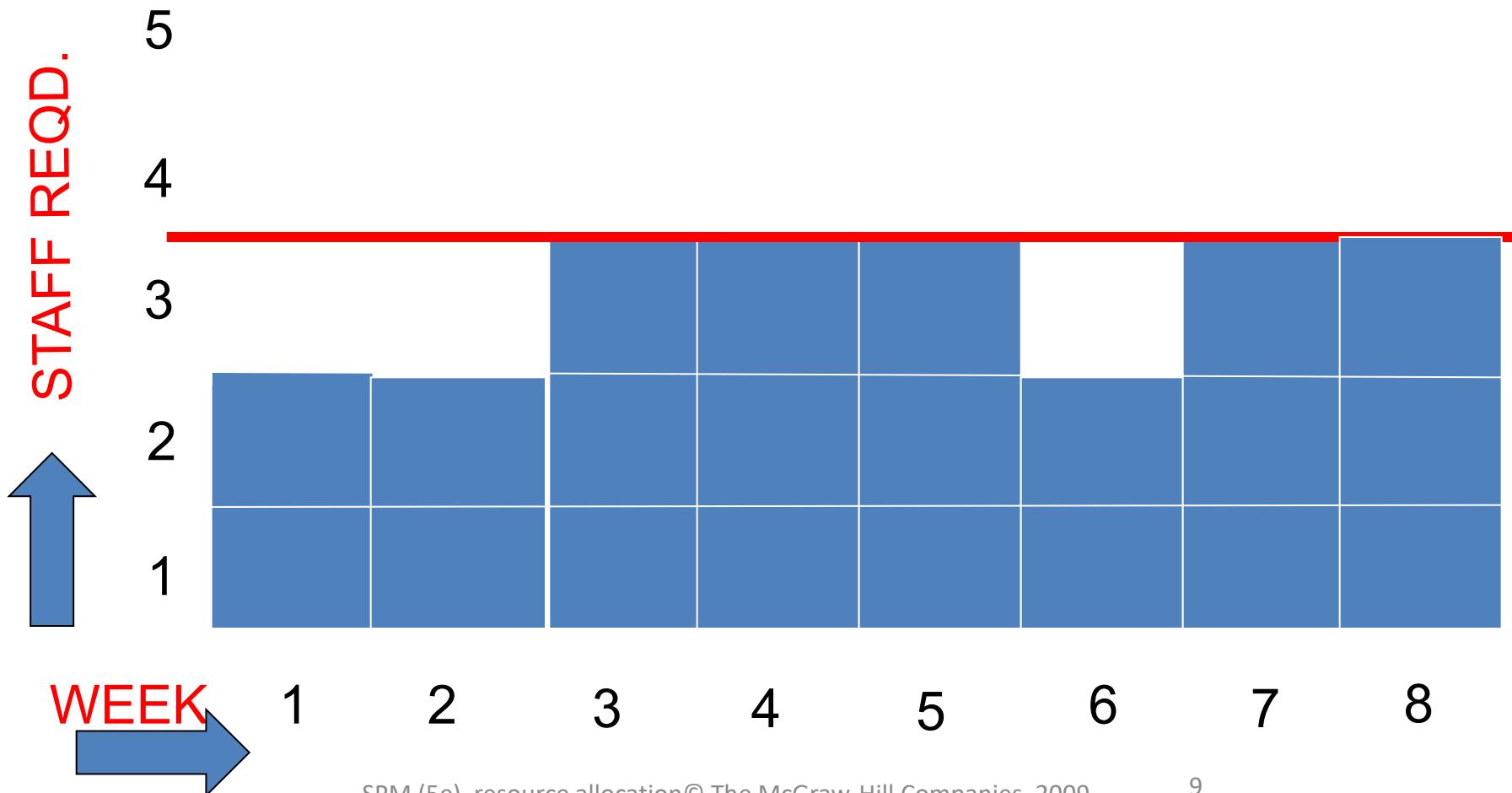


# Resource smoothing

---

- It is usually difficult to get specialist staff who will work odd days to fill in gaps – need for staff to learn about application etc.
- Staff often have to be employed for a continuous block of time
- Desirable to employ a constant number of staff on a project – who as far as possible are fully employed

# Resource smoothing



# Resource clashes

---

- Where same resource needed in more than one place at the same time
- can be resolved by:
  - delaying one of the activities
    - taking advantage of float to change start date
    - delaying start of one activity until finish of the other activity that resource is being used on - *puts back project completion*
  - moving resource from a non-critical activity
  - bringing in additional resource - *increases cost*

# Prioritizing activities

---

There are two main ways of doing this:

- *Total float priority* – those with the smallest float have the highest priority
- *Ordered list priority* – this takes account of the duration of the activity as well as the float  
*(contd..)*

# Burman's priority list

---

Give priority to:

- Shortest critical activities
- Other critical activities
- Shortest non-critical activities
- Non-critical activities with least float
- Non-critical activities

# Resource usage

---

- need to maximise %usage of resources i.e. reduce idle periods between tasks
- need to balance costs against early completion date
- need to allow for contingency

# Affecting Critical path?

---

- Scheduling resources can create new dependencies between activities
- It is best not to add dependencies to the activity network to reflect resource constraints
  - Makes network very messy
  - A resource constraint may disappear during the project, but link remains on network
- Amend dates on **schedule** to reflect resource constraints

# Allocating individuals to activities

---

The initial ‘resource types’ for a task have to be replaced by actual individuals.

Factors to be considered:

- Availability
- Criticality
- Risk
- Training
- Team building – and motivation

# Cost schedules

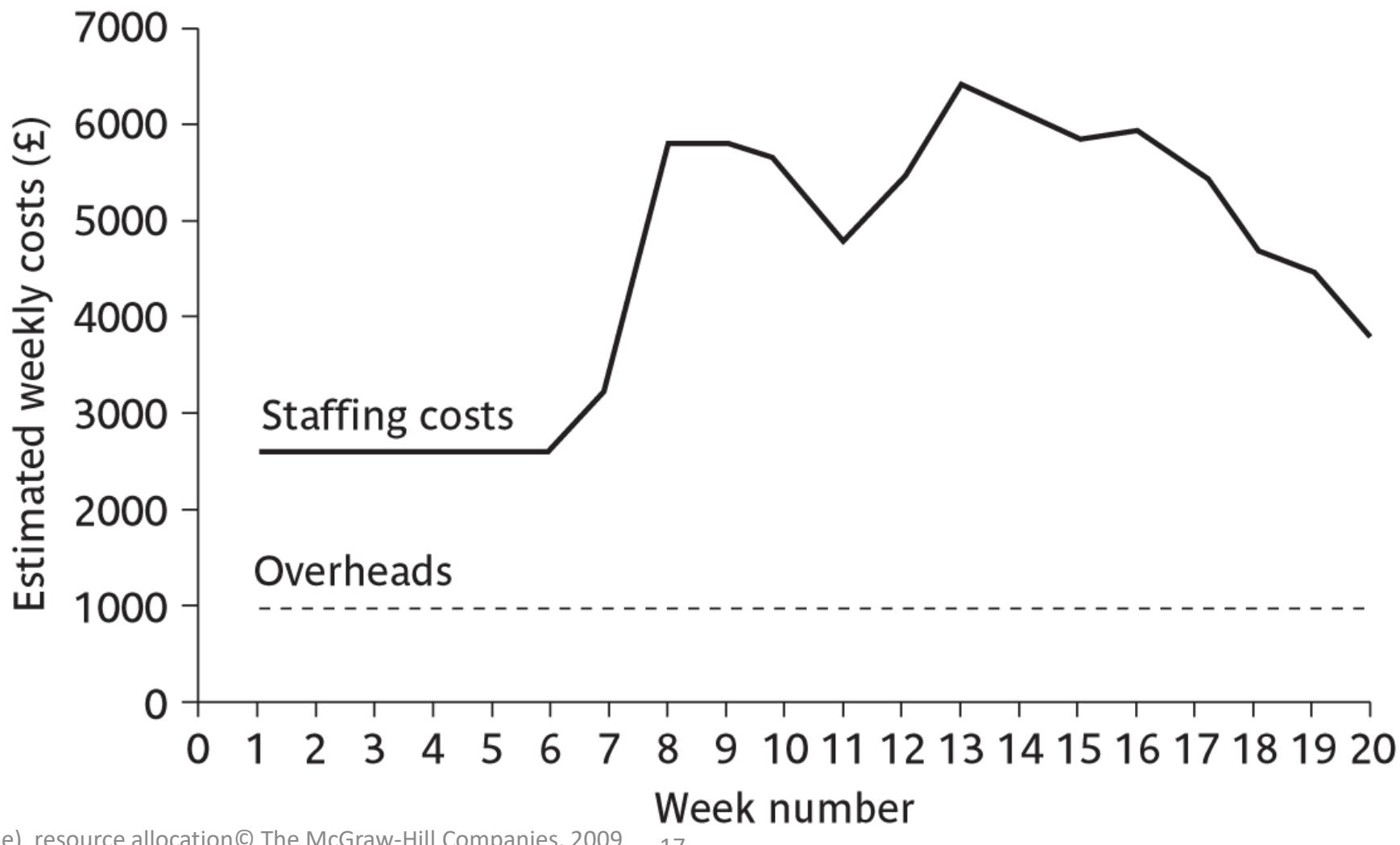
---

Cost schedules can now be produced:

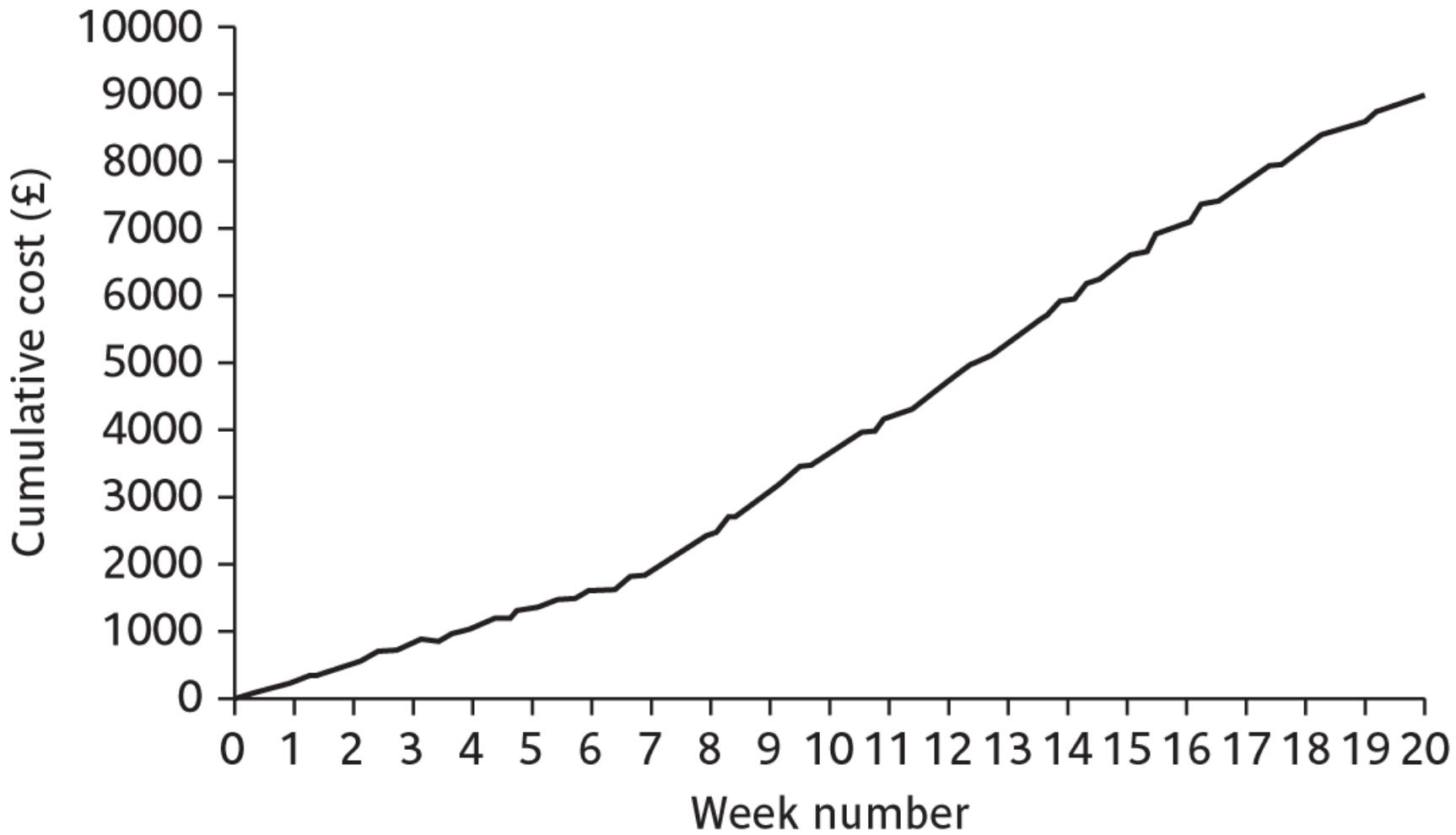
Costs include:

- Staff costs
- Overheads
- Usage charges

# Cost profile

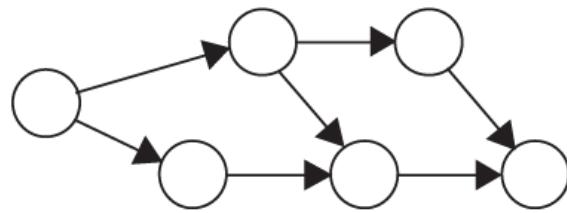


# Accumulative costs

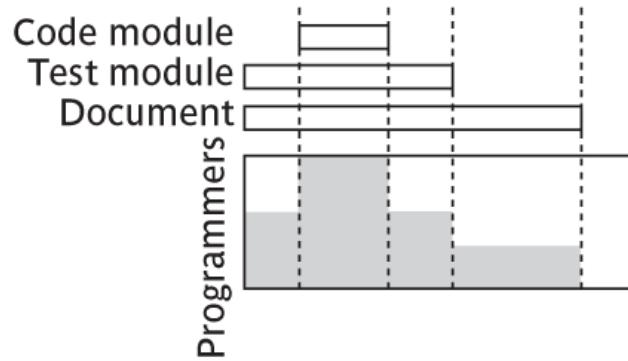


# Balancing concerns

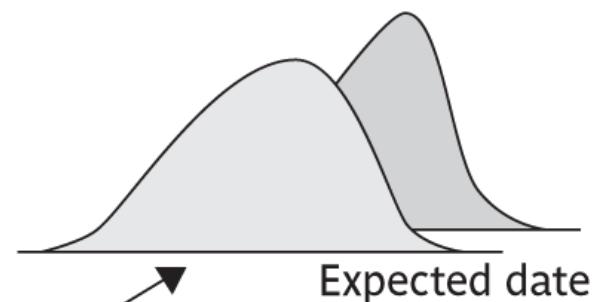
## Activity plan



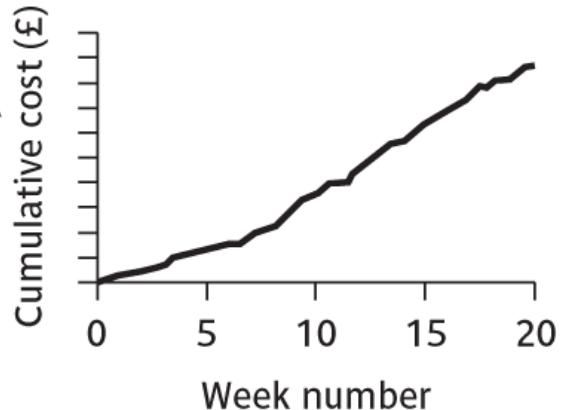
## Resource allocation



## Risk assessment



## Cost schedule



---

*Please do some simple exercises given in the text-book*

# Thank You



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

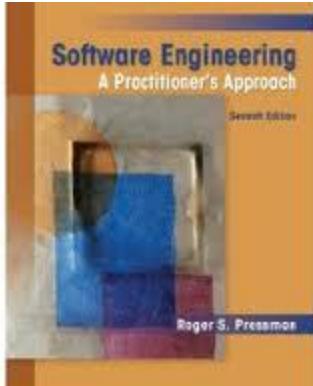
# SS ZG622:

## Software Project Management

### (Risk Analysis & Risk Management)

Prof K G Krishna, BITS-Pilani Off-Campus Centre, Hyderabad

# Text Books



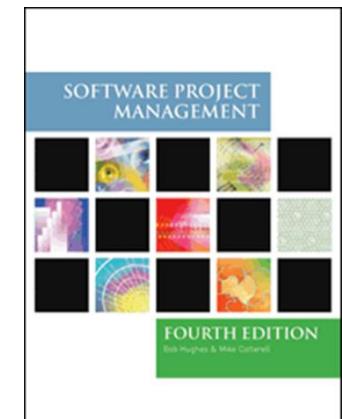
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Risk Analysis & Risk Management:**

## **Introduction, Concepts & Terminology**

# Topics

---

- Definition of 'risk' and 'risk management'
- Some ways of categorizing risk
- Risk management
  - Risk identification – what are the risks to a project?
  - Risk analysis – which ones are really serious?
  - Risk planning – what shall we do?
  - Risk monitoring – has the planning worked?

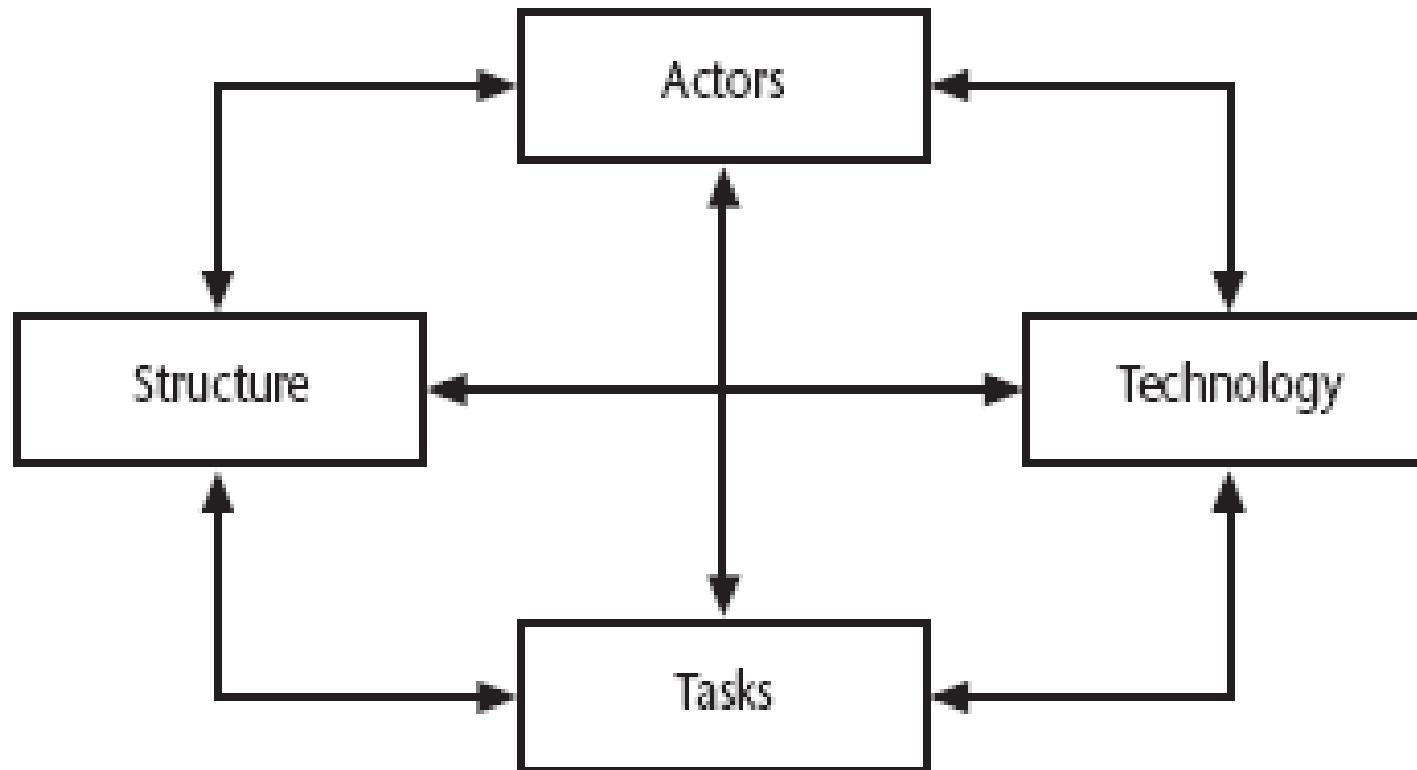
# Definitions of 'risk'

---

*'The chance of exposure to the adverse consequences of future events'*

- Project plans have to be based on *assumptions*
- *Risk* is the possibility that an assumption is wrong
- When the risk happens it becomes a *problem* or an *issue*

# Categories of risk



# Overview



- Risks are potential problems that might affect the successful completion of a software project.
- Risks involve uncertainty and potential losses.
- Risk analysis and management are intended to help a software team understand and manage uncertainty during the development process.
- The important thing is to remember that things can go wrong and to make plans to minimize their impact when they do.
- The work product is called a Risk Mitigation, Monitoring, and Management Plan (RMMM).

# Risk Component & Drivers

---

- *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- *Cost risk*—the degree of uncertainty that the project budget will be maintained.
- *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of four impact categories—**negligible, marginal, critical, or catastrophic**

# Risk Management

## Reactive

- project team reacts to risks when they occur
- mitigation—plan for additional resources in anticipation of fire fighting
- fix on failure—resource are found and applied when the risk strikes
- crisis management—failure does not respond to applied resources and project is in jeopardy

## Proactive

- formal risk analysis is performed
- organization corrects the root causes of risk
  - TQM concepts and statistical SQA
  - examining risk sources that lie beyond the bounds of the software
  - developing the skill to manage change

# Risk Check List

---

- *Product size (PS)*—risks associated with the overall size of the software to be built or modified.
- *Business impact (BU)*—risks associated with constraints imposed by management or the marketplace.
- *Customer characteristics (CU)*—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
- *Process definition (PR)*—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- *Development environment (DE)*—risks associated with the availability and quality of the tools to be used to build the product.
- *Technology to be built (TE)*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- *Staff size and experience (ST)*—risks associated with the overall technical and project experience of the software engineers who will do the work.

# Risk Projection & Building a Risk Table

---

- *Risk projection*, also called *risk estimation*, attempts to rate each risk in two ways
  - the likelihood or probability that the risk is real and
  - the consequences of the problems associated with the risk, should it occur.
- The project planner, along with other managers and technical staff, performs four risk projection activities:
  1. establish a scale that reflects the perceived likelihood of a risk,
  2. delineate the consequences of the risk,
  3. estimate the impact of the risk on the project and the product, and
  4. note the overall accuracy of the risk projection so that there will be no misunderstandings.

# Building Risk Table

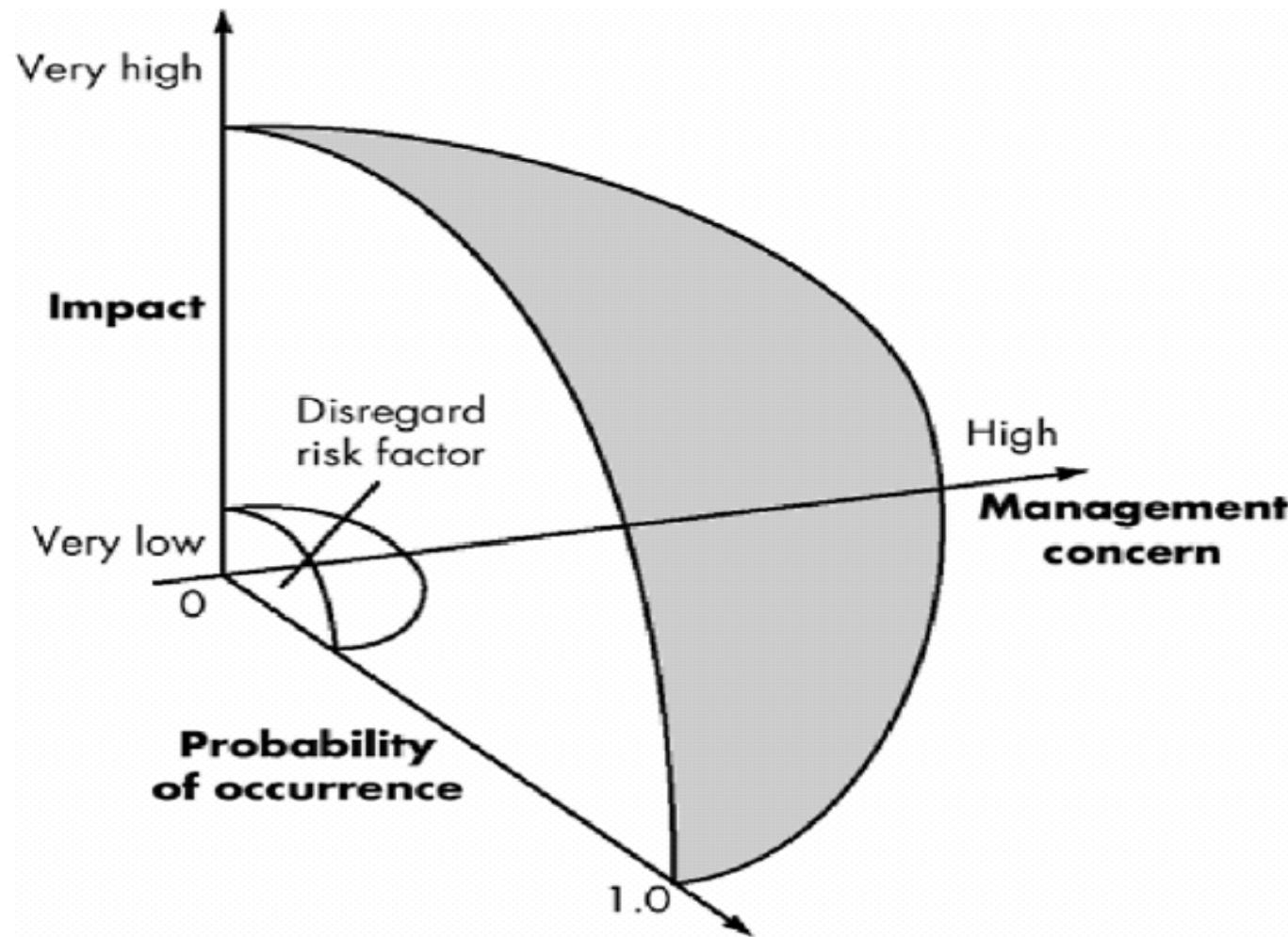
Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
•				
•				
•				

Impact values:

- 1—catastrophic
- 2—critical
- 3—marginal
- 4—negligible

**RMMM = Risk Mitigation, Monitoring and Management Plan**

# Risk and Management Concern



# Assessing Risk Impact

---

*The following steps are recommended to determine the overall consequences of a risk:*

1. Determine the average probability of occurrence value for each risk component.
2. Using table 1 (slide 10) determine the impact for each component based on the criteria shown.
3. Complete the risk table and analyze the results

The overall *risk exposure*, RE, is determined using the following relationship:  $RE = P \times C$

where  $P$  is the probability of occurrence for a risk, and  $C$  is the cost to the project should the risk occur.

# Example



*Assume that the software team defines a project risk in the following manner:*

**Risk identification.** Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

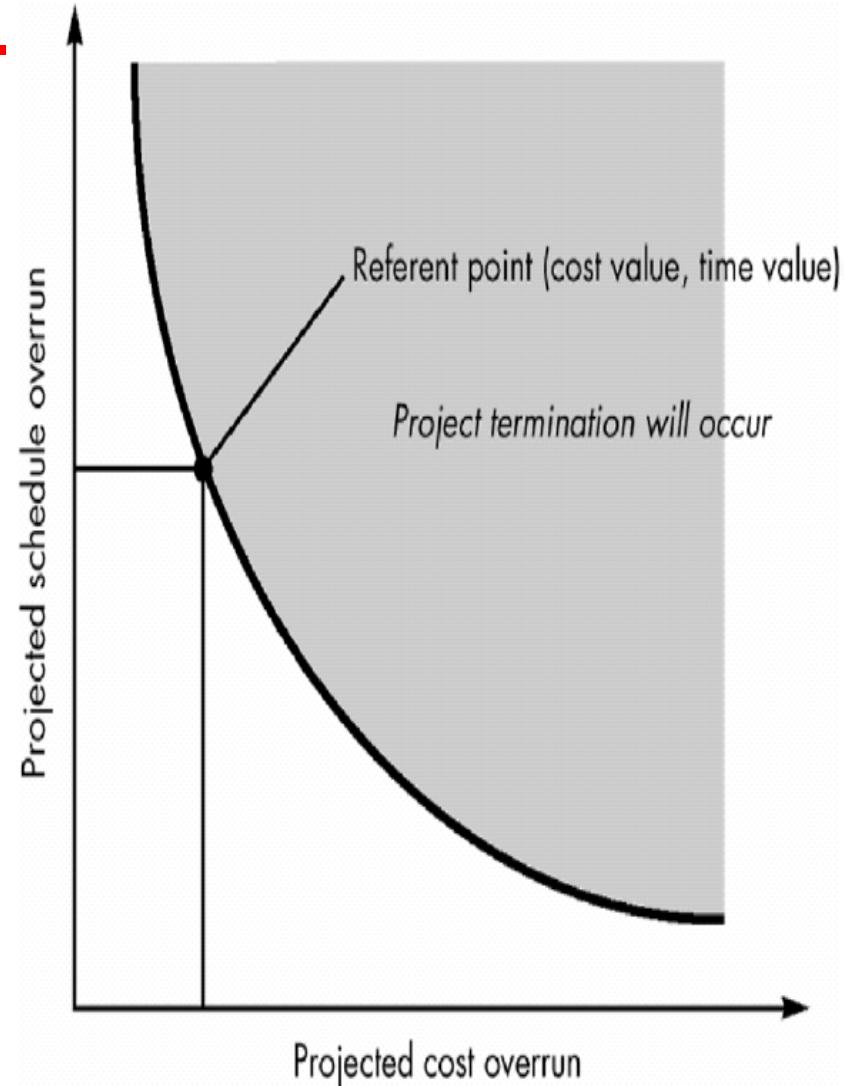
**Risk probability.** 80% (likely).

**Risk impact.** 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is \$14.00, the overall cost (impact) to develop the components would be  $18 \times 100 \times 14 = \$25,200$ .

**Risk exposure.**  $RE = 0.80 \times 25,200 \sim \$20,200$ .

# Risk Assessment

- For assessment to be useful, a *risk referent level* must be defined.
- In the context of software risk analysis, a risk referent level has a single point, called the *referent point* or *break point*, at which the decision to proceed with the project or terminate it (problems are just too great) are equally weighted.



# A framework for dealing with risk

---

*The planning for risk includes these steps:*

- Risk identification – what risks might there be?
- Risk analysis and prioritization – which are the most serious risks?
- Risk planning – what are we going to do about them?
- Risk monitoring – what is the current state of the risk?

# Risk identification

---

*Approaches to identifying risks include:*

- Use of **checklists** – usually based on the experience of past projects (previous slides)
- **Brainstorming** – getting knowledgeable stakeholders together to pool concerns
- **Causal mapping** – identifying possible chains of cause and effect

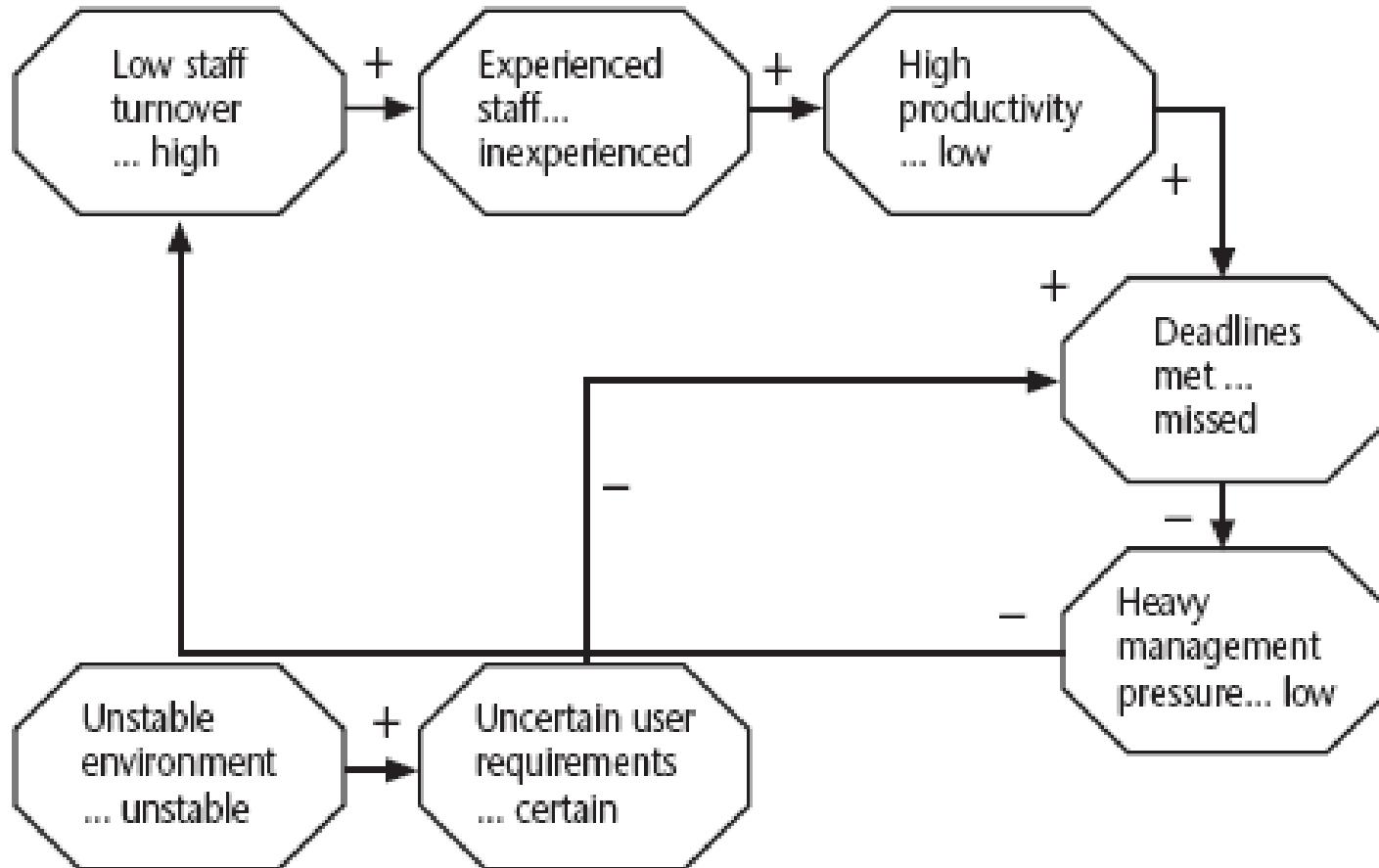
# Boehm's top 10 development risks

<i>Risk</i>	<i>Risk reduction techniques</i>
Personnel shortfalls	Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel
Unrealistic time and cost estimates	Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods
Developing the wrong software functions	Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals
Developing the wrong user interface	Prototyping; task analysis; user involvement

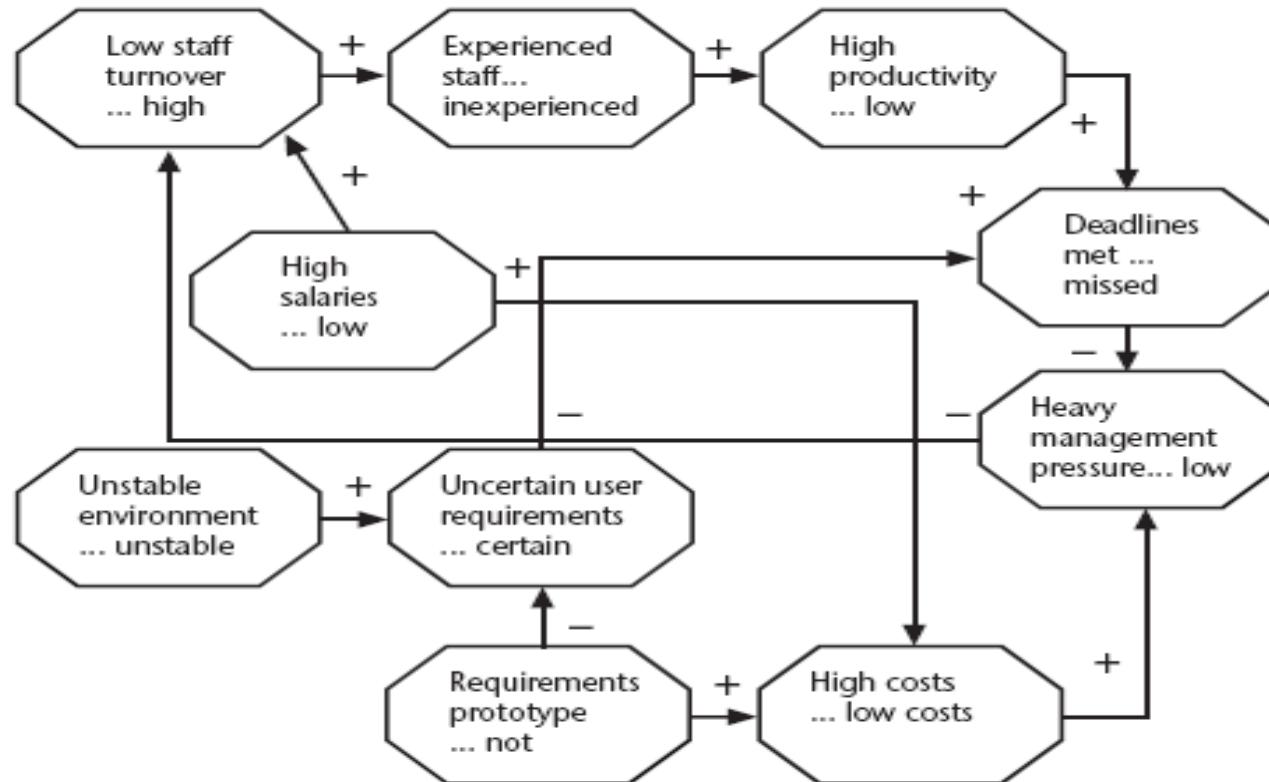
# Boehm's top ten risk – contd.,

Gold plating	Requirements scrubbing, prototyping, design to cost
Late changes to requirements	Change control, incremental development
Shortfalls in externally supplied components	Benchmarking, inspections, formal specifications, contractual agreements, quality controls
Shortfalls in externally performed tasks	Quality assurance procedures, competitive design etc
Real time performance problems	Simulation, prototyping, tuning
Development technically too difficult	Technical analysis, cost-benefit analysis, prototyping , training

# Causal mapping



# Causal mapping - interventions



A causal/cognitive map of a problem area with proposed solutions

# Risk prioritization

---

Risk exposure (RE) = (potential damage) x (probability of occurrence)

*Ideally*

- **Potential damage:** a money value e.g. a flood would cause \$0.5 millions of damage
- **Probability** 0.00 (absolutely no chance) to 1.00 (absolutely certain) e.g. 0.01 (one in hundred chance)  
$$RE = \$0.5m \times 0.01 = \$5,000$$
- analogous to the amount needed for an insurance premium

# Risk Exposure: Example

Ref	Hazard	Likelihood	Impact	Risk Exposure
R1	Changes to requirements specification during coding	8	8	64
R2	Specification takes longer than expected	3	7	21
R3	Significant staff sickness affecting critical path activities	5	7	35
R4	Significant staff sickness affecting non-critical activities	10	3	30
R5	Module coding takes longer than expected	4	5	20
R6	Module testing demonstrates errors or deficiencies in design	4	8	32

# Risk probability: qualitative descriptors

---

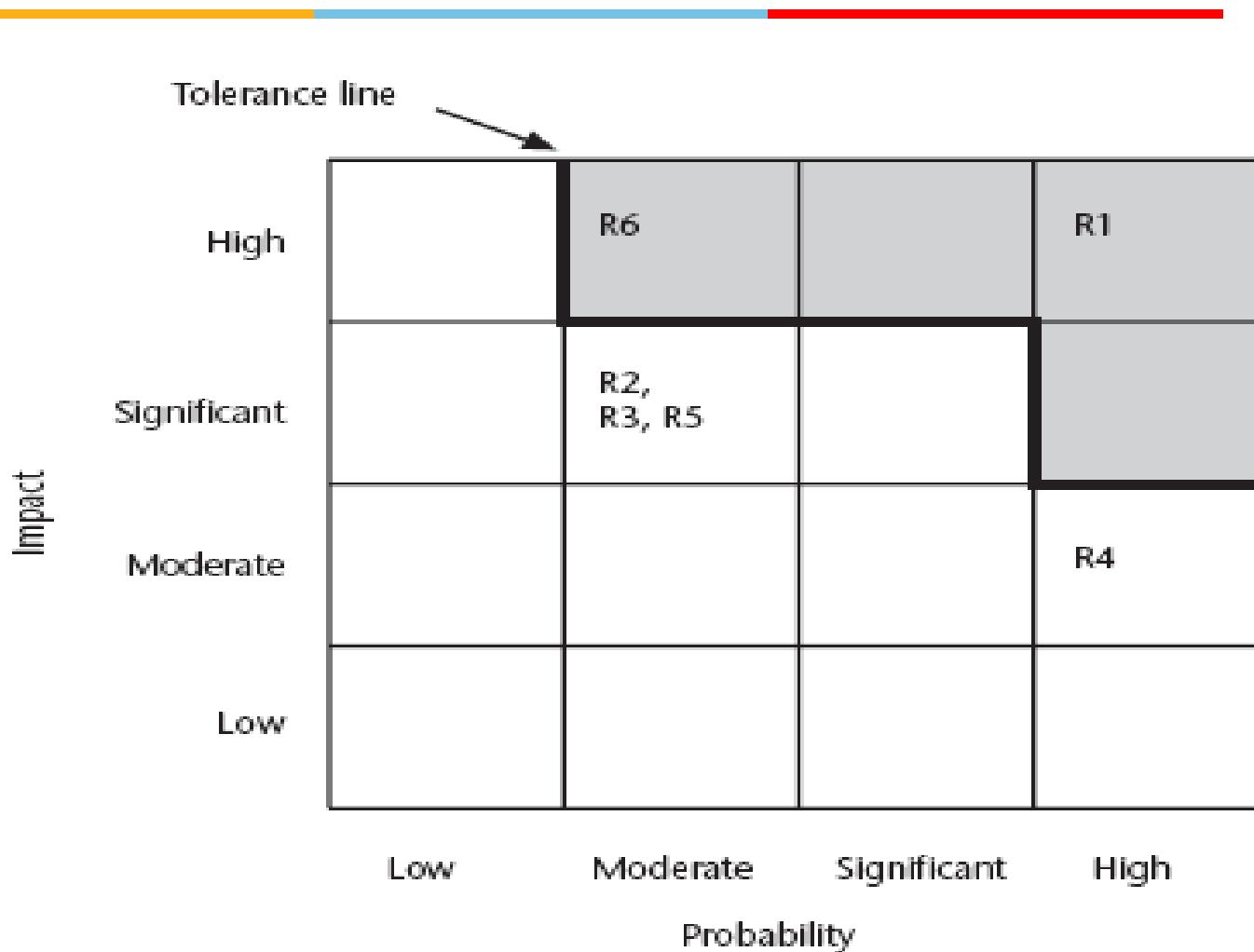
<i>Probability level</i>	<i>Range</i>
High	Greater than 50% chance of happening
Significant	30-50% chance of happening
Moderate	10-29% chance of happening
Low	Less than 10% chance of happening

# Qualitative descriptors of impact on cost and associated range values

---

<i>Impact level</i>	<i>Range</i>
High	Greater than 30% above budgeted expenditure
Significant	20 to 29% above budgeted expenditure
Moderate	10 to 19% above budgeted expenditure
Low	Within 10% of budgeted expenditure.

# Probability impact matrix



# Risk planning

---

*Risks can be dealt with by:*

- Risk acceptance
- Risk avoidance
- Risk reduction
- Risk transfer
- Risk mitigation/contingency measures

# Risk reduction leverage

---

## Risk reduction leverage (RRL)

$$= (RE_{\text{before}} - RE_{\text{after}}) / (\text{cost of risk reduction})$$

$RE_{\text{before}}$  is risk exposure before risk reduction e.g. 1% chance of a fire causing \$200k damage

$RE_{\text{after}}$  is risk exposure after risk reduction e.g. fire alarm costing \$500 reduces probability of fire damage to 0.5%

$$\text{RRL} = (1\% \text{ of } \$200k) - (0.5\% \text{ of } \$200k) / \$500 = 2$$

$\text{RRL} > 1.00$  therefore worth doing

# Risk Due to Product Size



*Product Attributes that affect risk:*

---

- estimated size of the product in LOC or FP?
- estimated size of product in number of programs, files and transactions?
- percentage deviation in size of product from average of previous products?
- size of database created or used by the product?
- number of users of the product?
- number of projected changes to the requirements for the product? before delivery? after delivery?
- amount of reused software?

# Risk Due to Business Impact



*Attributes that affect risk:*

---

- effect of this product on company revenue?
- visibility of this product by senior management?
- reasonableness of delivery deadline?
- number of customers who will use this product
- interoperability constraints
- sophistication of end users?
- amount and quality of product documentation that must be produced and delivered to the customer?
- governmental constraints
- costs associated with late delivery?
- costs associated with a defective product?

# Risks Due to the Customer



*Questions that must be answered:*

---

- Have you worked with the customer in the past?
- Does the customer have a solid idea of requirements?
- Has the customer agreed to spend time with you?
- Is the customer willing to participate in reviews?
- Is the customer technically sophisticated?
- Is the customer willing to let your people do their job—that is, will the customer resist looking over your shoulder during technically detailed work?
- Does the customer understand the software engineering process?

# Risks Due to Process Maturity

---

*Questions that must be answered:*

- Have you established a common process framework?
- Is it followed by project teams?
- Do you have management support for software engineering
  
- Do you have a proactive approach to SQA?
- Do you conduct formal technical reviews?
- Are CASE tools used for analysis, design and testing?
  
- Are the tools integrated with one another?
- Have document formats been established?

# Technology Risks

---

*Questions that must be answered:*

- Is the technology new to your organization?
- Are new algorithms, I/O technology required?
- Is new or unproven hardware involved?
- Does the application interface with new software?
- Is a specialized user interface required?
- Is the application radically different?
- Are you using new software engineering methods?
- Are you using unconventional software development methods, such as formal methods, AI-based approaches, artificial neural networks?
- Are there significant performance constraints?
- Is there doubt the functionality requested is "do-able?"

# Risk Management: In short...

---

- Risk Management = Project Management!
- Be proactive, anticipative
- Little ‘pessimism’ helps!
- Unanticipated risks? – escalate, escalate!

---

# Thank You

## Any Questions?



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Risk Management

BITS Pilani  
Viswanathan Hariharan  
2015



# Agenda

---

- Introduction
- Risk management framework
  - Risk identification
  - Risk analysis
  - Risk planning
  - Risk monitoring

# Introduction

---

- There are 2 terms commonly used in Project management – Risk & Issue
- Risk is a problem which has a certain probability of occurring
  - Example: Due to use of new technology there is a probability of cost over run
- Issue is a problem that has already occurred
  - Example: 2 key people have resigned. So the project will be delayed by 1 month
- The probability of risk is <100%, whereas the probability of issue is 100%, since it has already occurred

# Introduction ...

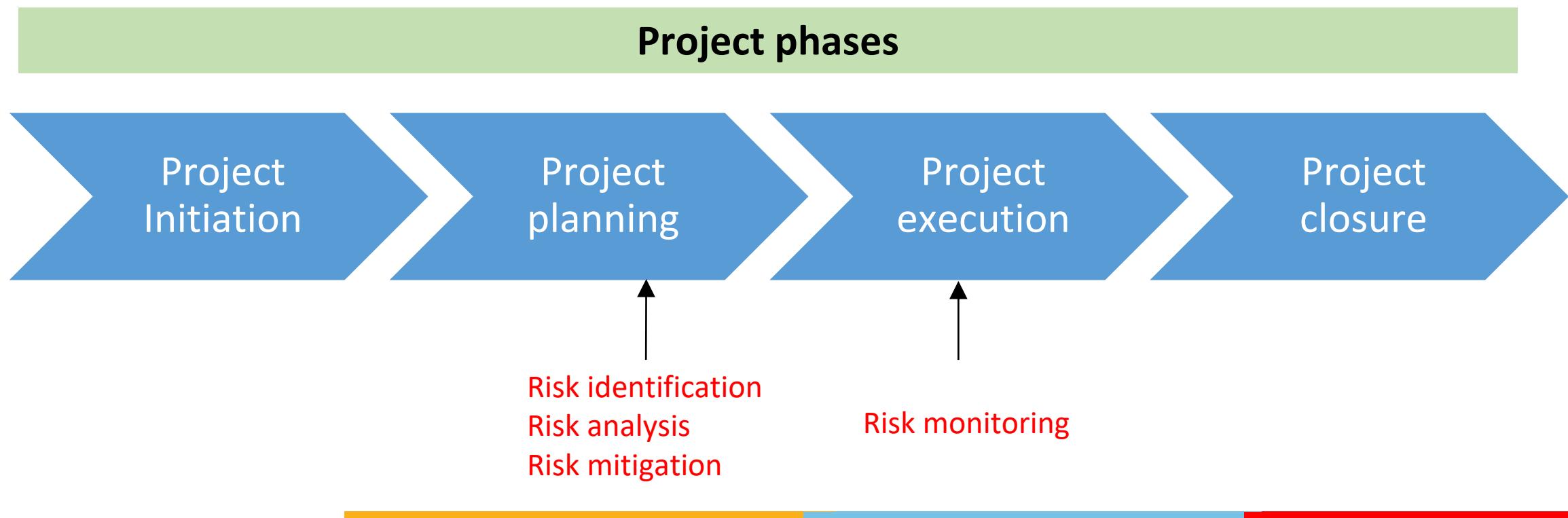
---

- Risk is associated with a ‘Cause’ & ‘Effect’
- Examples:

Cause	Effect
Staff inexperience	Planned effort & time is exceeded
Lack of top management commitment	Delay in getting approvals to plans changes
New technology	Development & testing takes longer than planned
Users are uncertain of their requirements	Project scope increases

# Risk management framework

Risk management framework provides a way to manage risks in a project



# Risk identification

---

- During the project planning phase, we need to identify potential risks
- Risk can be of different types. Here are some questions to ask to identify potential risks:
  - Technology risks – Is the technology familiar to us? Is the technology proven?
  - People risks – Do we have sufficiently skilled people, Is there high attrition in the organization
  - Commitment risks – Are the sponsor, end users, other key stakeholders committed?
  - Requirement risks – Is the customer clear about their requirement
  - Business value risk - Is there a significant business value in doing the project?
  - Schedule risks – Is the schedule realistic?

# Risk identification approaches

---

- Checklist Approach: Use a checklist of different types of risks and see which ones are applicable & which ones are not applicable
- Brainstorm Approach: Get representatives of all stakeholders such as Sponsor, End users, Marketing team, developers and testers and brainstorm to identify potential risks from their perspective
  - Examples:
  - Sponsors may say that competition is already developing a new system. So if we do not complete this project in 6 months, then we would lose edge over competition and the project may no longer be beneficial
  - End users may say that they are not familiar with RFID technology, so it might delay implementation of the project

# Risk analysis & prioritization

---

- All risks may not be critical. So we need to prioritize the risks
- Steps to prioritize:
  - Determine the probability of each risk. Ex. Probability of risk is 20%
  - Determine the adverse impact of each risk on a scale of 1 to 10 (Low to High)
  - Multiply the 2 to arrive at the priority

# Risk prioritization

## Example

#	Risk description	Probability	Impact	Priority
1	Key people might leave	20%	8	1.6
2	Customer is not clear about the requirement	40%	8	<b>3.2</b>
3	Schedule may be tight	30%	6	1.8
4	Defects during system testing may be high	20%	6	1.2
5	Customer commitment may not be high	30%	6	1.8

From the table, it is clear that we need to focus our energy on addressing the risk of “Customer is not clear about the requirement”

# Risk Mitigation

There are different ways to address a risk

Risk mitigation strategy	When to use the strategy	Example
Risk acceptance	When the risk is very small (low impact & low probability), we may not take any action	If the deployment may get delayed by 2 weeks due to vacation period and if the impact is very low, we may not take any specific action
Risk avoidance	When the risk is very high and we can not afford to take the risk, we must avoid it.	When the technology is new and we have no expertise to build it, we may go for off the shelf product
Risk reduction	When the risk can be effectively addressed, we take some action to reduce the probability of risk or its impact	If we know that customer is not clear about requirements, then we could adopt an iterative approach or build a prototype to confirm the requirements
Risk transfer	When the impact is very high but the probability is low, we can adopt a Risk transfer strategy	Satellite launch projects go for insurance in case the launch fails. This transfers the risk to the Insurance company

# Best practices

#	Risk	Recommended action
1	Customer is not clear about requirement	Adopt an iterative approach, developing a small sub-set of core functionality. Get feedback on the developed system. Then define requirements for the next iteration
2	Domain is new	Develop detailed business process flows and domain models to validate understanding. Develop User interface storyboard to validate understanding
3	Technology is new	Develop a prototype using core requirements
4	Attrition	<p>Consider aspirations of team members and try to accommodate roles that will satisfy their aspirations such as design work, on-site role, etc.</p> <p>Involve team member in addressing project issues. This will increase belongingness</p> <p>Improve team bonding through team get-togethers</p>
5	Changes to requirements	Include only critical requirements based on business value. Shift the rest to next release
6	Tight schedule	<p>Get the best people</p> <p>Monitor project closely to identify issues</p> <p>Do not allow scope creep</p> <p>Remove non-critical requirements from scope</p>
7	System performance may be poor	Identify scenarios where performance could be an issue (eg. high volume transaction during holidays) and address them via effective design

# Risk monitoring

---

- During project execution phase, we need to regularly monitor the project for any new risks or change in probability of already identified risks
- Based on this we need to plan appropriate actions
- We should maintain a risk log and regularly monitor the progress of planned actions such as
  - Development of prototype using technology or
  - Training of developers in domain knowledge or
  - Communicating the benefits to end users to ensure there is no resistance at the time of deployment

# Risk monitoring

## Key things to monitor to identify new risks

1. Customer commitment: Are customers attending meetings regularly? Do they respond promptly? Do they approve changes on time?
2. Schedule and effort variance: Is the variance significant? If so, why?
3. People skills, team work and motivation levels: Do the team members have required skills? Is there any attrition? Are team members cooperative?
4. Quality of work products: Is the customer happy with User interface design? How severe are the design review comments? What is the system test defect density?
5. Scope change: How many change requests are we getting every month? What is the cumulative effort of all change requests?

# Case study: Developing software on Tandem Non-Stop computer

---

- In order to serve the customers better, a bank in Europe asked an IT company to develop a “Unified Customer Information System” that would gather data from various back-end systems such as Savings account system, Loan system, etc. and provide a unified view of customer to the customer service officers.
- Since this was a business critical system used by 400+ branches in Europe, it was decided to host it on a fault tolerant computer – Tandem Non-Stop computer “Himalaya”
- While the requirements were gathered, the IT company placed an order for the Tandem computer which was to be delivered in 2 months
- However, it was realized that it would take 4 months instead of 2.
- So a risk management plan was put in place.
  - It was decided that the software would be developed on a Unix box which is similar to Tandem’s OS.
  - Once Tandem computer arrives, the software would be ported to Tandem. During the porting, the Unix system calls would be replaced with Tandem’s system calls
- This enabled the company to deliver the software on time, though some extra effort had to be spent during transition from Unix to Tandem



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

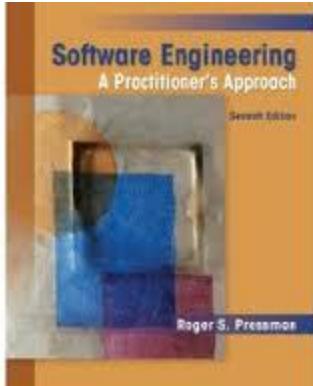
# SS ZG622:

## Software Project Management

### (Quality Planning in Projects)

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad

# Text Books



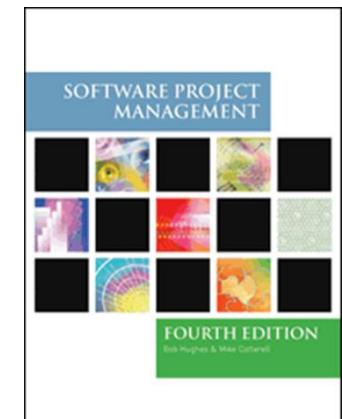
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Quality Planning in Projects**

Ref: T1 / Chap-24

# Topics

---

- Quality Management in Projects
- Software Quality Attributes
- Contents of a Quality Plan
- Testing, Reviews and inspections
- Software measurement and metrics

# Software quality

---

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
  - Some quality requirements are difficult to specify in an unambiguous way;
  - Software specifications are usually incomplete and often inconsistent.
- The focus may be ‘fitness for purpose’ rather than specification conformance.

# Software Quality: fitness for purpose!

---

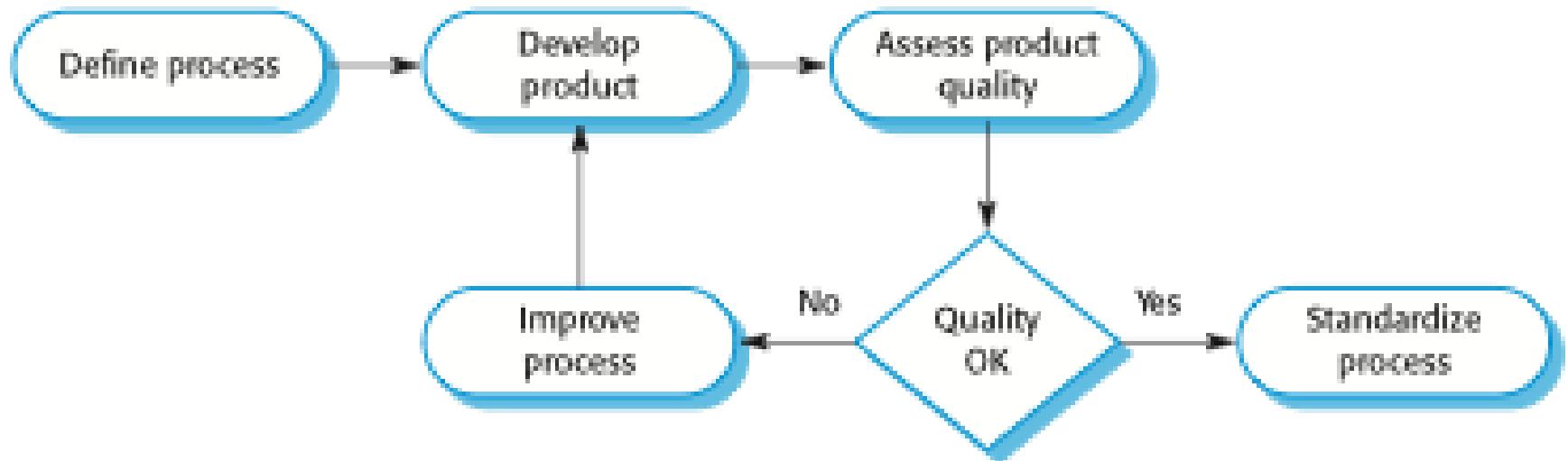
- Have programming and documentation standards been followed in the development process?
- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?

# Software quality attributes

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

# Process-based quality



# Process quality influences



## Product quality

---

- The quality of a developed product is influenced by the quality of the production process.
- This is important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood relationship between software processes and product quality.
  - The application of individual skills and experience is particularly important in software development;
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

# Product and process standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious form filling work is often involved to maintain the documentation associated with the standards.

# ISO 9001 standards framework

---

- An international set of standards that can be used as a basis for developing quality management systems.
- ISO 9001, the most general of these standards, applies to organizations that design, develop and maintain products, including software.
- The ISO 9001 standard is a framework for developing software standards.
  - It sets out general quality principles, describes quality processes in general and lays out the organizational standards and procedures that should be defined. These should be documented in an organizational quality manual.

# ISO 9001 core processes

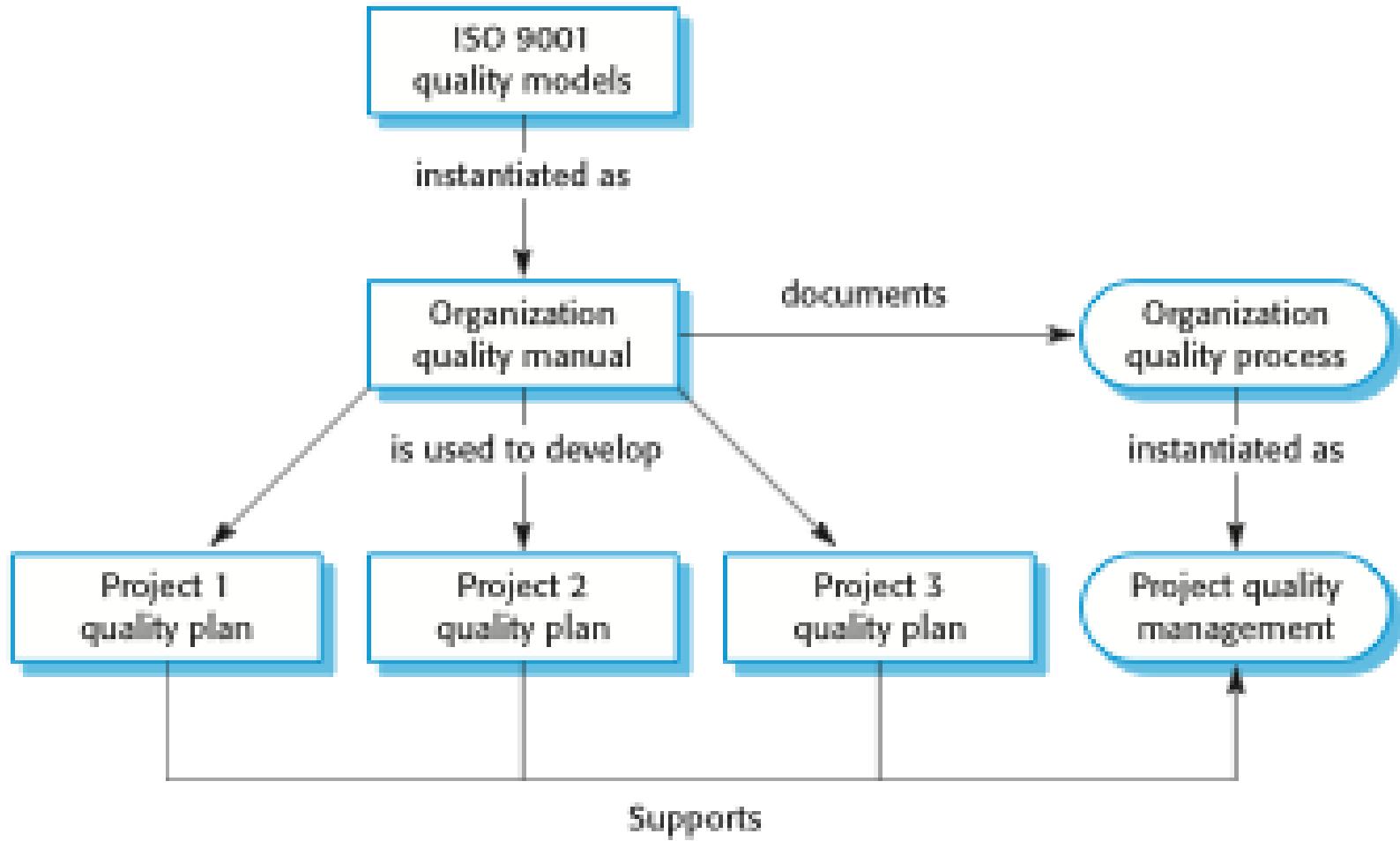
## Product delivery processes



## Supporting processes



# ISO 9001 and quality management



# Software quality management

---

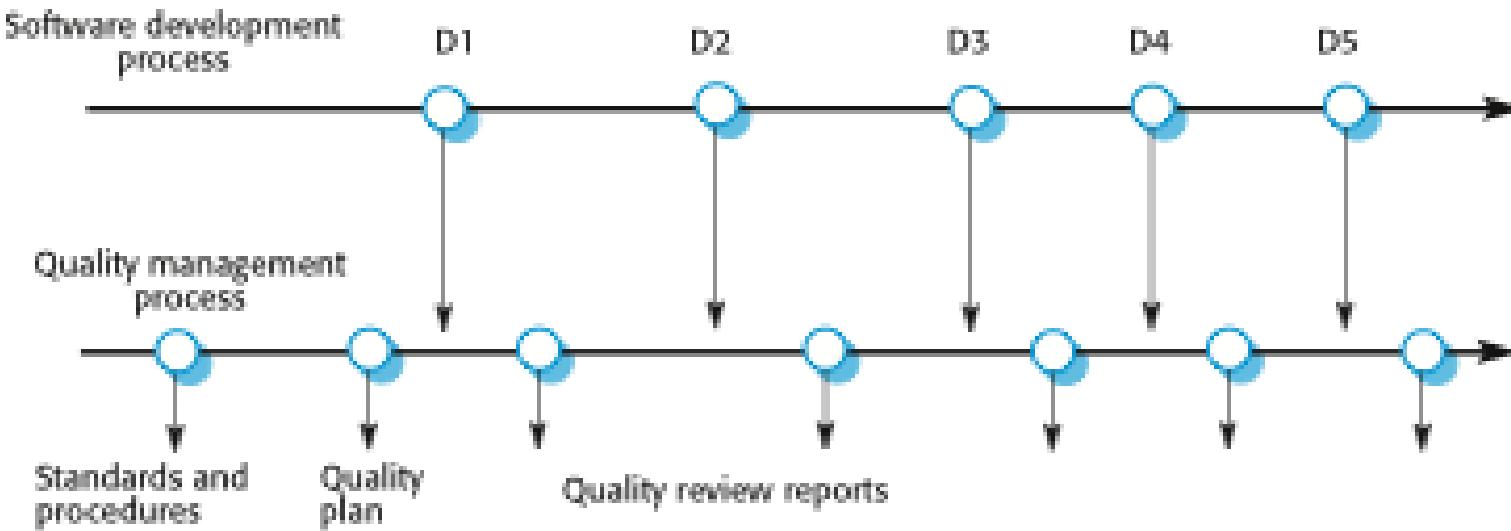
- Concerned with ensuring that the required level of quality is achieved in a software product.
- Three principal concerns:
  - At the **organizational level**, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the **project level**, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the project level, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the **quality goals** for the project and define what **processes and standards** are to be used.

# Quality management activities

---

- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals
- The quality team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

# Quality management and software development



# Scope of quality management

---

- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture (or Agile Methodology).

# Quality planning

---

- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.
- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents
- If they are too long, no-one will read them.

# Typical contents of a quality plan

- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- Documentation (of work items) to be produced
- standards, checklists, practices and conventions
- reviews and audits (schedule)
- testing
- problem reporting and corrective action
- tools, techniques, and methodologies
- code, media and supplier control
- records collection, maintenance and retention
- training
- risk management

# Software Quality Planning: Summary



- Software quality management is concerned with ensuring that software has a low number of defects and that it reaches the required standards of maintainability, reliability, portability and so on.
- SQM includes defining standards for processes and products and establishing processes to check that these standards have been followed.
- Software standards are important for quality assurance as they represent an identification of 'best practice'.
- Quality management procedures may be documented in an organizational quality manual, based on ISO 90001 model or CMMI framework

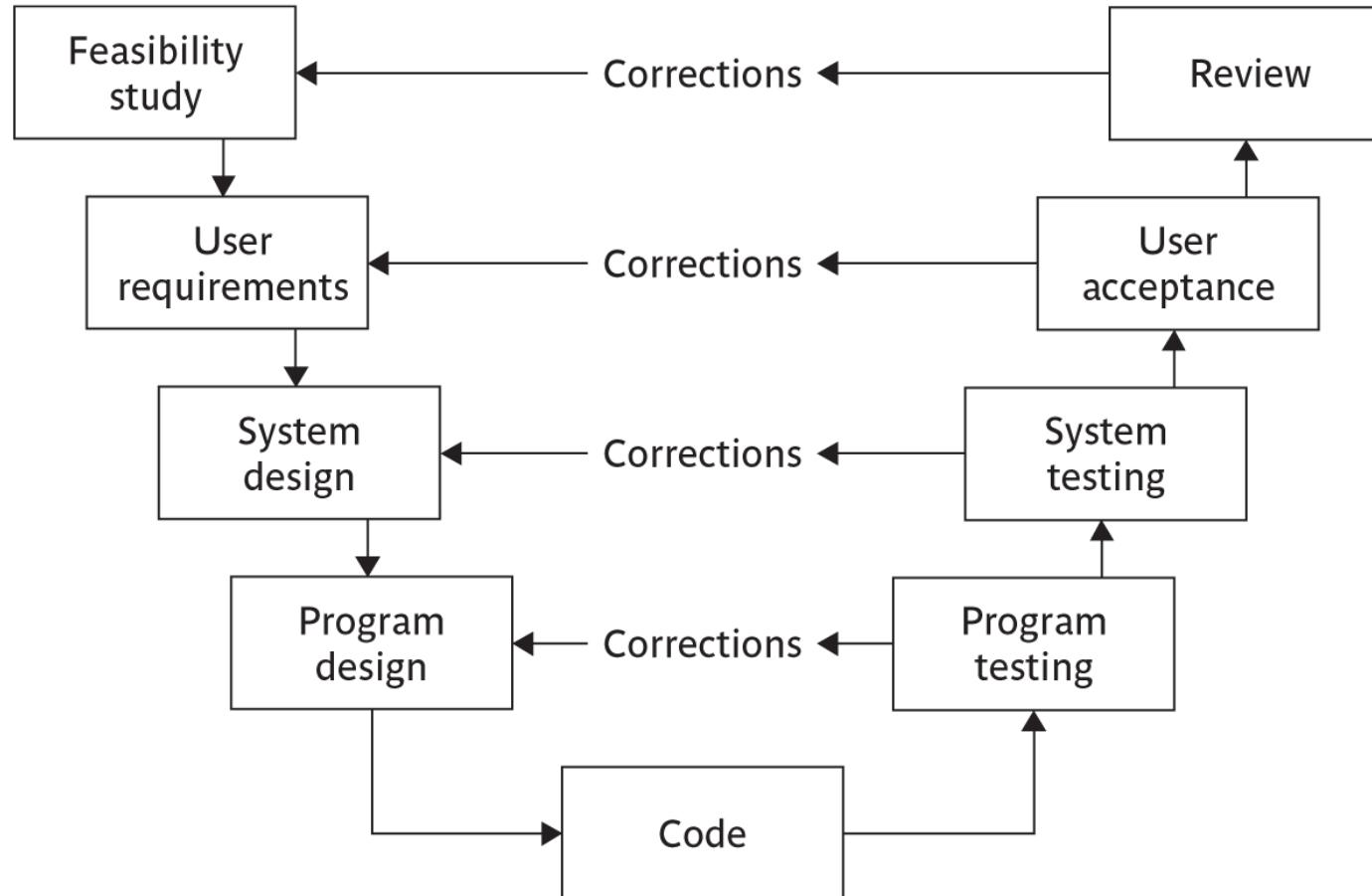
---

# Thank You

## Any Questions?

Next... ➔ Testing, Reviews/Inspections,  
Software Metrics

# Testing: the V-process model



# Black box vs. White box testing

---

- White box testing
  - The tester is aware of the internal structure of the code; can test each path; can assess percentage test coverage of the tests e.g. proportion of code that has been executed
- Black box testing
  - The tester is not aware of internal structure; concerned with degree to which it meets user requirements

# Test plans

- Specify test environment
  - In many cases, especially with software that controls equipment, a special test system will need to be set up
- Usage profile
  - failures in operational system more likely in the more heavily used components
  - Faults in less used parts can lie hidden for a long time
  - Testing heavily used components more thoroughly tends to reduce number of operational failures

# Management of testing

---

The tester executes test cases and may as a result find discrepancies between actual results and expected results – **issues**

**Issue resolution** – could be:

- a mistake by tester
- a fault – needs correction
- a fault – may decide not to correct: **off-specification**
- a change – software works as specified, but specification wrong: submit to change control

# When to stop testing?

---

- The problem: impossible to know there are no more errors in code
- Need to estimate how many errors are likely to be left

**Bug seeding** – insert (or leave) known bugs in code

Estimate of bugs left =

$$\frac{\text{total errors found}}{\text{seeded errors found}} \times \text{total seeded errors}$$

# Reviews and inspections

---

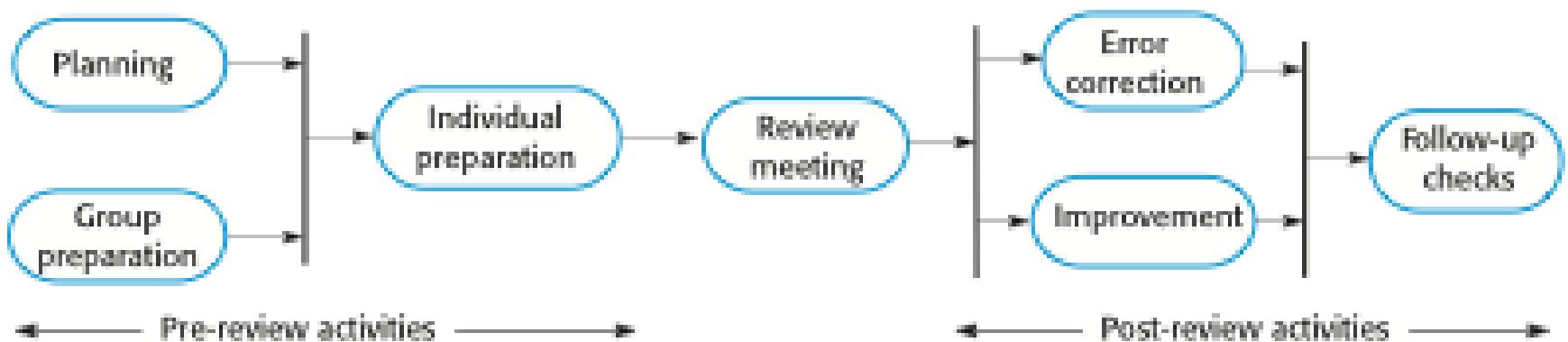
- A group examines part or all of a process or system and its documentation to find potential problems.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.
- There are different types of review with different objectives
  - Inspections for defect removal (product);
  - Reviews for progress assessment (product and process);
  - Quality reviews (product and standards).

# Quality reviews

---

- A group of people carefully examine part or all of a software system and its associated documentation.
- Code, designs, specifications, test plans, standards, etc. can all be reviewed.
- Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

# The software review process



# Reviews and agile methods

---

- The review process in agile software development is usually informal.
  - In Scrum, for example, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member.
- XP relies on individuals taking the initiative to improve and refactor code. Agile approaches are not usually standards-driven, so issues of standards compliance are not usually considered.

# Program inspections

---

- These are **peer reviews** where engineers examine the source of a system with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

# Inspection checklists

---

- Checklist of common errors should be used to drive the inspection.
- Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
- In general, the 'weaker' the type checking, the larger the checklist.
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

# An inspection checklist (a)

Fault class	Inspection check
Data faults	<ul style="list-style-type: none"> <li>• Are all program variables initialized before their values are used?</li> <li>• Have all constants been named?</li> <li>• Should the upper bound of arrays be equal to the size of the array or Size -1?</li> <li>• If character strings are used, is a delimiter explicitly assigned?</li> <li>• Is there any possibility of buffer overflow?</li> </ul>
Control faults	<ul style="list-style-type: none"> <li>• For each conditional statement, is the condition correct?</li> <li>• Is each loop certain to terminate?</li> <li>• Are compound statements correctly bracketed?</li> <li>• In case statements, are all possible cases accounted for?</li> <li>• If a break is required after each case in case statements, has it been included?</li> </ul>
Input/output faults	<ul style="list-style-type: none"> <li>• Are all input variables used?</li> <li>• Are all output variables assigned a value before they are output?</li> <li>• Can unexpected inputs cause corruption?</li> </ul>

# An inspection checklist (b)

Fault class	Inspection check
Interface faults	<ul style="list-style-type: none"> <li>• Do all function and method calls have the correct number of parameters?</li> <li>• Do formal and actual parameter types match?</li> <li>• Are the parameters in the right order?</li> <li>• If components access shared memory, do they have the same model of the shared memory structure?</li> </ul>
Storage management faults	<ul style="list-style-type: none"> <li>• If a linked structure is modified, have all links been correctly reassigned?</li> <li>• If dynamic storage is used, has space been allocated correctly?</li> <li>• Is space explicitly deallocated after it is no longer required?</li> </ul>
Exception management faults	<ul style="list-style-type: none"> <li>• Have all possible error conditions been taken into account?</li> </ul>

# Agile methods and inspections

---

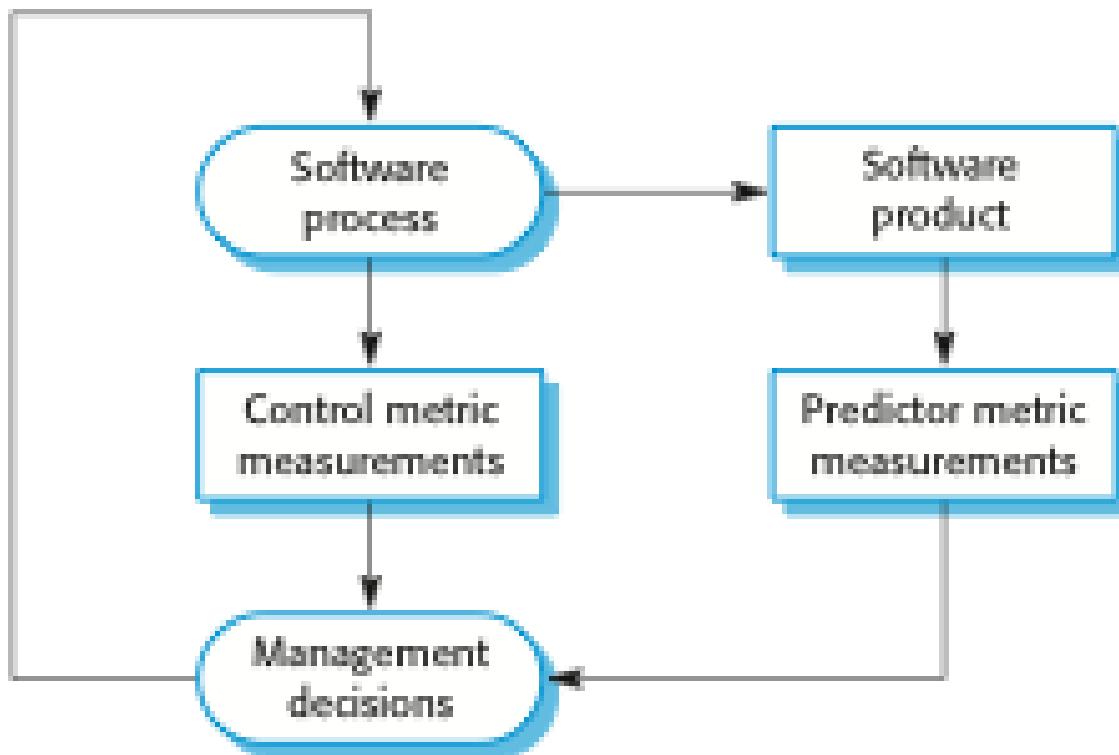
- Agile processes rarely use formal inspection or peer review processes.
- Rather, they rely on team members cooperating to check each other's code, and informal guidelines, such as 'check before check-in', which suggest that programmers should check their own code.
- Extreme programming practitioners argue that pair programming is an effective substitute for inspection as this is, in effect, a continual inspection process.
- Two people look at every line of code and check it before it is accepted.

# Software metrics

---

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- **Product metrics** can be used for general predictions or to identify anomalous components.
- **Process metrics** indicate the capability of the organization in delivering Product quality

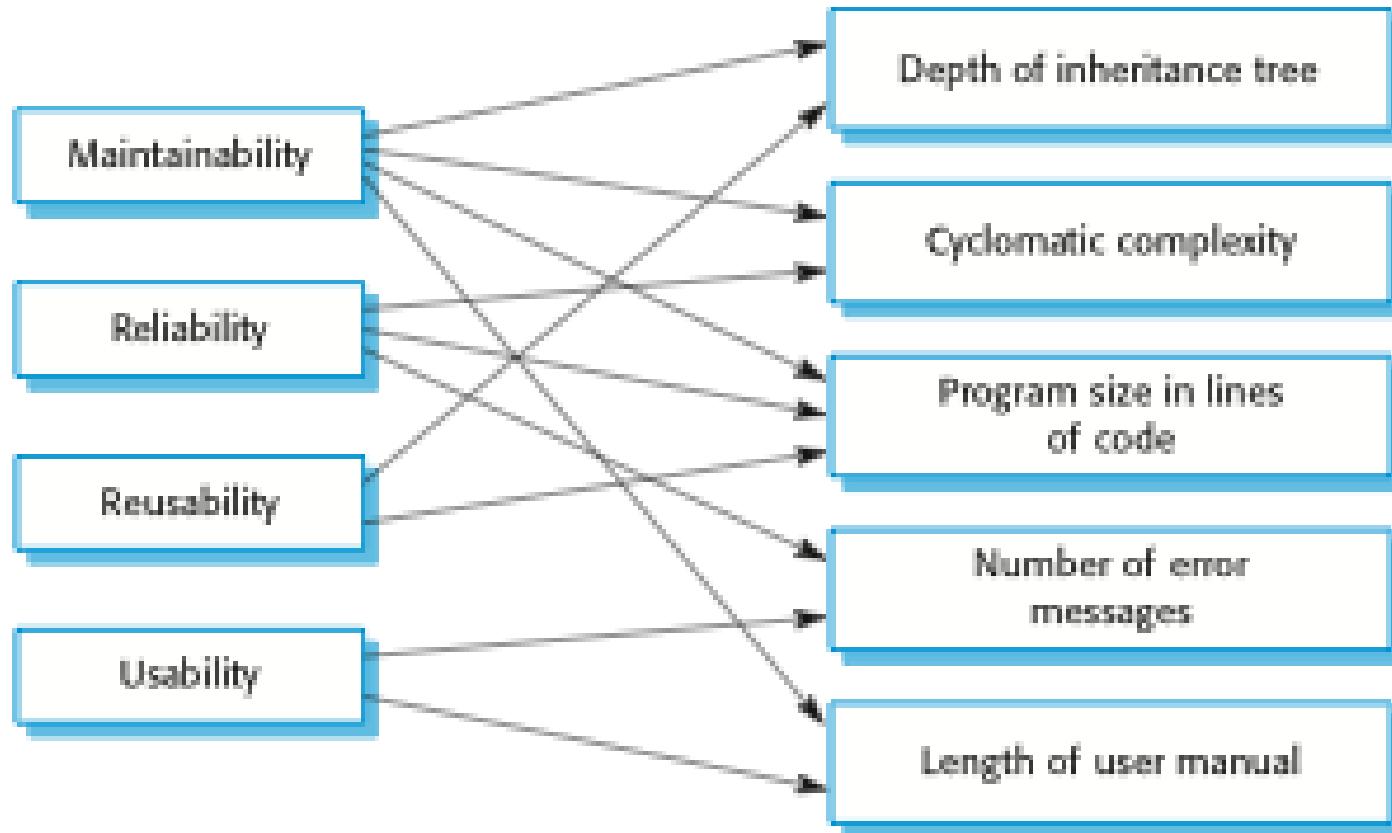
# Metrics are key for Process control



# Relationships between internal and external software

## External quality attributes

## Internal attributes



# Reviews & Inspections: Key points

---

- Reviews of the software process deliverables involve a team of people who check that quality standards are being followed.
- In a program inspection or peer review, a small team systematically checks the code. They read the code in detail and look for possible errors and omissions
- Software measurement can be used to gather data about software and software processes.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems.

---

# Thank You

## Any Questions?



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad



# Software Project Management – Caselet 3- Change Request (CR) Management

S Subramanian ( Subbu)

# Caselet (CR Management) context



- Software change is inevitable
  - New requirements emerge when the software is used
  - The business environment changes
  - Errors must be repaired
  - New equipment must be accommodated
  - The performance or reliability may have to be improved
- To identify an issue - if is a CR or defect (bug)
- CRs can also provide business opportunities.

# How to identify an issue as a CR or not



- Follow standards in creating Business Requirements Documents (BRD)
- BRD to be Signed ( by client) and Versioned/Baselined
  - Both functional and non functional
- Run all CRs against approved BRD
  - Role of Business analyst and Architect vital
- Show flexibility in bucketing CRs
  - Defects/ New Requirements/ Minor CRs

# Change Control Board Process (CCB)



- Upfront set up CCB with right representation
- Ensure CCB process compliance.
- Create Requirement Traceability matrix (RTM) and maintain
- CR - Impact analysis
  - Cost
  - Schedule
  - effort

# Summary

---

- Change is constant
  - Tool based Software Configuration Management
  - Strictly follow CCB process
  - Be flexible and realistic
-





**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

# Change Management

BITS Pilani  
Viswanathan Hariharan  
2015



# Agenda

---

1. Introduction
2. Case Study: Uncontrolled scope change
3. Change control mechanism

# Introduction

---

- No matter how well we plan, some changes are inevitable
- Examples of changes:
  - Additional requirements, since some requirements were missed out during requirement gathering phase
  - Omission of some project activities such as Data migration
- If we do not have a mechanism to control the changes to the project, it may jeopardize the project

# Case study: Uncontrolled changes

---

- An IT company was developing a new Marketing Support system for a Health care services company, with the objective of gaining a higher market share
- The estimated time was 1 year
- The project started well. However, soon, change requests started trickling in and the scope kept getting bigger & bigger
- It had been agreed that the IT company would evaluate the impact of change in terms of cost and schedule, get approval from the client and only then implement the change.
- Due to several change requests the project ultimately finished 1 year late
- Valuable time had been lost and the objective of gaining higher market share was not met
- The client was extremely disappointed and blamed the IT company for the delay
- A big lesson was learnt in Change control – Evaluate the impact of change request based on its impact on business goal, not just on cost & time

# Change control mechanism

---

Steps to control changes in scope of a project

1. Determine the benefit (value) of the Change Request (CR) to business
  - This needs to be done by the sponsor
2. Determine additional cost & time (impact) to incorporate the CR.
  - This should include effort for all activities such as design change, change to DB, coding, design of additional test cases, documentation, etc.
3. If the CR is valuable to business and if the impact is acceptable to sponsors, then proceed with the CR

# Determining value of CR

CRs can be categorized as follows:

- **Essential** (Must Have): Ex. A monthly statement feature in a banking application
- **Desirable** (Offers benefits but not essential): Ex. Display last 10 transactions
- **Cosmetic** (Affects Look & Feel): Ex. Change sequence of Menu items on the screen

# Best practices

---

- Keep track of all changes in a CR log
- Do not incorporate changes without approval from sponsors
- Publish a metric that highlights the extent of change from original plan,

Example

- 20% increase in effort due to CRs,
- 10% increase in duration of project,
- # of essential CRs,
- # of cosmetic CRs



**BITS Pilani**

Pilani | Dubai | Goa | Hyderabad

# **Software Project Management**

## **Change Management: A Case Study**

Anita Ramachandran  
2015

# A Dynamic Project

- This case study deals with a hypothetical project in which:
  - time to market is critical
  - external software interfaces play a key role
  - market dynamics influence the requirements of the system
- Objectives:



Risk Analysis



Process Models



Requirements Changes



Schedule Changes

# The Scenario

- Mobile application development
- Aggregator application that collects movie and reservation data from IMAX theatres and displays consolidated show timing/booking/availability information
- IMAX theatre web interfaces publish APIs for:
  - Queries: queries for movies, show timings, seat availability, offers
  - Reservation: make/cancel reservations
  - Payment: make payment, reverse payment in case of cancellation
- User maintenance
- Clear requirements on UI, scaling, application functionality etc., but:
  - You don't have a confirmed list of IMAX movie theatres which your app should interface with
  - For some of these theatres, the APIs are under development stage or are published only based on partnerships

# Questions to Ponder

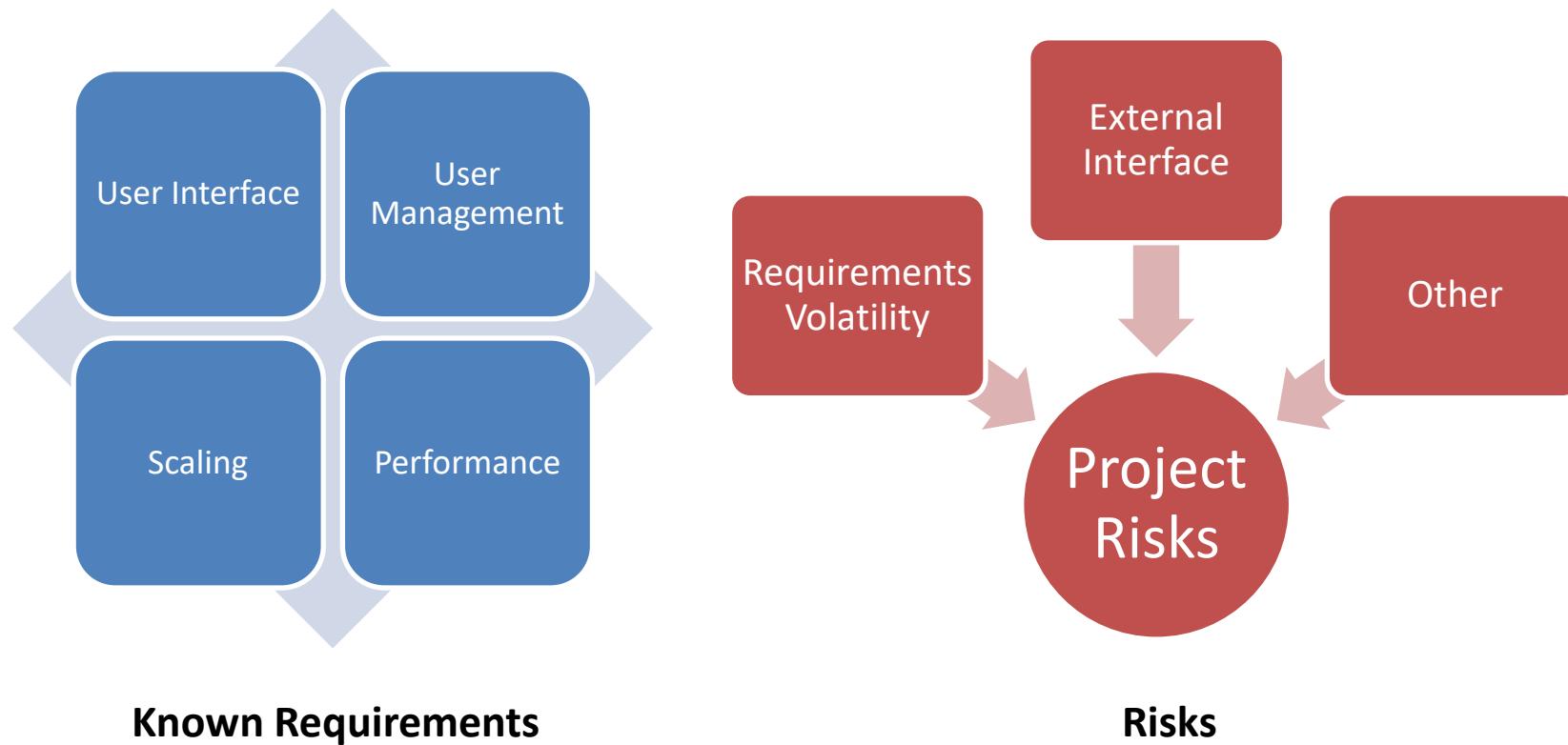
---

- What are the main features of your project, from a project management perspective? For e.g., what are the risks that your project might face? How would you evaluate those risks in terms of their probability and impact?
- What process model will you follow? How will you develop a detailed plan for this project?
- What activities would come under the critical path for the project?

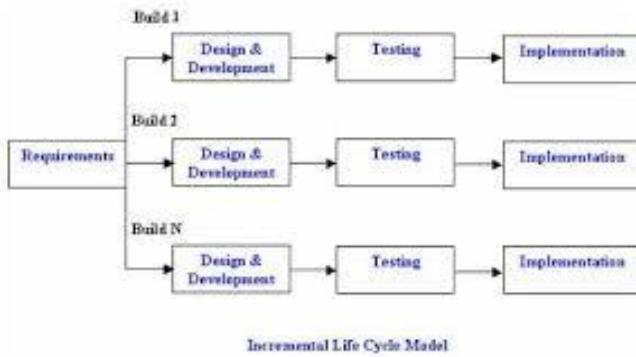
---

**Let's think over each question  
in detail**

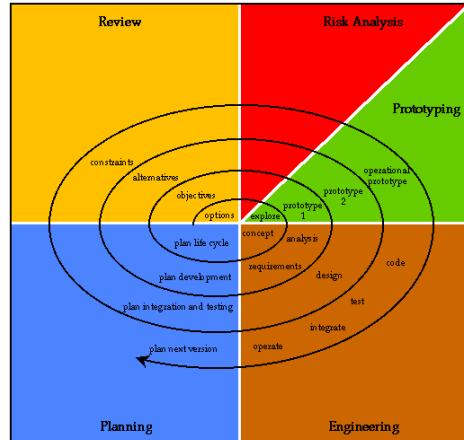
# Project Risks



# Process Models to Follow



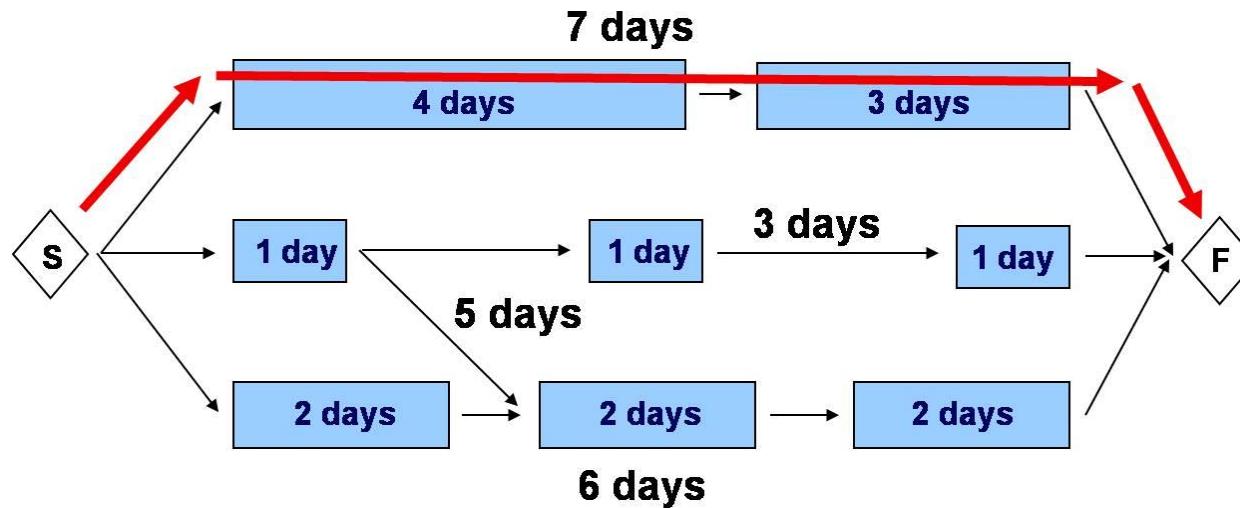
Incremental-iterative



- Advantage: Allows for flexibility in requirements changes

# Critical Path Analysis

- All activities that relate to the two risk factors - requirements volatility and freezing on external APIs
- In the beginning of the project, they are the activities that define the end date of development cycle
- As the requirements and external dependencies get baselined during the course of the project, there may be other activities (depending on our work breakdown structure) that drive the critical path



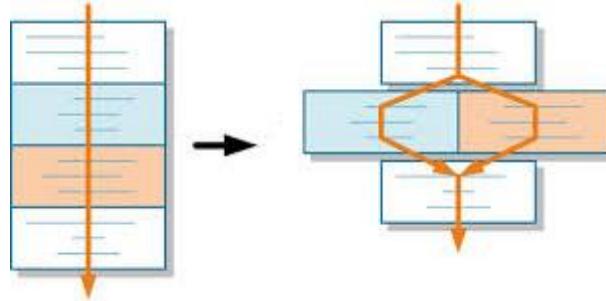
---

**Let's look at some scenarios that  
may arise during project  
execution**

# How to Handle Schedule Changes



Adding Resources



Parallelization



Phased Delivery



Component Reuse



Sub-contracting



Working in Shifts for  
Resource Sharing

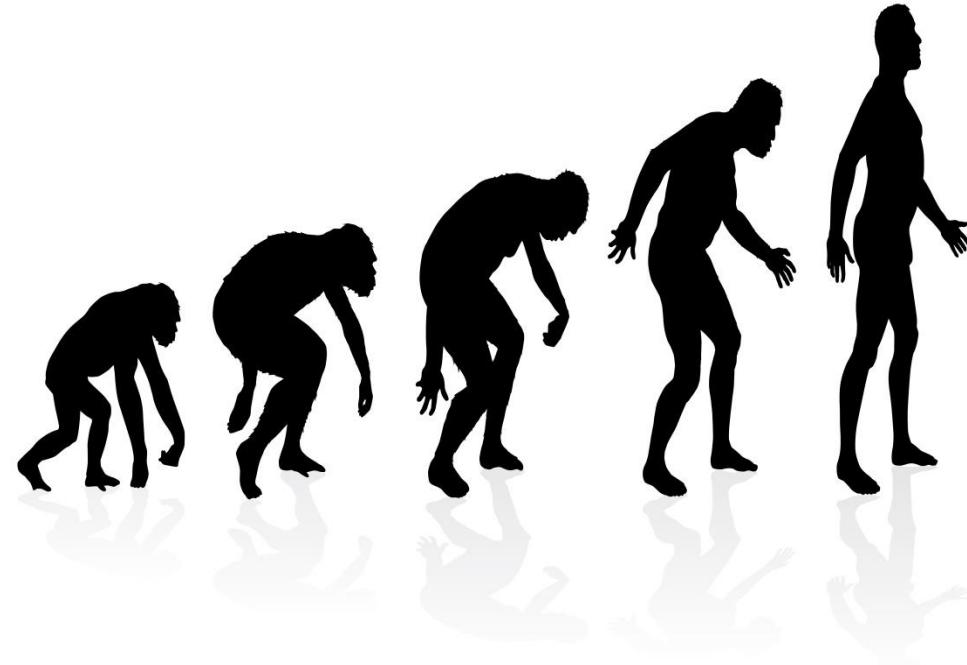
# How to Handle Additional Business Requirements

- New knowledge may change our design approach
- Evaluate if such changes, as placeholders for future enhancements, can be done in the current release
- Fit in those changes as an activity in the current release or as a user story in the upcoming sprints



# How to Handle Project Transition

- Phased transition, where there is continuous working together of both teams
- Identify the impacts of the transition on the current project schedule and quality
- Travel



# Summary

---

- Project with dynamic nature, requirements and schedule changes driven by competition
- Examined the following
  - Risk analysis, identification of process model
  - Scenarios
  - New marketing requirements
  - Shortened schedule
  - Project transition
- Exercise
  - Gather together as small groups
  - Think through the project characteristics
  - Identify a high level work breakdown structure
  - Provide a detailed analysis on each of the scenarios

---

# Thank you!

Image courtesy: Selected websites



SS ZG622:

# Software Project Management

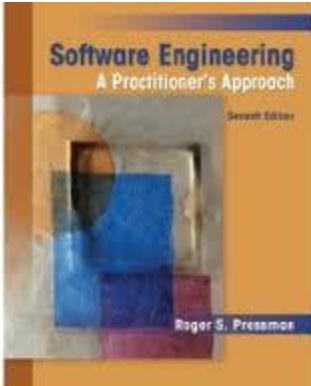
## (Project Monitoring / Earned Value Method)

**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad

# Text Books



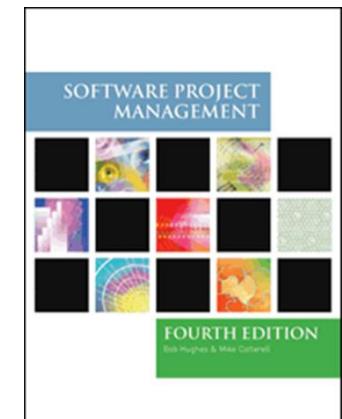
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations

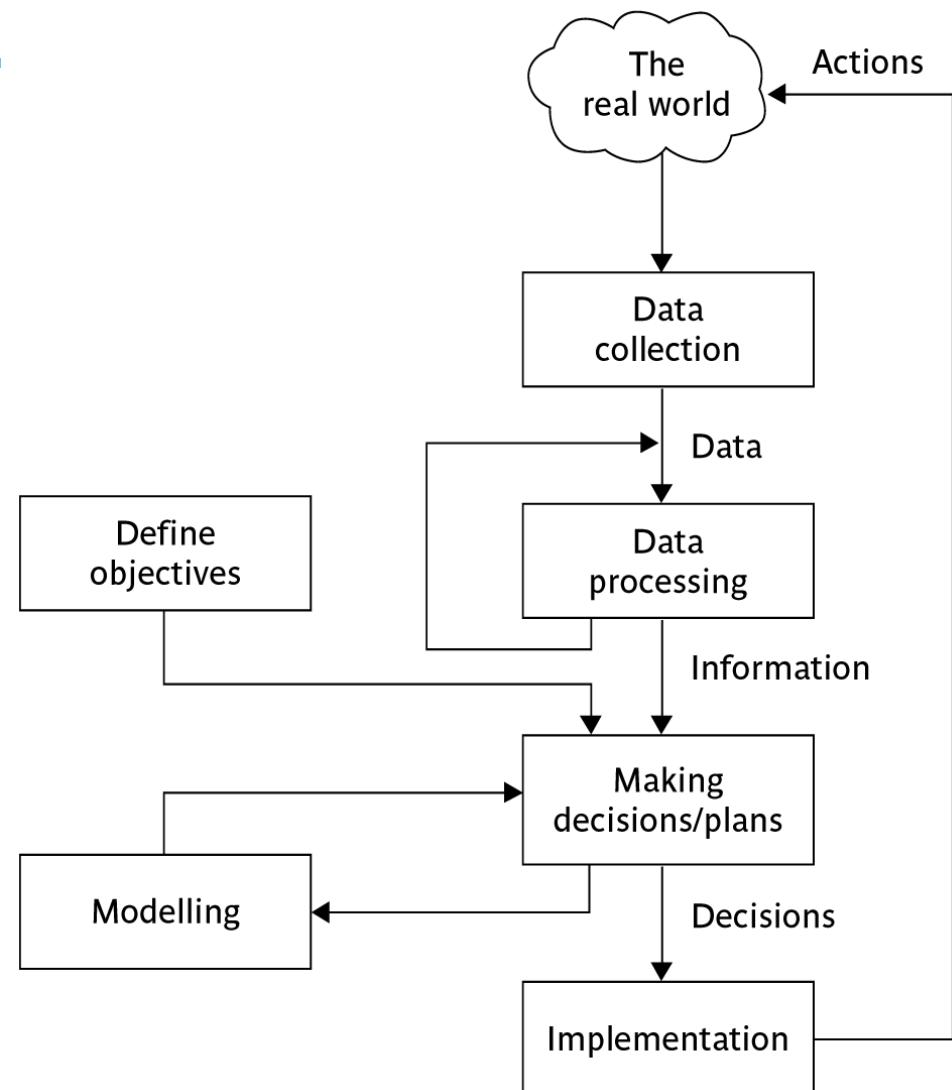




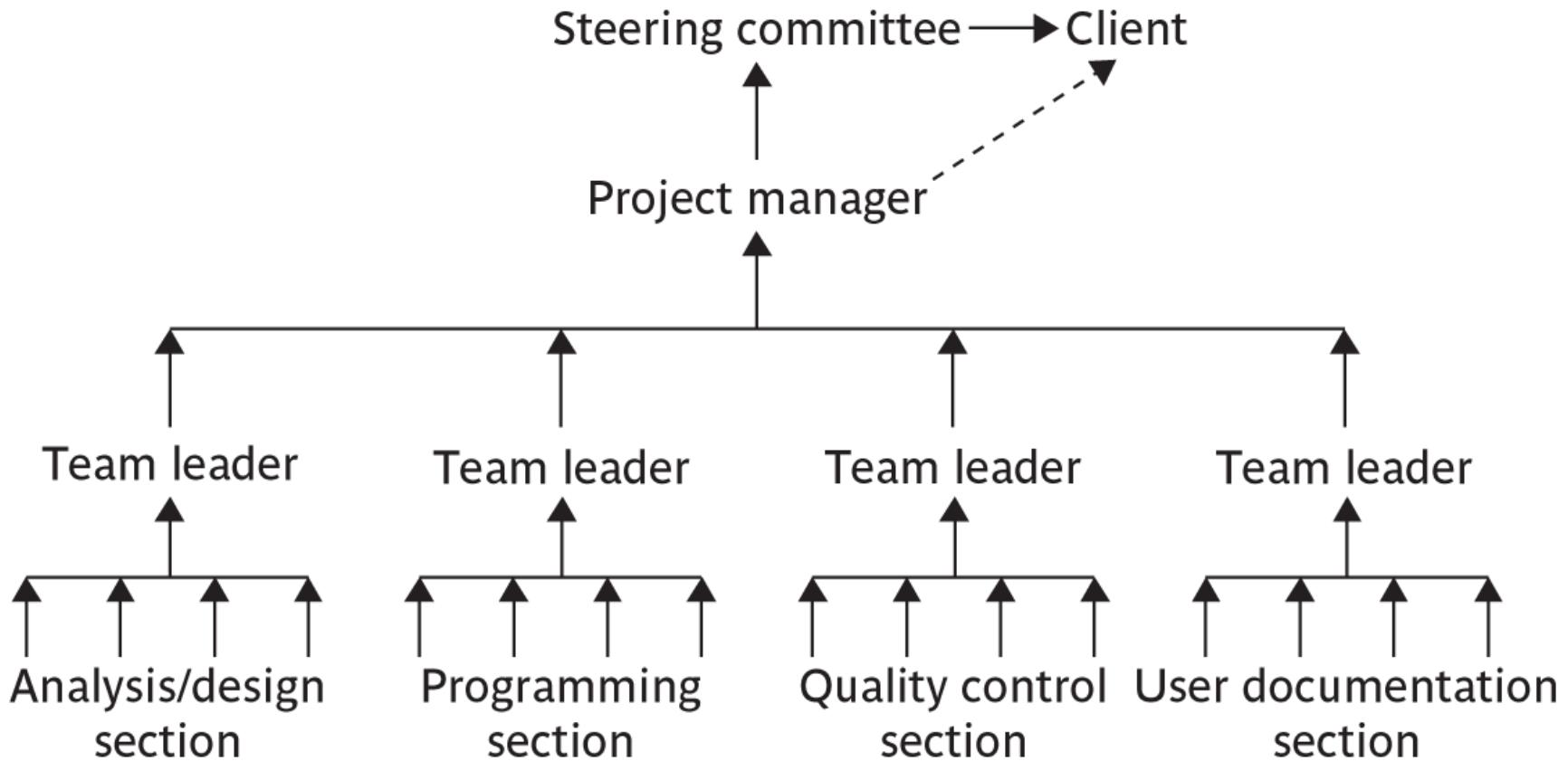
# **Project Monitoring & Earned Value Analysis**

**Source Courtesy:** Some of the contents of this PPT are sourced from materials provided by Publishers of T1 & T2

# The control cycle



# Responsibilities



# Assessing progress



Checkpoints – predetermined times when progress is checked

- Event driven: check takes place when a particular event has been achieved
- Time driven: date of the check is pre-determined

## Frequency of reporting

The higher the management level then generally the longer the gaps between checkpoints

# Collecting progress details

---

Need to collect data about:

- Achievements
- Costs

A big problem: how to deal with *partial completions*

*99% completion syndrome*

Possible solutions:

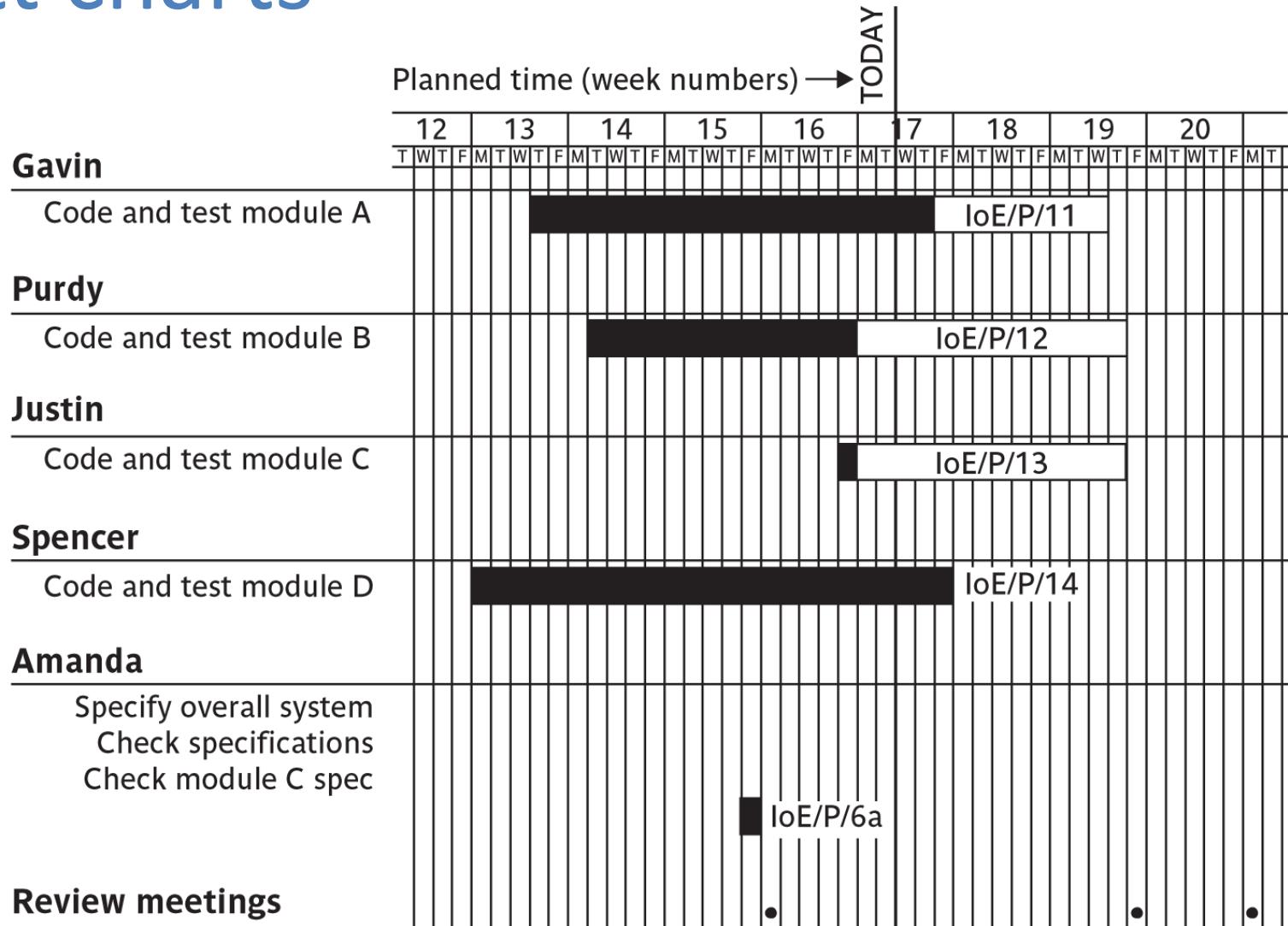
- Control of products, not activities
- Subdivide into lots of sub-activities

# Red/Amber/Green reporting

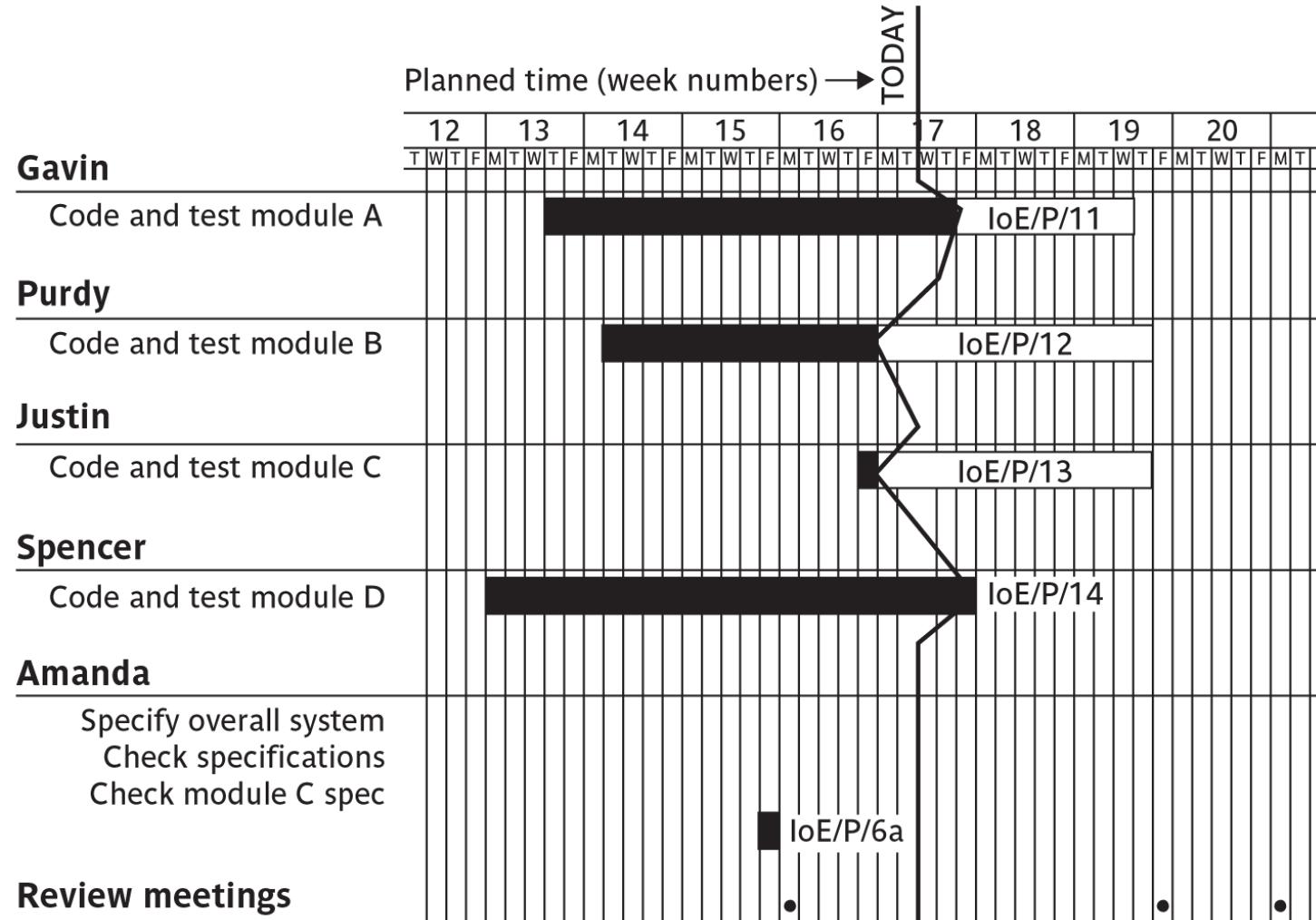
---

- Identify key tasks
- Break down into sub-tasks
- Assess subtasks as:
  - Green – ‘on target’
  - Amber – ‘not on target but recoverable’
  - Red – ‘not on target and recoverable only with difficulty’
- Status of ‘critical’ tasks is particularly important

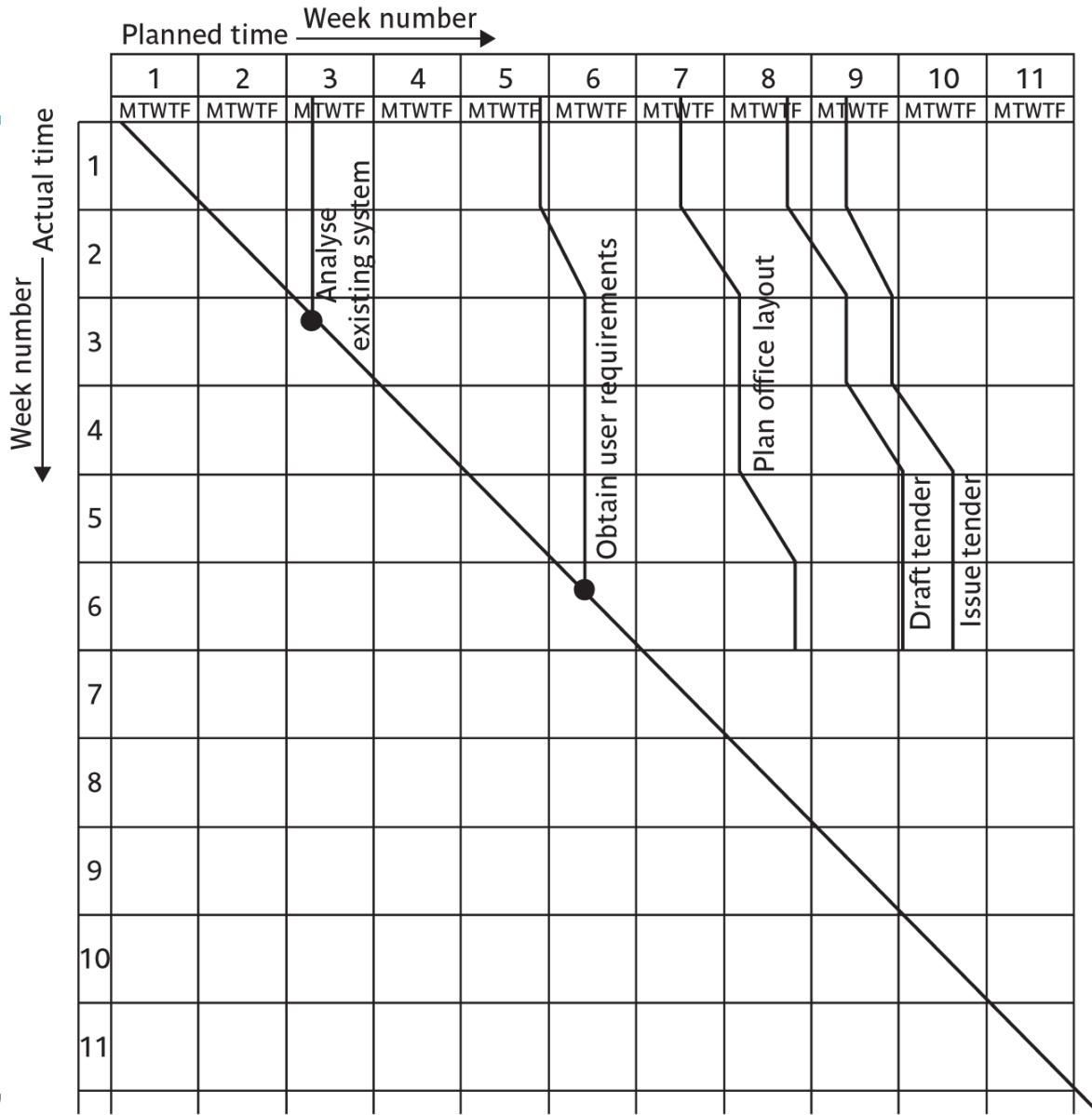
# Gantt charts



# Slip charts



# The timeline



# Cost monitoring

---

- A project could be late because the staff originally committed have not been deployed
- In this case the project will be *behind time* but *under budget*
- A project could be on time but only because additional resources have been added and so be *over budget*
- Need to monitor both achievements and costs

# Earned value analysis

- *Planned value (PV)* or *Budgeted cost of work scheduled (BCWS)* – original estimate of the effort/cost to complete a task (compare with idea of a ‘price’)
- *Earned value (EV)* or *Budgeted cost of work performed (BCWP)* – total of PVs for the work completed at this time

# Accounting conventions

---

- Work completed allocated on the basis
  - 50/50 half allocated at start, the other half on completion. These proportions can vary e.g. 0/100, 75/25 etc
  - *Milestone* current value depends on the milestones achieved
  - *Units processed*
- Can use money values, or staff effort as a surrogate

# Earned value – an example

- Tasks
  - Specify module 5 days
  - Code module 8 days
  - Test module 6 days
- At the beginning of day 20, PV = 19 days
- If everything but testing completed EV = 13 days
- Schedule variance = EV-PV i.e.  $13-19 = -6$
- Schedule performance indicator (SPI) =  $13/19 = 0.68$
- SV negative or SPI <1.00, project behind schedule

# Earned value analysis – actual cost

---

- Actual cost (AC) is also known as Actual cost of work performed (ACWP)
- In previous example, if
  - ‘Specify module’ actually took 3 days
  - ‘Code module’ actually took 4 days
- Actual cost = 7 days
- Cost variance (CV) = EV-AC i.e.  $13-7 = 6$  days
- Cost performance indicator =  $13/7 = 1.86$
- Positive CV or CPI  $> 1.00$  means project within budget

## Earned value analysis – actual costs

---

- CPI can be used to produce new cost estimate
- Budget at completion (BAC) – current budget allocated to total costs of project
- Estimate at completion (EAC) – updated estimate =  $BAC/CPI$ 
  - e.g. say budget at completion is £19,000 and CPI is 1.86
  - $EAC = BAC/CPI = £10,215$  (projected costs reduced because work being completed in less time)

# Time variance

---

- Time variance (TV) – difference between time when specified EV should have been reached and time it actually was
- For example say an EV of £19000 was supposed to have been reached on 1<sup>st</sup> April and it was actually reached on 1<sup>st</sup> July then TV = - 3 months

# Earned value chart with revised forecasts

## Activity Assessment Sheet

Staff Justin

Ref: IoE/P/13

Activity: Code and test module C

Week number	13	14	15	16	17	18	
Activity summary	G	A	A	R			

Component							Comments
Screen handling procedures	G	A	A	G			
File update procedures	G	G	R	A			
Housekeeping procedures	G	G	G	A			
Compilation	G	G	G	R			
Test data runs	G	G	G	A			
Program documentation	G	G	A	R			

# Prioritizing monitoring

We might focus more on monitoring certain types of activity e.g.

- Critical path activities
- Activities with no free float – if delayed later dependent activities are delayed
- Activities with less than a specified float
- High risk activities
- Activities using critical resources

# Getting back on track: options

---

- Renegotiate the deadline – if not possible then
- Try to shorten critical path e.g.
  - Work overtime
  - Re-allocate staff from less pressing work
  - Buy in more staff
- Reconsider activity dependencies
  - Over-lap the activities so that the start of one activity does not have to wait for completion of another
  - Split activities

# Exception planning



- Changes that could affect the Project
  - Users
  - The business case (e.g. costs increase reducing the potential profits of delivered software product)
  - These changes could be to
    - Delivery date
    - Scope
    - Cost
- In these cases an **exception report** is needed

# Exception planning - continued

---

- First stage
  - Write an **exception report** for sponsors (perhaps through project board)
    - Explaining problems
    - Setting out options for resolution
- Second stage
  - Sponsor selects an option ( or identifies another option)
  - Project manager produces an **exception plan** implementing selected option
  - Exception plan is reviewed and accepted/rejected by sponsors/Project Board

# Project/Cost Monitoring using EVM: Summary

---

- Applicable for Large Projects with Budgeted Cost (using 'waterfall model' of development)
- Realistic impact assessment of overruns (in \$)
- CFO friendly

*Please do some simple exercises given in the text-book*

# Thank You



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

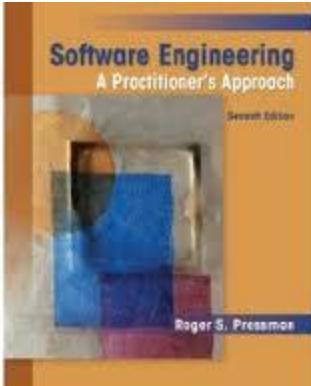
# SS ZG622:

## Software Project Management

### (Managing Teams, Motivation & Leadership)

Prof K G Krishna, BITS-Pilani Off-campus centre, Hyderabad

# Text Books



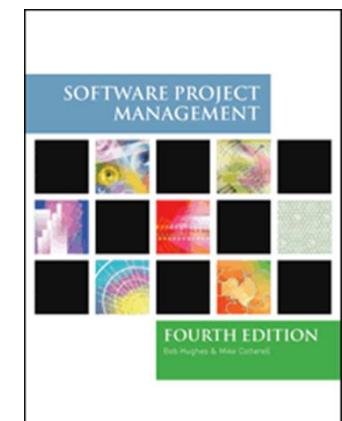
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# Managing Teams

[Motivation, Formation of Teams, Work-Life Balance, ...]

Ref: T2 / Chap-11,12

**Source Courtesy:** Some of the contents of this PPT are sourced from materials provided by Publishers of T1 & T2

# Topics

---

- Selection of Team Members
- Motivation & Job-satisfaction
- Team Formation
- Consensus Making
- Teams in Agile Methods
- Virtual Teams
- Leadership Styles
- Health, Safety and Environment
- Ethical code-of-conduct

# Taylor's Scientific Management

---

- To select the best people for the job
- To instruct them in the best methods
- To give financial incentives as per the work measured
- The problem: 'group norms'

# Supervision, Control?

---

- Hawthorne's effect (1920) – series of experiments at the Hawthorne Plant of Western Electric, Chicago
  - Found that simply showing an interest in a group increased productivity
  - Theory X: there is a need for coercion, direction, and control of people at work
  - Theory Y: work is as natural as rest or play

# Selecting the best people

---

- Distinguish between **eligible** (having the right qualifications) and **suitable** candidates (can do the job).
- The danger is employ someone who is eligible but not suitable
- The best situation is to employ someone who is suitable but not eligible! For example, these are likely to be cheaper and to stay in the job.

# Do good software developers have innate characteristics?

---

- 1968 study – difference of 1:25 in time taken by different programmers to code program
- Other research found experience better than maths skills as a guide to software skills
- Some research suggested software developers less sociable than other workers
- Later surveys have found no significant social differences between IT workers and others – this could be result of broader role of IT in organizations

# Typical selection process

---

## 1. Create a job specification.

Content includes types of task to be carried out.

## 2. Create a job holder profile

Describes the characteristics of the person who could do the job

## 3. Obtain applicants

Identify the media that potential job holders are likely to consult. Elicit CVs

## 4. Select potential candidates from CVs.

Do not waste everybody's time interviewing people whose CV clearly indicates are unsuitable.

## 5. Further selection, including interview

Selection processes could include aptitude tests, examination of work portfolios. Make sure selection processes map to the job holder profile

## 6. Other procedures.

e.g. taking up references, medicals etc

# Induction / Orientation / 'Entry Level Training Programs'

---

- The induction of new staff should be carefully planned – worst case where new recruit is simply ignored and not given any tasks
- Good induction leads to new recruit becoming productive more quickly
- Need to review staff progress frequently and provide feedback
- Need to identify training that could enhance staff effectiveness.

# Herzberg's Factors Affecting Job Satisfaction

---



1. *Hygiene or maintenance factors* – make you dissatisfied if they are not right e.g. pay, working conditions
2. *Motivators* – make you feel the job is worthwhile e.g. a sense of achievement

# Motivating for Performance

---

- Motivation can often make up for shortfalls in innate skills
- Taylor's approach - financial incentives
- Abraham Maslow (1908-1970)
  - motivations vary from individual to individual
  - hierarchy of needs – as lower ones fulfilled, higher ones emerge
  - Lowest level – food, shelter
  - Highest level – self-actualization

# Vroom's Motivating Influences

---

1. *Expectancy* – the belief that working harder leads to better performance
2. *Instrumentality* – the belief that better performance will be rewarded
3. *Perceived value* of the reward

# Making job meaningful (Oldham-Hackman)

---

- Making job meaningful
  - Skill variety
  - Task identity
  - Task significance
- Contributing to job satisfaction:
  - Autonomy
  - Feedback

# Methods to improve job satisfaction

---

- Set specific goals
- Provide feedback on the progress towards meeting those goals
- Consider job redesign
  - Job enlargement
  - Job enrichment

# Stress / Burnout

---

- Edward Yourdon quotes a project manager: '*Once a project gets rolling, you should be expecting members to be putting in at least 60 hours a week....The project manager must expect to put in as many hours as possible.*'
- 1960 study in US: people under 45 who worked more than 48 hours a week twice the risk of death from coronary heart disease.
- XP practice – maximum 40 hour working week

# Stress can be reduced by good project management

---



Good project management should lead to:

- Reasonable estimates of effort
- Good project control leading fewer unexpected crises
- Making clear what is expected of each team member – reduces **role ambiguity**
- Reduced **role conflict** where a person is torn between conflicting responsibilities

Bullying tactics are a symptom of incompetent project management.

# Health and safety

---

- Apart from stress, health and safety less likely to be an issue compared to other engineering projects.
- ...but sometimes IT infrastructure may be set up as other building work is going on
- Local laws insist that organizations employing over 5 staff should have a **written safety policy**
- Management of Health, Safety and Environment (HSE) should be embedded in project management.

# Ethical and professional concerns

---

- Ethics relates to the moral obligation to respect the rights and interests of others – goes beyond strictly legal responsibilities
- Three groups of responsibilities:
  - Responsibilities that everyone has
  - Responsibilities that people in organizations have
  - Responsibilities relating to your profession or calling
- Professionals have knowledge about the technical domain that the general public does not
  - Many professions, or would be professions, have codes of conduct for their members e.g.
  - <<http://www.bcs.org/upload/pdf/cop.pdf>>
  - <<http://www.ieee.org/web/aboutus/ethics>>
  - <[http://www.apm.org/about/se\\_code](http://www.apm.org/about/se_code)>

Next ... → Becoming a Team Member



# Becoming a team member...

---

- Five basic stages of development (Tuckman and Jensen):
  - Forming
  - Storming
  - Norming
  - Performing
  - Adjourning

# Balanced teams (Meredith Belbin)

---

- Putting the 'best' people together? – performs badly!
- The need for a balance of skills and management roles in a successful team

# Management team roles

---

- The co-ordinator – good at chairing meetings
- The ‘plant’ – an idea generator
- The monitor-evaluator – good at evaluating ideas
- The shaper – helps direct team’s efforts
- The team worker – skilled at creating a good working environment
- The resource investigator – adept at finding resources, including information
- The completer-finisher – concerned with getting tasks completed
- The implementer – a good team player who is willing to undertake less attractive tasks if they are needed for team success
- The specialist – the ‘techie’ who likes to acquire knowledge for its own sake

# Group performance

---

- Some tasks are better carried out collectively while other tasks are better delegated to individuals
  - *Additive tasks* – the effort of each participant is summed
  - *Compensatory tasks* – the judgements of individual group members are summed – errors of some compensated for by judgements of others
  - *Disjunctive tasks* – there is only one correct answer – someone must:
    - Come up with right answer
    - Persuade the other that they are right
  - *Conjunctive* – the task is only finished when all components have been completed

# ‘Social loafing’ in teams

---

- Tendency for some team participants to ‘coast’ and let others do the work
- Also tendency not to assist other team members who have problems
- Suggested counter-measures:
  - Make individual contributions identifiable
  - Consciously involve group members ( ‘loafer’ could in fact just be shy!)
  - Reward ‘team players’

# Team Decision Making

---

- Barriers to good team decisions
  - Inter-personal conflicts
  - Conflicts tend to be dampened by emergence of *group norms* – shared group opinions and attitudes
  - *Risky shift* – people in groups are more likely to make risky decisions than they would as individuals

# Delphi approach for consensus

---

- To avoid dominant personalities intruding the following approach is adopted
  1. Enlist co-operation of experts
  2. Moderator presents experts with problem
  3. Experts send in their recommendations to the moderator
  4. Recommendations are collated and circulated to all experts
  5. Experts comment on ideas of others and modify their own recommendation if so moved
  6. If moderator detects a consensus, stop; else back to 4

# Team 'heedfulness'

---

- Where group members are aware of the activities of other members that contribute to overall group success
- Impression of a 'collective mind'
- Some attempts to promote this:
  - Egoless programming
  - Chief programmer teams
  - XP
  - Scrum

# Egoless programming (Gerry Weinberg)

---

- Tendency for programmers to be protective of their code and to resist perceived criticisms by others of the code
- Encourage programmers to read each others code
- Software should become communal, not personal – hence ‘egoless programming’

# Extreme programming (XP)

---

- XP reduce the length of communication paths (the time between something being recorded and it being used)
- Software code enhanced to be self-documenting
- Software regularly refactored to clarify its structure
- Test cases/expected results created *before* coding – acts as a supplementary specification
- Pair programming – a development of the co-pilot concept

# Scrum

---

- Named as an analogy to a rugby scrum – all pushing together
- Originally designed for new product development where ‘time-to-market’ is important
- ‘Sprints’ increments of typically one to four weeks
- Daily ‘scrums’ – daily stand-up meetings of about 15 minutes
- Unlike XP, requirements are frozen during a sprint
- At the beginning of the sprint there is a sprint planning meeting where requirements are prioritized
- At end of sprint, a review meeting where work is reviewed and requirements may be changed or added to

# Team Co-ordination of dependencies

---

- Co-ordination theory has identified the following types of coordination between teams and other units:
  - *Shared resources*. e.g. where several projects need the services of scarce technical experts for certain parts of the project.
  - *Producer-customer ('right time') relationships*. A project activity may depend on a product being delivered first.
  - *Task-subtask dependencies*. In order to complete a task a sequence of subtasks have to be carried out.
  - *Accessibility ('right place')* dependencies. This type of dependency is of more relevance to activities that require movement over a large geographical area, but arranging the delivery and installation of IT equipment might be identified as such.
  - *Usability ('right thing')* dependencies. Broader concern than the design of user interfaces: relates to the general question of fitness for purpose, e.g. the satisfaction of business requirements.
  - *Fit requirements*. This is ensuring that different system components work together effectively.

Next... → Virtual Projects, Communication  
& Leadership Styles



# ‘Virtual Projects’?

---

The physical needs of software developers (according to an IBM report):

- 100 square feet of floor space
- 30 square feet of work surface
- Dividers at least 6 feet high to muffle noise
- Demarco and Lister found clear statistical links between noise and coding error rates
- One answer: Teleworking ('work from home')

# Virtual Working: Possible advantages

---

- Can use staff from developing countries – lower costs
- Can use short term contracts:
  - Reduction in overheads related to use of premises
  - Reduction in staff costs, training, holidays, pensions etc.
- Can use specialist staff for specific jobs
- Productivity of home workers can be higher – fewer distractions
- Can take advantage of time zone differences e.g. overnight system testing

# Virtual Working: Challenges

---

- Work requirements have to be carefully specified
- Procedures need to be formally documented
- Co-ordination can be difficult
- Payment methods need to be modified – piece-rates or fixed price, rather than day-rates
- Possible lack of trust when there is no face-to-face contact
- Assessment of quality of delivered products needs to be rigorous
- Different time zones can cause communication and co-ordination problems

# Time/place constraints on communication

	Same place	Different place
Same time	Meetings, interviews	Telephone, Instant messaging
Different times	Notice boards Pigeon-holes	Email Voicemail Documents

# Best method of communication depends on stage of project



- **Early stages**
  - Need to build trust
  - Establishing context
  - Making important 'global' decisions
  - *Favours same time/ same place*
- **Intermediate stages**
  - Often involves the parallel detailed design of components
  - Need for clarification of interfaces etc
  - *Favours same time/different place*
- **Implementation stages**
  - Design is relatively clear
  - Domain and context familiar
  - Small amounts of operational data need to be exchanged
  - Favours different time/different place communications e.g. e-mail
- ***Face to face co-ordination meetings – the 'heartbeat' of the project***

# Communications plans

- As we have seen choosing the right communication methods is crucial in a project
- Therefore, a good idea to create a **communication plan**
- **Stages** of creating a communication plan
  - Identify all the major stakeholders for the project – see chapter 1
  - Create a plan for the project
  - Identify stakeholder and communication needs for each stage of the project
  - Document in a communication plan

# Content of a communication plan

For each communication event and channel, identify:

- *What*. This contains the name of a particular communication event, e.g. 'kick-off meeting', or channel, e.g. 'project intranet site'.
- *Who/target*. The target audience for the communication.
- *Purpose*. What the communication is to achieve.
- *When/frequency*. If the communication is by means of a single event, then a date can be supplied. If the event is a recurring one, such as a progress meeting then the frequency should be indicated.
- *Type/method*. The nature of the communication, e.g., a meeting or a distributed document.
- *Responsibility*. The person who initiates the communication.

# Leadership: types of authority

---

## *Position power*

- Coercive power – able to threaten punishment
- Connection power – have access to those who do have power
- Legitimate power – based on a person's title conferring a special status
- Reward power – able to reward those who comply

# Leadership: types of power

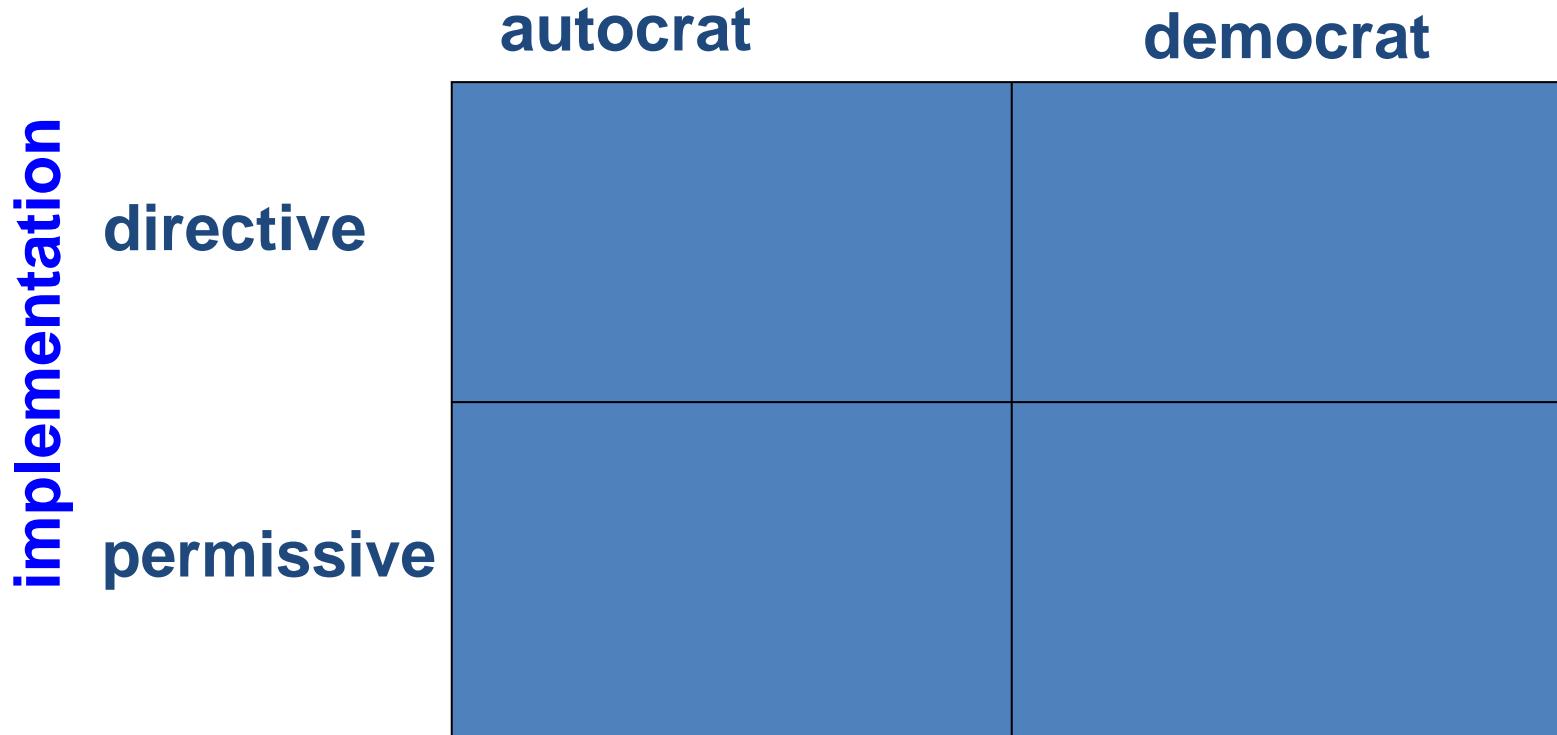
---

## *Personal power*

- *Expert power*: holder can carry out specialist tasks that are in demand
- *Information power*: holder has access to needed information
- *Referent power*: based on personal attractiveness or charisma

# Leadership styles

decision-making



# Leadership styles

---

- Task orientation – focus on the work in hand
- People orientation – focus on relationships
- Where there is uncertainty about the way job is to be done or staff are inexperienced they welcome task oriented supervision
- Uncertainty is reduced – people orientation more important
- Risk that with reduction of uncertainty, managers have time on their hands and become more task oriented (interfering)

---

# Thank You

## Any Questions?



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# **Software Project Management – Caselet 5- Miscellaneous challenges**

**S Subramanian ( Subbu)**

# Caselet 5 (Key Resources) context



- How to handle when key/critical resources leave the project during critical phase of a software development project.

# Suggested solutions

---

- Maintain good process and documentation
  - Create a shadow resource for critical resources
  - Have buffer resources, if cost permits
  - Develop good network among PM community
    - Expertise on demand
  - Hire contractors on emergency
  - Escalate when required
  - Keep client informed ( no one likes surprises)
-

---

# Stakeholder Management

# Stakeholder management

---

- The Project Management team has a professional responsibility to its stakeholders including customers, the performing organization and the public.
- Project Stakeholder Management includes the processes required to identify the people, groups, or organizations that could impact or be impacted by the project, to analyze stakeholder expectations and their impact on the project, and to develop appropriate management strategies for effectively engaging stakeholders in project decisions and execution.

# Identifying the Players (Stakeholders)



- Key point: your chances of success in a extreme project are often zero if you don't know who the key players are
- Also crucial that everyone on the project understands this – not just the project manager
- Typical players:
  - Owner: who pays for, or “accepts” the system?
  - Customer: who will actually use the system?
  - Shareholder: one of many “co-owners”
  - Stakeholder: someone whose interests are affected by the success/failure of the project, and who can affect the outcome of the project – and who may thus become an ally or obstacle to project success.
- The "loser user"

*Note: these roles may not be obvious, and they may shift during the course of the project*

# Some tips in Stakeholder Management – Communication

---

- It is widely agreed that communication comprises 90% of project management.
- We believe how communications are delivered, the medium, tone, and expression, is just as, if not more crucial than what is being communicated
- Some Key Principles
  - Deal with facts, not opinions.
  - Summarise the detail for appropriate levels of management.
  - Keep it timely, accurate and of a high quality.
  - Follow a pattern – get people accustomed to your updates.
  - Present program/project impacts and alternatives to key stakeholders, not just 'here are the issues'.
  - Don't focus on blame if things go wrong, focus on solutions, that is, the options analysed and the recommendation





**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad



# **Software Project Management – Caselet 2- Managing Distributed Development ( Multisite).**

**S Subramanian ( Subbu)**

# Caselet – Scenario

---

- Your company is requesting you ( as a PM) to develop software in a Distributed ( multisite) Environment
  - The sites ( teams) could be a combination of offshore and onshore locations
  - Outsourcing development may involve other organizations too.
  - To reduce development costs
  - Access to most talented resources across sites
  - Reduce time to market
-

# Challenges in Distributed Software Development (DSD)

---

- Pain Points
  - Misunderstood processes or mismatched processes across teams
  - Cultural issues.
  - Project Management issues – status reporting, control, collaboration/sharing, communication, trust among teams
  - Political issue
  - Concerns with security and IP protection,
  - Infrastructures and development tools

# Critical success factors for DSD

## Success Factors for Global Development and Delivery



# Summary

---

- Collaboration Tools and Environments
- Resource optimization – go /use where talent is available
- Increase productivity
- Use standards based development
- Security and privacy





**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# **Software Project Management – Caselet 4- User Acceptance Testing (UAT) Completion**

**S Subramanian ( Subbu)**

# Caselet (UAT Management) context



- Often as a PM you will find the following test cycles prolonging too long:
  - System Integration Testing (SIT)
  - User Acceptance testing ( UAT)
  - Warranty Support
- Client acceptance of the delivered system gets extended.
- Result in schedule overruns

# Define exit - SIT

---

- SIT Exit
    - Done by Development team.
    - Done in development environment
    - Define exit criteria agreed with client
      - No. of open errors in each of the severity { may be 1(high) to 4 (low)}
    - Agree on test cases
    - Submit results to client
  - Successful completion- ready for UAT
-

# Define exit - UAT

- **UAT Exit ( with Beta phase)**
  - Pre UAT – Beta 1 - is strictly for IT personnel (client) who support the various departments & Beta 2 – end users from each of those departments. Preferably, select two users from each department
  - Beta 1 & Beta 2 are led by the service provider under development environment
- **UAT Exit at client environment**
  - Done by client with vendor support
  - Done at client environment
  - Define exit criteria agreed with client
    - No. of open errors in each of the severity { may be 1(high) to 4 (low)}
  - Agree on test cases
  - Submit results to client
- **ON Successful completion, system goes to Production- ready for Warranty phase**

# Warranty support & exit

---

- Determine who handles warranty support
    - Project Development team or Production support team ( of Vendor)?
  - Define Scope of Warranty
  - Classify issues reported - CR or Defect
  - Define exit criteria from Warranty period
  - Define handover process to Production support team
-

# Summary

---

- Make sure exit criteria is defined for all test phases.
  - Define scope SIT/UAT environment
  - Involve business users during UAT
  - Plan for UAT & Warranty support by development team
-





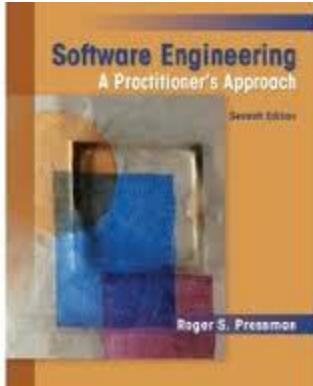
**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

# SS ZG622: Software Project Management (Quality Management Systems)

Prof K G Krishna, BITS-Pilani Off-campus Centre, Hyderabad

# Text Books



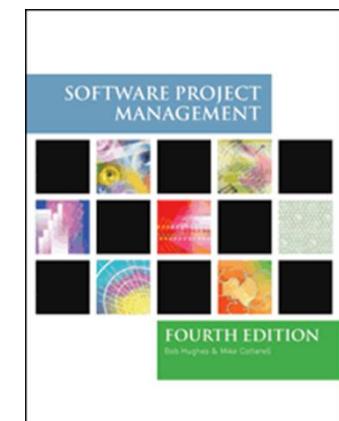
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Quality Management Systems**

## **(ISO 9001 / CMMI Models)**

Ref: T2

---

# *Overview of* ISO 9001:2000 - CMMI (SQA Model)

Source Ref: T2-Chap 8

# ISO 9001:2000 and quality management systems

---



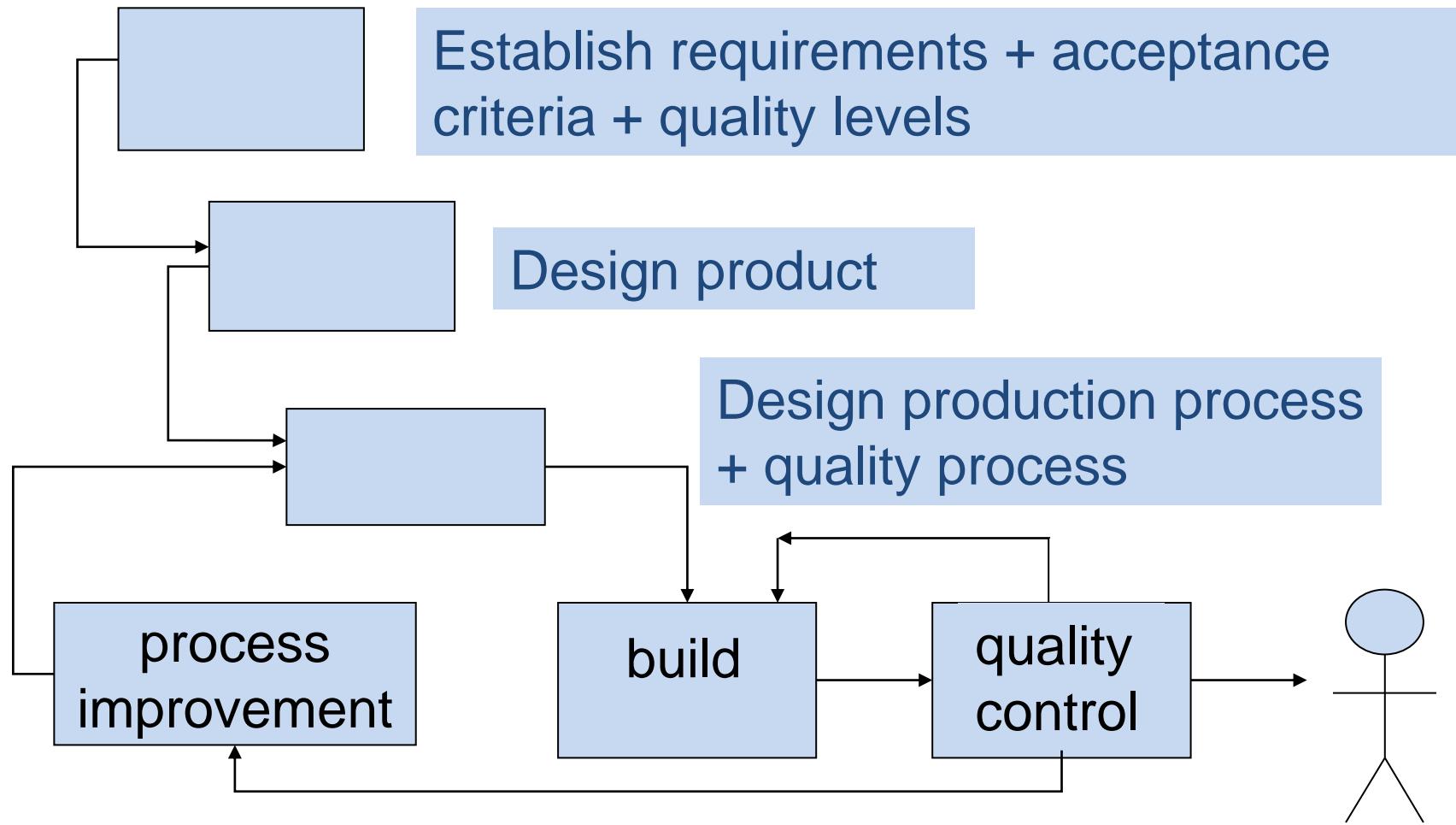
- ISO 9001 is one of a family of standards that specify the characteristics of a good quality management system (QMS)
- Can be applied to the creation of any type of product or service, not just IT and software
- Does NOT set universal product/service standards
- DOES specify the way in which standards are established and monitored

# ISO 9001:2000 principles

---

1. Focus on interrelation of processes that deliver products and services
2. Continuous process improvement
3. Decision-making based on factual evidence
4. Mutually beneficial relationships with suppliers
5. Focus on interrelation of processes that deliver products and services
6. Continuous process improvement
7. Decision-making based on factual evidence
8. Mutually beneficial relationships with suppliers

# ISO 9001:2000 cycle



# Capability maturity model

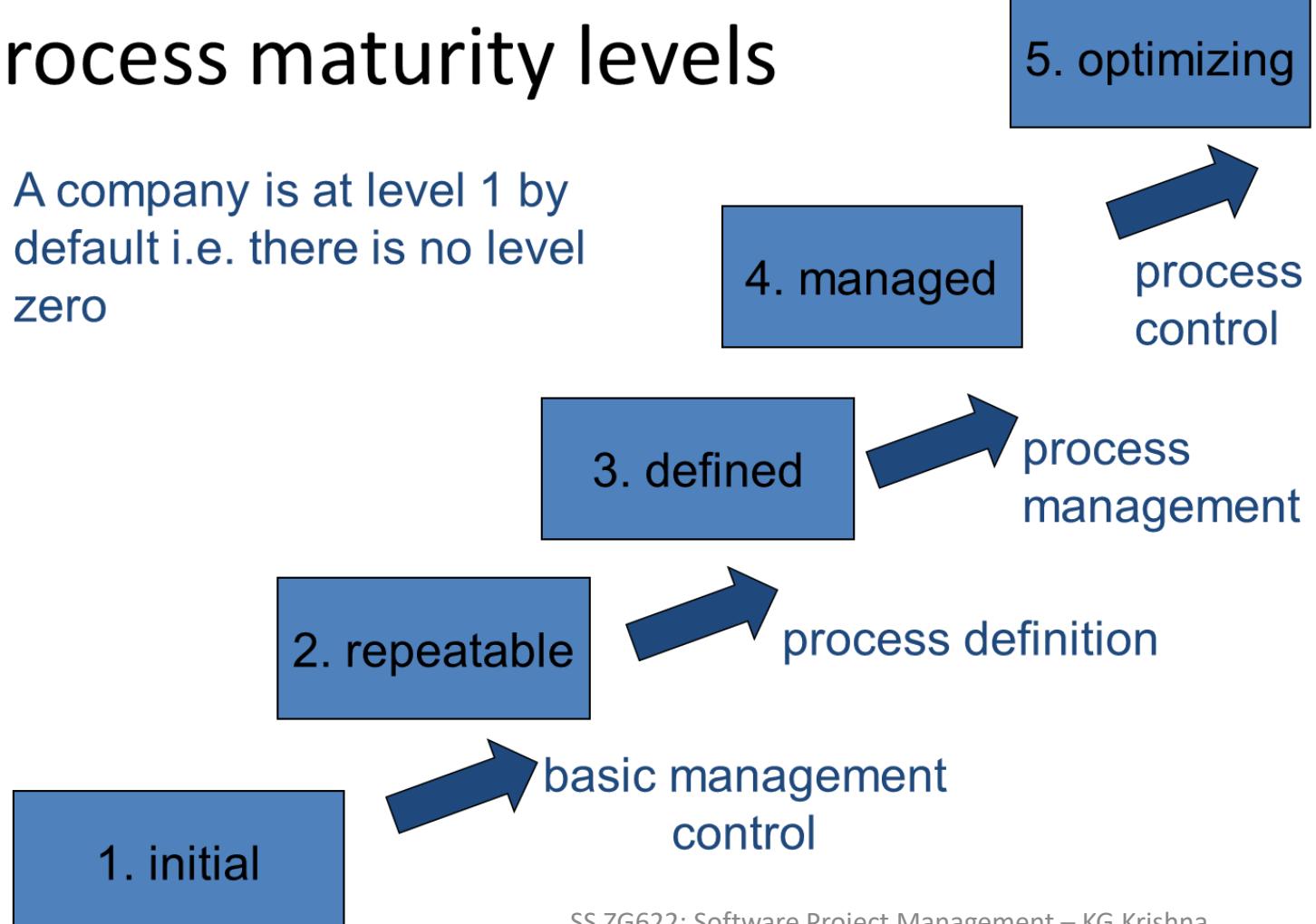
---

- created by Software Engineering Institute, Carnegie Mellon University
- CMM developed by SEI for US government to help procurement
- Watts S. Humphrey 'Managing the software process' Addison Wesley
- Assessment is by questionnaire and interview
- Different versions have been developed for different environments e.g. software engineering
- New version CMMI tries to set up a generic model which can be populated differently for different environments

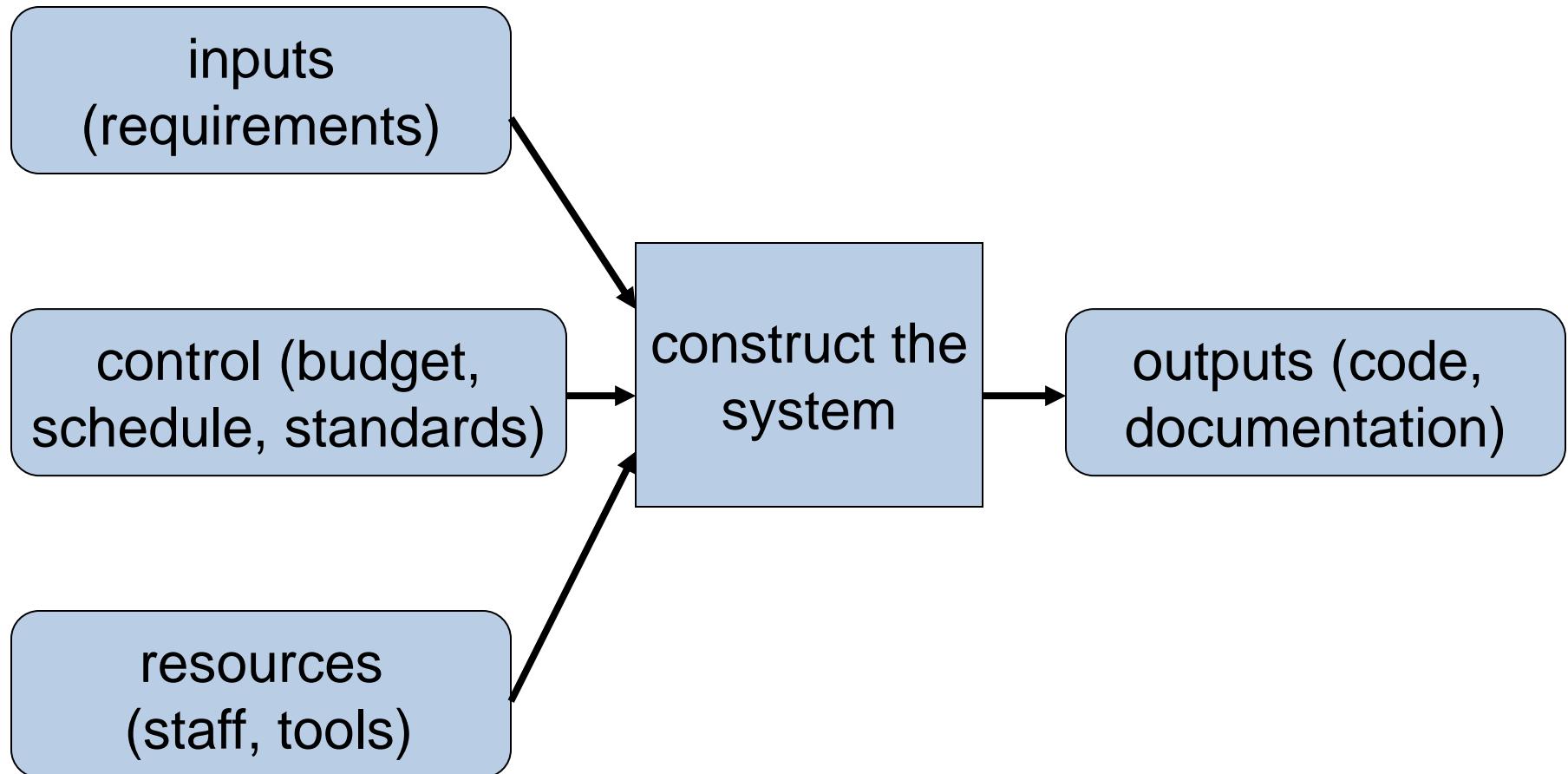
# CMMI Process maturity levels

## Process maturity levels

A company is at level 1 by default i.e. there is no level zero



# A repeatable model



# Repeatable model KPAs

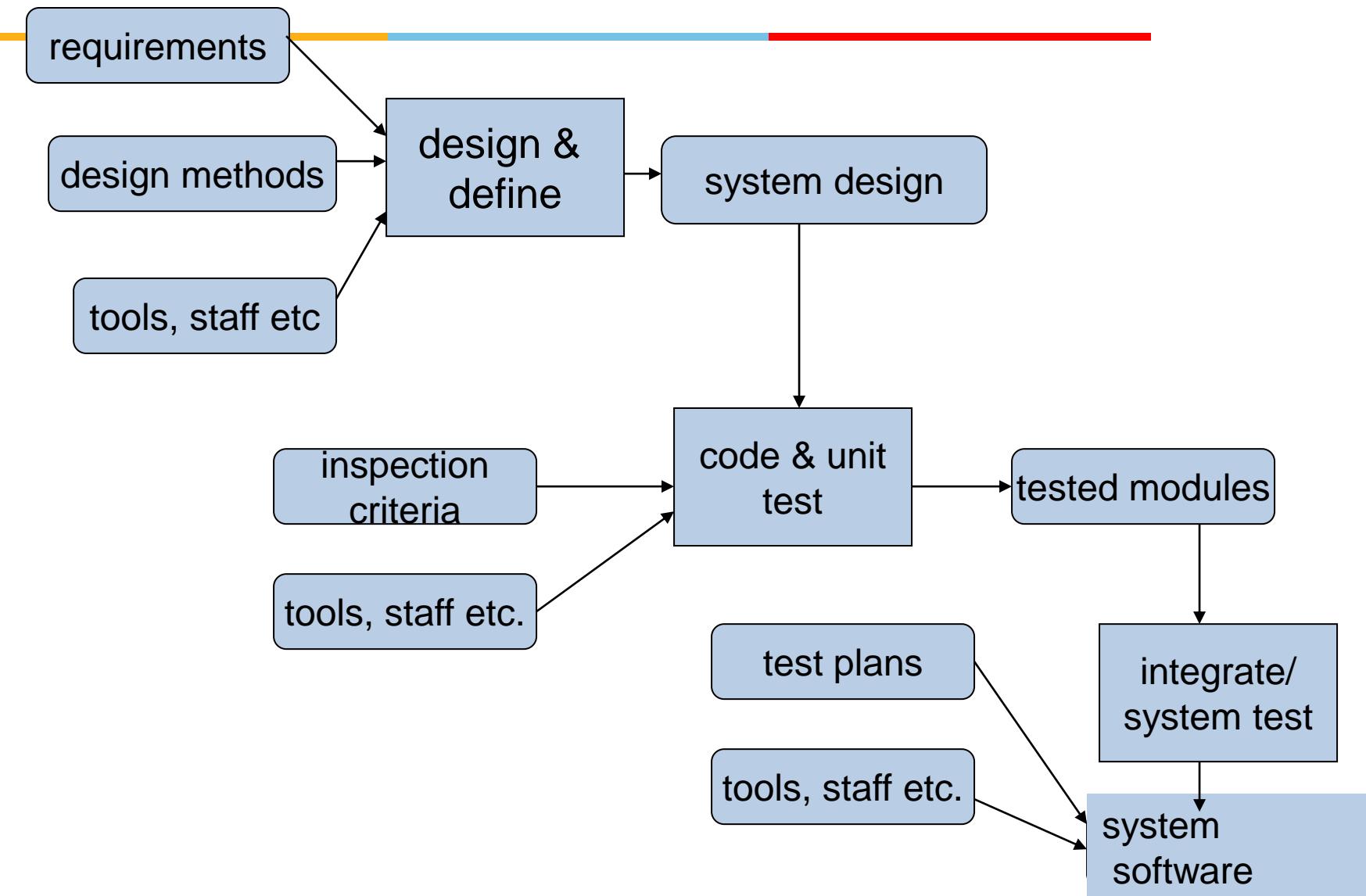


---

To move to this level concentrate on:

- Configuration management
- Quality assurance
- Sub-contract management
- Project planning
- Project tracking and oversight
- Measurement and analysis

# A defined process

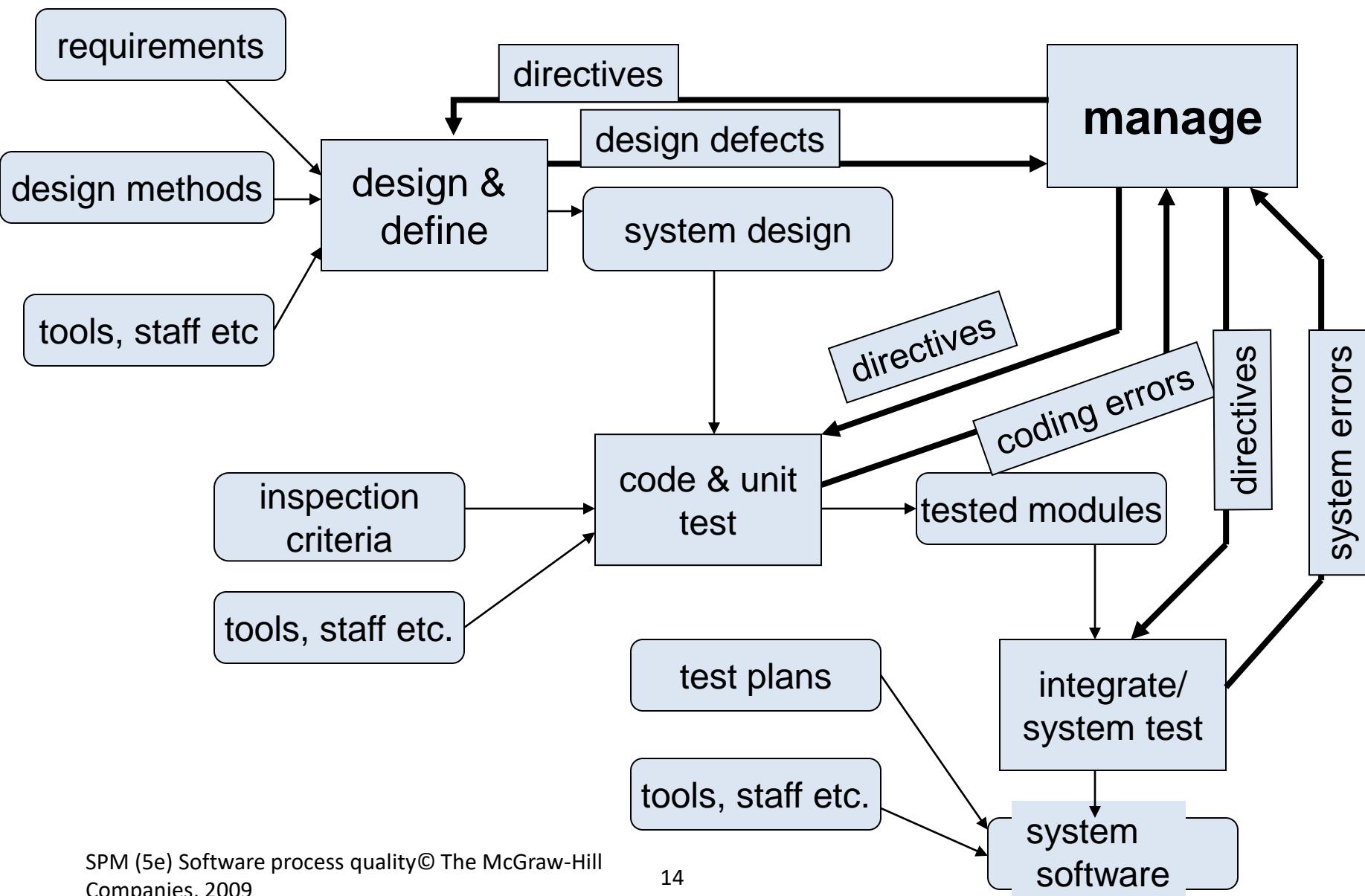


# Repeatable to defined KPAs

---

- Requirements development and technical solution
- Verification and validation
- Product integration
- Risk management
- Organizational training
- Organizational process focus (function)
- Decision analysis and resolution
- Process definition
- Integrated project management

# a managed process



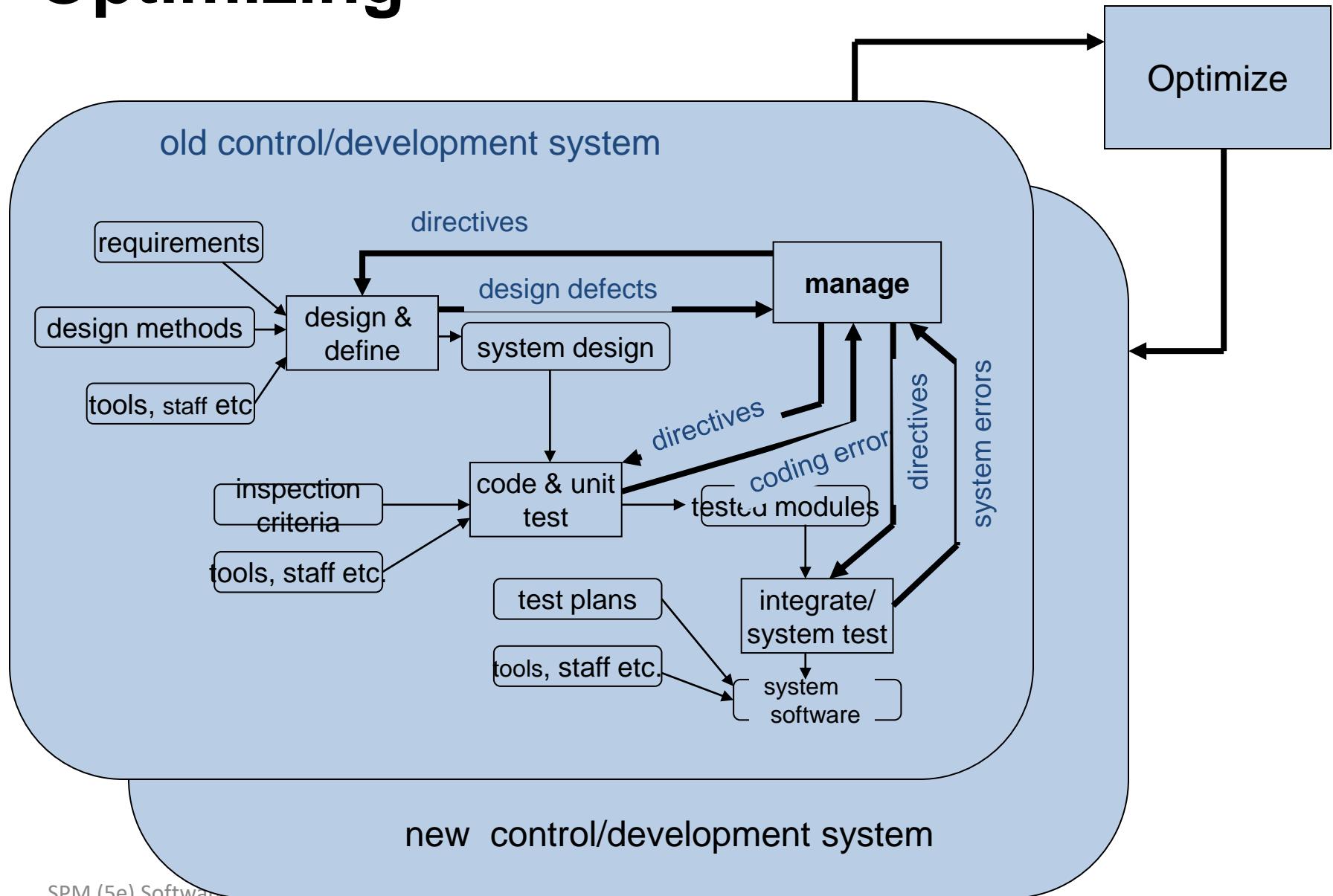
# Defined to managed KPAs

---

Focus on:

- Organizational process performance
- Quantitative project management

# Optimizing



# Managing to optimizing: KPAs

---

Focus on:

- Causal analysis and resolution
- Organizational innovation and deployment

# Some questions about CMMI

---

- suitable only for large organizations?
  - e.g. need for special quality assurance and process improvement groups
- defining processes may not be easy with new technology
  - how can we plan when we've not used the development method before?
- higher CMM levels easier with maintenance environments?
- can you jump levels?

---

# Thank You

## Any Questions?



**BITS Pilani**

Pilani|Dubai|Goa|Hyderabad

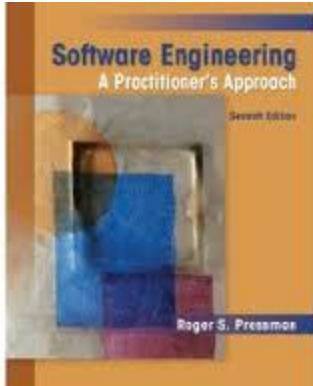
# SS ZG622:

# Software Project Management

## (Types of Contracts / Business Models)

Prof K G Krishna, BITS-Pilani Off-campus centre, Hyderabad

# Text Books



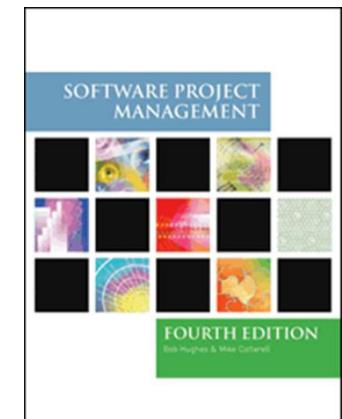
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# **Contracts / Business Models, Managing Contracts**

# Acquiring software from external supplier

---



This could be:

- a *bespoke system* - created specially for the customer
- *off-the-shelf* - bought 'as is'
- *customised off-the-shelf (COTS)* - a core system is customised to meet needs of a particular customer

# Types of contract

---

- fixed price contracts (FPP)
- time and materials (T&M) contracts
- fixed price per delivered unit

→ Note difference between products and services

Often license to use software product is bought rather than the product itself (ex: ERP products)

# Fixed price contracts

---

## Advantages to customer

- known expenditure
- supplier motivated to be cost-effective

# Fixed price contracts

---

## Disadvantages

- supplier will increase price to meet contingencies
- difficult to modify requirements
- cost of changes likely to be higher
- threat to system quality

# Time and materials

---

## Advantages to customer

- easy to change requirements
- lack of price pressure can assist product quality

# Time and materials

---

## Disadvantages

- Customer liability - the customer absorbs all the risk associated with poorly defined or changing requirements
- Lack of incentive for supplier to be cost-effective

# Fixed price per unit delivered

<i>FP count</i>	<i>Design cost/FP</i>	<i>implementation cost/FP</i>	<i>total cost/FP</i>
to 2,000	\$242	\$725	\$967
2,001- 2,500	\$255	\$764	\$1,019
2,501- 3,000	\$265	\$793	\$1,058
3,001- 3,500	\$274	\$820	\$1,094
3,501- 4,000	\$284	\$850	\$1,134

# Fixed price/unit example

---

- Estimated system size 2,600 FPs
- Price
  - 2000 FPs x \$967 *plus*
  - 500 FPs x \$1,019 *plus*
  - 100 FPs x \$1,058
  - i.e. \$2,549,300
- What would be charge for 3,200 FPs?

# Fixed price/unit

---

## Advantages for customer

- customer understanding of how price is calculated
- comparability between different pricing schedules
- emerging functionality can be accounted for
- supplier incentive to be cost-effective

# Fixed price/unit

---

## Disadvantages

- difficulties with software size measurement - may need independent FP counter
- changing (as opposed to new) requirements: how do you charge?

# The tendering process

---

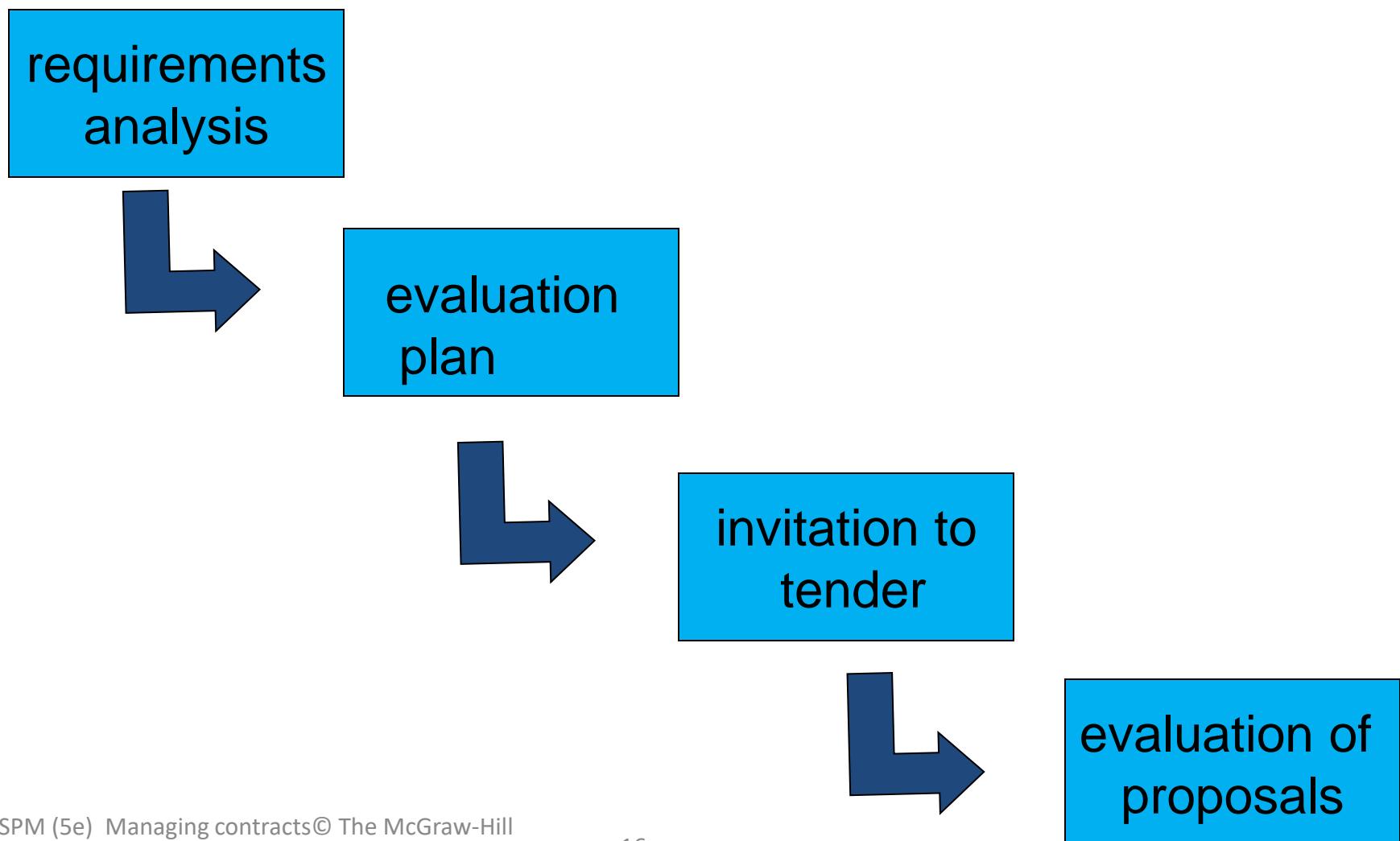
- Open tendering
  - any supplier can bid in response to the *invitation to tender*
  - all tenders must be evaluated in the same way
  - government bodies may have to do this by local/international law (including EU and WTO, World Trade Organization, requirements)

# The tendering process

---

- Restricted tendering process
  - bids only from those specifically invited
  - can reduce suppliers being considered at any stage
- Negotiated procedure
  - negotiate with one supplier e.g. for extensions to software already supplied

# Stages in contract placement



# Requirements document: structure

---

- introduction
- description of existing system and current environment
- future strategy or plans
- system requirements -
  - mandatory/desirable features
- deadlines
- additional information required from bidders

# Requirements

---

- These will include
  - functions in software, with necessary inputs and outputs
  - standards to be adhered to
  - other applications with which software is to be compatible
  - quality requirements e.g. response times

# Evaluation plan

---

- How are proposals to be evaluated?
- Methods could include:
  - reading proposals
  - interviews
  - demonstrations
  - site visits
  - practical tests

# Evaluation plan -contd.

- Need to assess value for money (VFM) for each desirable feature
- VFM approach an improvement on previous emphasis on accepting lowest bid
- Example:
  - feeder file saves data input
  - 4 hours work a month saved at £20 an hour
  - system to be used for 4 years
  - if cost of feature £1000, would it be worth it?

# Invitation to tender (ITT)

---

- Note that bidder is making an *offer* in response to ITT
- *acceptance* of offer creates a *contract*
- Customer may need further information
- Problem of different technical solutions to the same problem

# Memoranda of agreement (MoA)

- Customer asks for technical proposals
- Technical proposals are examined and discussed
- Agreed technical solution in MoA
- Tenders are then requested from suppliers based in MoA
- Tenders judged on price
- Fee could be paid for technical proposals by customer

# Contracts

---

- A project manager cannot be expected to be a legal expert – needs advice
- BUT must ensure contract reflect true requirements and expectations of supplier and client

# Contract checklist

---

- Definitions – what words mean precisely e.g. ‘supplier’, ‘user’, ‘application’
- Form of agreement. For example, is this a contract for a sale or a lease, or a license to use a software application? Can the license be transferred?
- Goods and services to be supplied – this could include lengthy specifications
- Timetable of activities
- Payment arrangements – payments may be tied to completion of specific tasks

# Contract checklist - continued

---

- Ownership of software
  - Can client sell software to others?
  - Can supplier sell software to others? Could specify that customer has 'exclusive use'
  - Does supplier retain the copyright?
  - Where supplier retains source code, may be a problem if supplier goes out of business; to circumvent a copy of code could be deposited with an **escrow** service

# Contract checklist - continued

---

- Environment – for example, where equipment is to be installed, who is responsible for various aspects of site preparation e.g. electricity supply?
- Customer commitments – for example providing access, supplying information
- Standards to be met

# Contract management

---

Some terms of contract will relate to management of contract, for example,

- Progress reporting
- Decision points – could be linked to release of payments to the contractor
- Variations to the contract, i.e. how are changes to requirements dealt with?
- Acceptance criteria

# How do we evaluate the following?

---

- usability of an existing package
- usability of an application yet to be built
- maintenance costs of hardware
- time taken to respond to requests for software support
- training

# Contract management

---

- Contracts should include agreement about how customer/supplier relationship is to be managed e.g.
  - *decision points* - could be linked to payment
  - *quality reviews*
  - *changes to requirements*

# Other Types of Contracts

---

- Risk-Reward Share
- Partnership

---

# Thank You



**BITS Pilani**  
Pilani|Dubai|Goa|Hyderabad

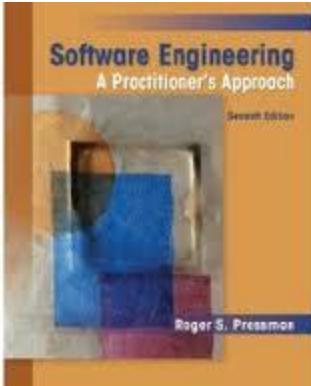
# SS ZG622:

## Software Project Management

### (Project Evaluation & Program Management)

Prof K G Krishna, BITS-Pilani Off-Campus Centre, Hyderabad

# Text Books



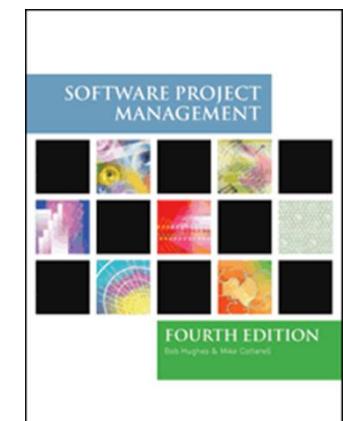
**T1:** Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, TMH, 2010

**T2:** Hughes, B and Cotterel, M., Software Project Management, 11th Edition, TMH, 2011

**S1:** Frank Tsui, Managing Software Projects, Jones&Bartlett, 2011

**S2:** Pankaj Jalote, Software Project Management in Practice, Pearson Education, 2002

**Note:** In order to broaden understanding of concepts as applied to Indian IT industry, students are advised to refer books of their choice and case-studies in their own organizations





# Project Evaluation & Program Management

# Evaluation of Projects & Programs

---

- The business case for a project
- Project portfolios
- Project evaluation
  - Cost benefit analysis
  - Cash flow forecasting
  - Decision Tree
- Programme management
- Benefits management

# The business case

---

- **Feasibility studies** can also act as a ‘business case’
- Provides a justification for starting the project
- Should show that the benefits of the project will exceed development, implementation and operational costs
- Needs to take account of business risks

# Contents of a business case

---

1. Introduction/ background
2. The proposed project
3. The market
4. Organizational and operational infrastructure
5. The benefits
6. Outline implementation plan
7. Costs
8. The financial case
9. Risks
10. Management plan

# Content of the business case

---

- **Introduction/background:** describes a problem to be solved or an opportunity to be exploited
- **The proposed project:** a brief outline of the project scope
- **The market:** the project could be to develop a new product (e.g. a new computer game). The likely demand for the product would need to be assessed.

# Content of the business case - continued

---

- **Organizational and operational infrastructure:** How the organization would need to change. This would be important where a new information system application was being introduced.
- **Benefits** These should be express in financial terms where possible. In the end it is up to the client to assess these – as they are going to pay for the project.

# Content of the business case - continued

---

- **Outline implementation plan:** how the project is going to be implemented. This should consider the disruption to an organization that a project might cause.
- **Costs:** the implementation plan will supply information to establish these
- **Financial analysis:** combines costs and benefit data to establish value of project

# Project portfolio management

---

The concerns of project portfolio management include:

- Evaluating proposals for projects
- Assessing the risk involved with projects
- Deciding how to share resources between projects
- Taking account of dependencies between projects
- Removing duplication between projects
- Checking for gaps

# Project portfolio management - continued

---

There are three elements to PPM:

## 1. Project portfolio definition

- Create a central record of all projects within an organization
- Must decide whether to have ALL projects in the repository or, say, only ICT projects
- Note difference between new product development (NPD) projects and renewal projects e.g. for process improvement

# Project portfolio management - continued

---

## 2. Project portfolio management

Actual costing and performance of projects can be recorded and assessed

## 3. Project portfolio optimization

Information gathered above can be used to achieve better balance of projects e.g. some that are risky but potentially very valuable balanced by less risky but less valuable projects

You may want to allow some work to be done outside the portfolio e.g. quick fixes

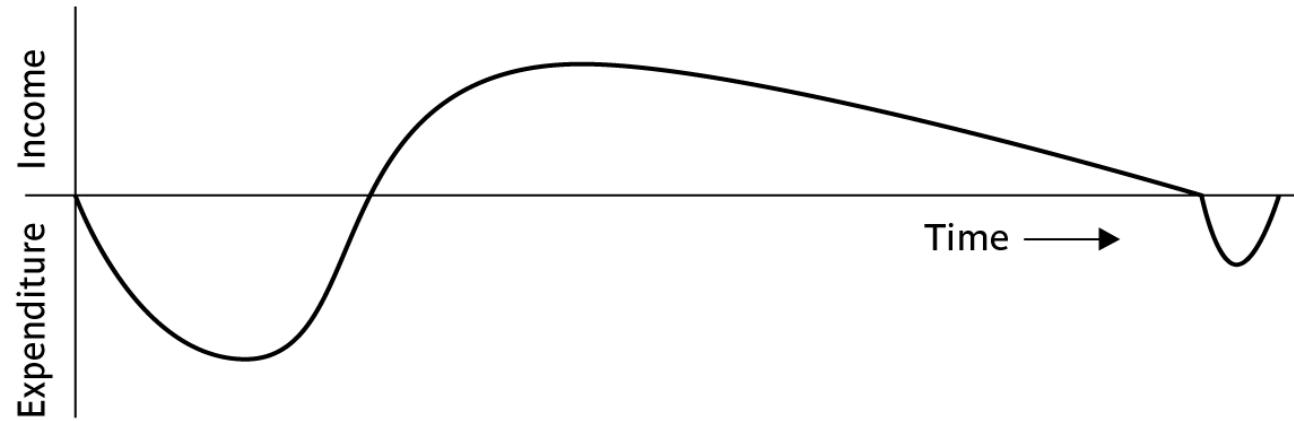
# Cost benefit analysis (CBA)

---

This relates to an individual project. You need to:

- Identify all the costs which could be:
  - Development costs
  - Set-up
  - Operational costs
- Identify the value of benefits
- Check benefits are greater than costs

# Product/system life cycle cash flows



- The timing of costs and income for a product of system needs to be estimated.
- The development of the project will incur costs.
- When the system or product is released it will generate income that gradually pays off costs
- Some costs may relate to decommissioning – think of demolishing a nuclear power station.

# Net profit

Year	Cash-flow
0	-100,000
1	10,000
2	10,000
3	10,000
4	20,000
5	100,000
Net profit	50,000

‘Year 0’ represents all the costs before system is operation

‘Cash-flow’ is value of income less outgoing

Net profit value of all the cash-flows for the lifetime of the application

# Pay back period

This is the time it takes to start generating a surplus of income over outgoings. What would it be below?

Year	Cash-flow	Accumulated
0	-100,000	-100,000
1	10,000	-90,000
2	10,000	-80,000
3	10,000	-70,000
4	20,000	-50,000
5	100,000	50,000

# Return on investment (ROI)

---

$$\text{ROI} = \frac{\text{Average annual profit}}{\text{Total investment}} \times 100$$

*In the previous example*

- average annual profit  
=  $50,000/5$   
= 10,000
- ROI =  $10,000/100,000 \times 100$   
= 10%

# Net present value (NPV)

---

Would you rather I gave you Rs. 100 today or in 12 months time?

If I gave you Rs. 100 now you *could* put it in savings account and get interest on it.

If the interest rate was 10% how much would I have to invest now to get Rs.100 in a year's time?

This figure is the ***net present value*** of Rs.100 in one year's time

# Discount factor

---

Discount factor =  $1/(1+r)^t$

$r$  is the interest rate (e.g. 10% is 0.10)

$t$  is the number of years

In the case of 10% rate and one year

Discount factor =  $1/(1+0.10) = 0.9091$

In the case of 10% rate and two years

Discount factor =  $1/(1.10 \times 1.10) = 0.8294$

# Applying discount factors

Year	Cash-flow	Discount factor	Discounted cash flow
0	-100,000	1.0000	-100,000
1	10,000	0.9091	9,091
2	10,000	0.8264	8,264
3	10,000	0.7513	7,513
4	20,000	0.6830	13,660
5	100,000	0.6209	62,090
		NPV	618

# Dealing with uncertainty: Risk evaluation

---

- Project A might appear to give a better return than B but could be riskier
- Draw a project risk matrix for each project to assess risks
- For riskier projects, we could use higher discount rates

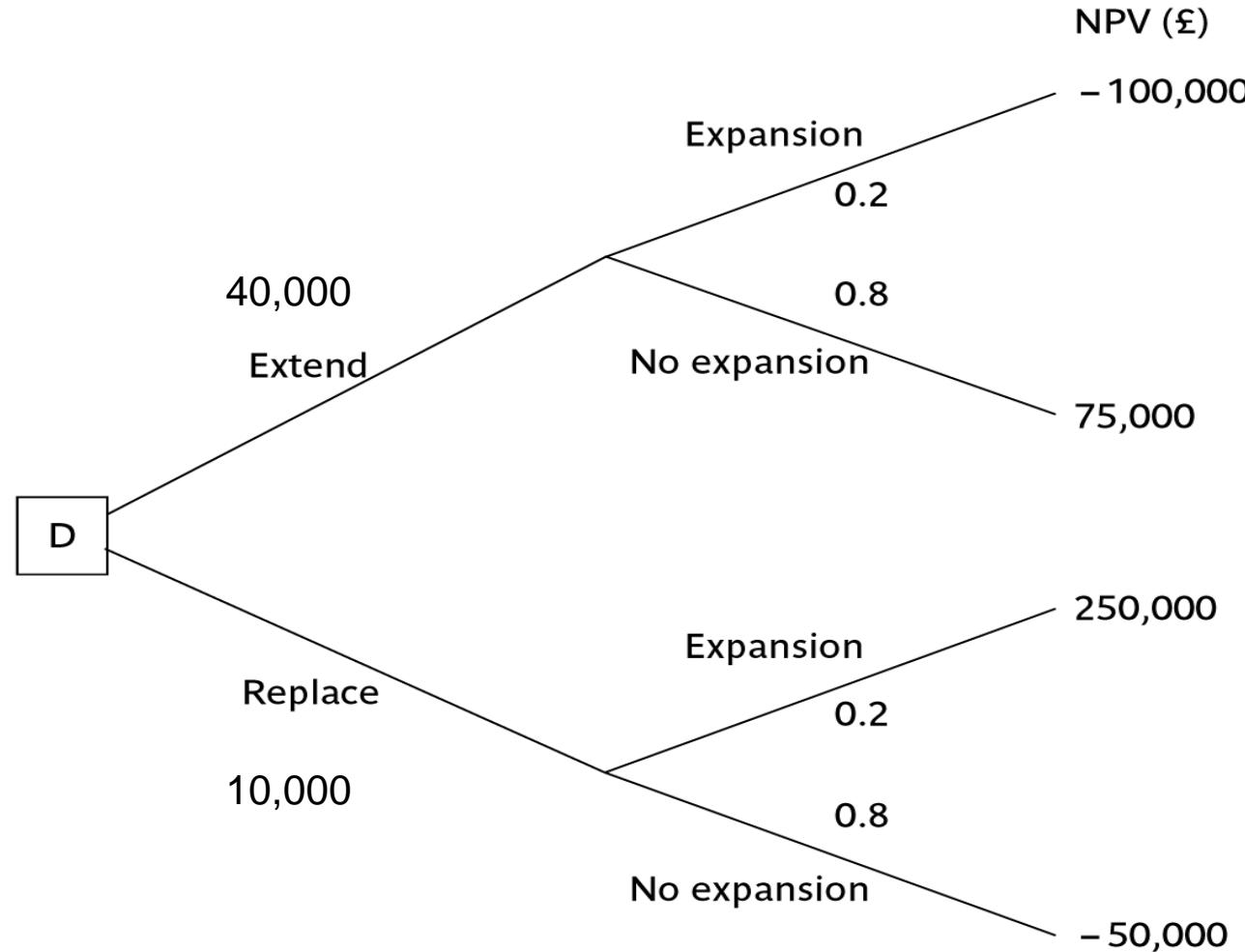
# Example of a project risk matrix

Risk	Importance	Likelihood
Client rejects proposed look and feel of site	H	—
Competitors undercut prices	H	M
Warehouse unable to deal with increased demand	M	L
Online payment has security problems	M	M
Maintenance costs higher than estimated	L	L
Response times deter purchasers	M	M

TABLE 2.5 A fragment of a basic project/business risk matrix for an e-commerce application

# Decision trees

(Decision: When to replace its System / Evaluate Expected Benefit for each Decision / Uncertainty: Market expansion / no-expansion)



# Programme management

---

- One definition:

*'a group of projects that are managed in a co-ordinated way to gain benefits that would not be possible were the projects to be managed independently'* Ferns

# Programmes may be

---

- Strategic
- Business cycle programmes
- Infrastructure programmes
- Research and development programmes
- Innovative partnerships

# Programme managers vs. Project managers

## Programme manager

- Many simultaneous projects
- Personal relationship with skilled resources
- Optimization of resource use
- Projects tend to be seen as similar

## Project manager

- One project at a time
- Impersonal relationship with resources
- Minimization of demand for resources
- Projects tend to be seen as unique

# Strategic programmes

---

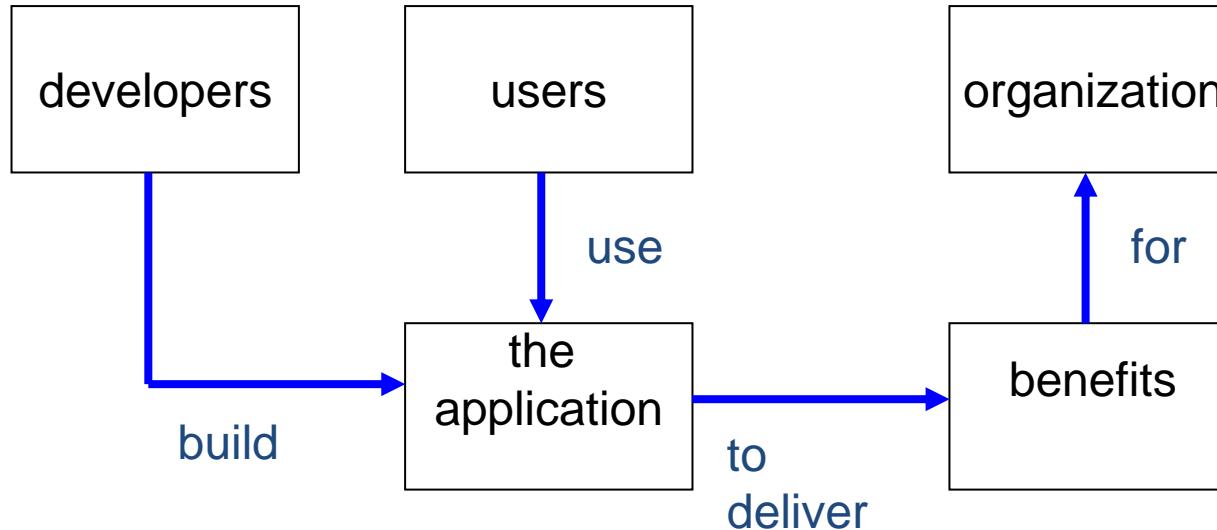
- Initial planning document is the **Programme Mandate** describing
  - The new services/capabilities that the programme should deliver
  - How an organization will be improved
  - Fit with existing organizational goals
- A **programme director** appointed a champion for the scheme

# Next stages/documents

---

- **The programme brief** – equivalent of a feasibility study: emphasis on costs and benefits
- **The vision statement** – explains the new capability that the organization will have
- **The blueprint** – explains the changes to be made to obtain the new capability

# Benefits management



- Providing an organization with a capability does not guarantee that this will provide benefits envisaged – need for *benefits management*
- This has to be outside the project, done at *programme level*

# Benefits management

---

To carry this out, you must:

- Define expected benefits
- Analyse balance between costs and benefits
- Plan how benefits will be achieved
- Allocate responsibilities for their achievement
- Monitor achievement of benefits

# Benefits

---

These might include:

- Mandatory requirement
- Improved quality of service
- Increased productivity
- More motivated workforce
- Internal management benefits
- Risk reduction
- Economies
- Revenue enhancement/acceleration
- Strategic fit

# Quantifying benefits

---

Benefits can be:

- Quantified and valued e.g. a reduction of  $x$  staff saving Rs.  $y$
- Quantified but not valued e.g. a decrease in customer complaints by  $x\%$
- Identified but not easily quantified – e.g. public approval for a organization in the locality where it is based

# Project Evaluation: Summary

---

- A project may fail not through poor management but because it should never have been started
- A project may make a profit, but it may be possible to do something else that makes even more profit
- A real problem is that it is often not possible to express benefits in accurate financial terms
- Projects with the highest potential returns are often the most risky

---

# Thank You



# *Chammach* VS *Chopsticks*



**Cross-Cultural Lessons from  
Indian and Japanese Software Project Mgmt.**

**KG KRISHNA**  
5 Sep 2010

**Prefectures of Japan**

1. Aichi
2. Akita
3. Aomori
4. Chiba
5. Ehime
6. Fukui
7. Fukuoka
8. Fukushima
9. Gifu
10. Gunma
11. Hiroshima
12. Hokkaido
13. Hyogo
14. Ibaraki
15. Ishikawa
16. Iwate
17. Kagawa
18. Kagoshima
19. Kanagawa
20. Kochi
21. Kumamoto
22. Kyoto
23. Mie
24. Miyagi
25. Miyazaki
26. Nagano
27. Nagasaki
28. Nara
29. Niigata
30. Oita
31. Okayama
32. Okinawa
33. Osaka
34. Saga
35. Saitama
36. Shiga
37. Shimane
38. Shizuoka
39. Tochigi
40. Tokushima
41. Tokyo
42. Tottori
43. Toyama
44. Wakayama
45. Yamagata
46. Yamaguchi
47. Yamamanashi



# PLEASE, MIND YOUR STEP.

When crossing 『臥竜橋』,

be careful of the footing sufficiently.

Understand beforehand because the responsibility can not be assumed about the accident in case and so on.

## ご利用下さい。

## 2階のトイレは、ご利用頂けます

The second floor is under preparation now. Please be afraid, and although it is needed, use the seat for audience of the first floor. use the toilet of the second floor.



### お願い

ペットの糞は、お持ち帰り下さい。



W i s h  
Please bring a pet's excrement home.

# Cultural Backdrop

- Dual Personalities (“Outer” / “Inner”)
- Integrated Work-Life
- Expect Conformity / Harmony in Group
- Mutual Trust/Respect
- Supplier/Vendor Relationship
- Honouring Commitments
- Consensus Driven
  
- Highly Observant
- Detail Oriented (Metrics)
- Extensive Note-Takers
- Visual/Graphics

# Initiating Relationship

- Long-Drawn / Long-Term
- Based On Mutual Trust
- Require Patience (Testing the Waters)
- Pilot Project(s)
- Business Intermediary
- Scope for Negotiations
- Positive Attitude

# Project Initiation

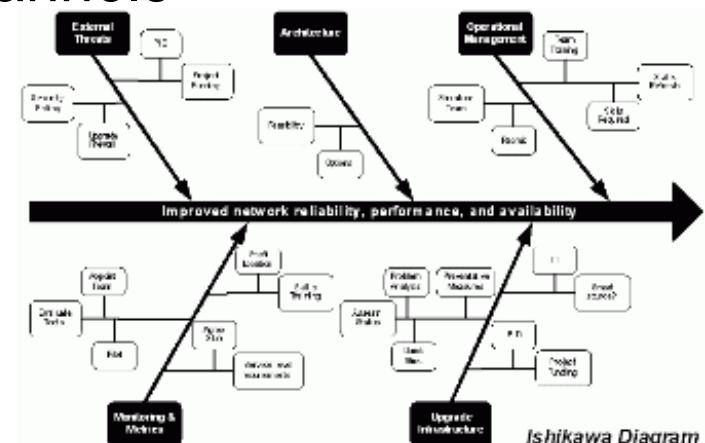
- Aptitude & Attitude is the Key
  - Do a LOT of Homework
  - Honesty and Consistency
  - Love Your Company/Country/Competition
  - Build Personal Relationship
- 
- “Requirements To Capture?” – Not Really!
  - Ask Questions & Appreciate Answers
  - You Don’t Get All Requirements in One-Go
  - Onsite KT is Just the Beginning

# Change Management

- “Change Management”?
- Requirements ALWAYS Change
- So is, Effort
- Deadlines are DEADLY.
- Cost/Budgets are FIXED
- Quality should be the HIGHEST

# Project Communications

- No “English”
  - Extensive Visuals / Graphics / Prototypes
  - Adopt “Language of Metrics”
  - Email / Fax / Personal Meetings
  - Do NOT Force Decision Making
  - Proactive Communication of Issues
  - Leverage “Informal/After-Office” Channels
  - “Issue – Backup” Plan
  - Apologize. Apologize. Apologize



# Project Delivery

- No Last-Minute Surprises
- Help “Save Face”
- Use “Informal” Channels
- Have a Trusted Negotiator
- Emphasize Relationship
- Focus on Analysis/Lessons-Learnt
- No Blame-Game
- Always Suggest ‘Way Forward’

# Testing & QC

- Developers design, write test cases, and run test case.
- Statistical bug data is collected throughout product life cycle.
- Project Manager analyzes the data to control the project.
- Independent QA engineers Test Tools.

# **Only 2 Bugs / 10,000 Surface at Customer's Site.**

## **% Bug Detection in Software Development Phases:**

Code-after-release: 21.5

Unit-testing: 35.1

Subsystem-testing: 28.0

System Testing: 6.1

Inspection (QC Dept.): 9.3

At Customer Site: 0.02

# Test-Case Design Guidelines

## a. Approx. 1 test case per 5 - 15 LOC

Language processor	1 test case per 8-12LOC
Online system	1 test case per 5-10 LOC
Batch system	1 test case per 10-15 LOC

## b. Normal cases

Abnormal cases	60% or less,
Boundary cases	15% or more,
Environmental cases	10% or more,

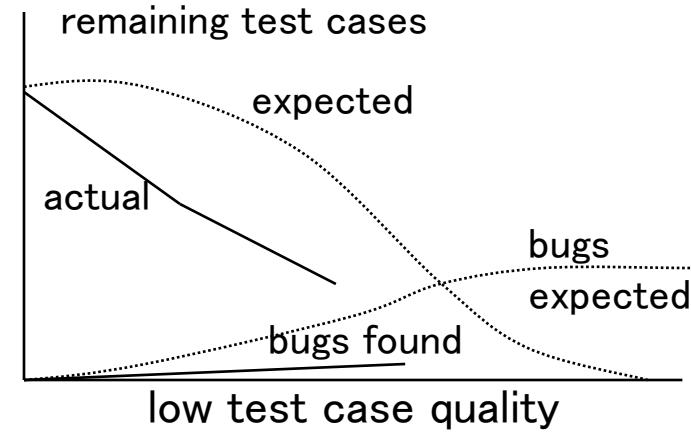
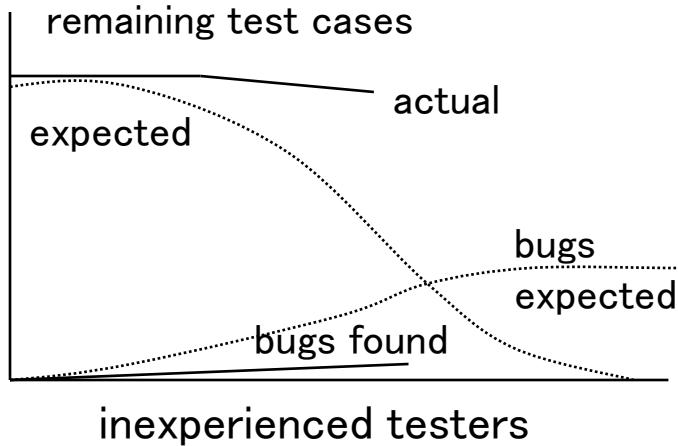
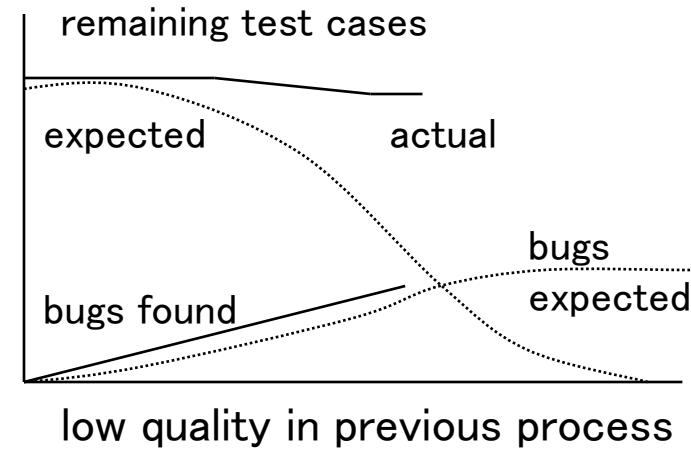
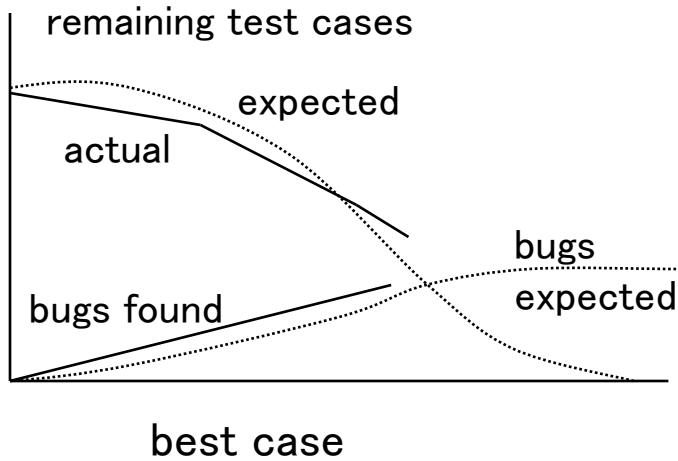
  

Abnormal cases	15% or more
Boundary cases	10% or more
Environmental cases	15% or more

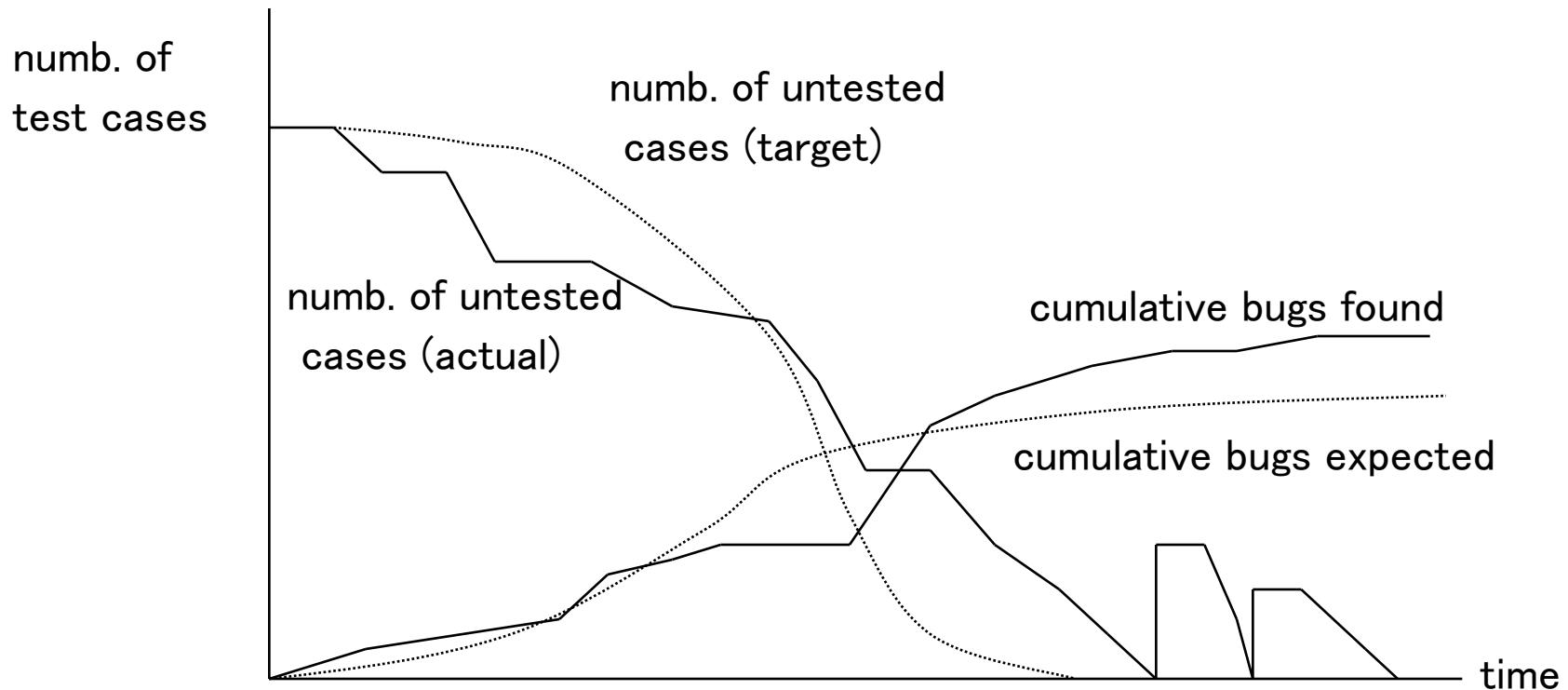
# Test-Case Matrix

test case ID		AD001	AD002	AD003	AD004	AD005	AD006	AD007	AD008
Age	ZZ	X							
	-1		X						
	0			X					
	6				X				
	12					X			
	18						X		
	19							X	
	999								X
Admission Fee	\$0			X	X				
	\$5					X			
	\$8						X		
	\$10							X	
	Err msg	X	X						X
Category		abnrm	bound	norm	norm	norm	norm	norm	abnrm
Code inspection		5–May	6–May						
Machine test		1–Jun	1–Jun	2–Jun	2–Jun	2–Jun	2–Jun	3–Jun	3–Jun

# Good and Bad Testing



# When To Release Software

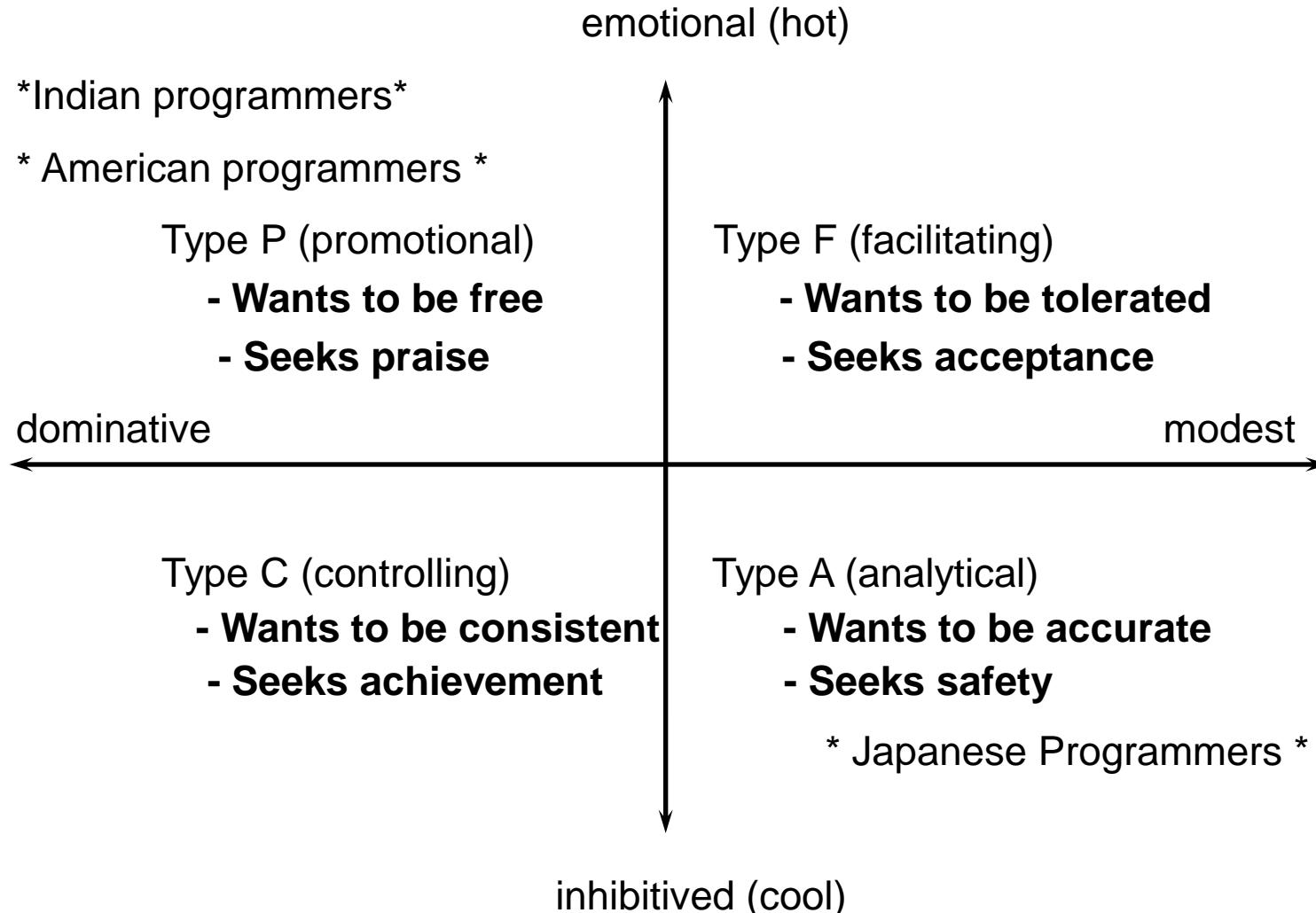


# Developers Write Test-cases

- Density of Test-Cases Measure Programmer Competency
- Statistical Bug Data collection throughout the life cycle
- Bug Data Analysis by Project Manager
  - Determine the best strategies to improve the quality
  - Predict the release date
- Independent QA engineers Test Tools
  - Approximately 8% of the entire engineers (330 out of 4,000).

# Classification of Behaviours

(Shepard's)



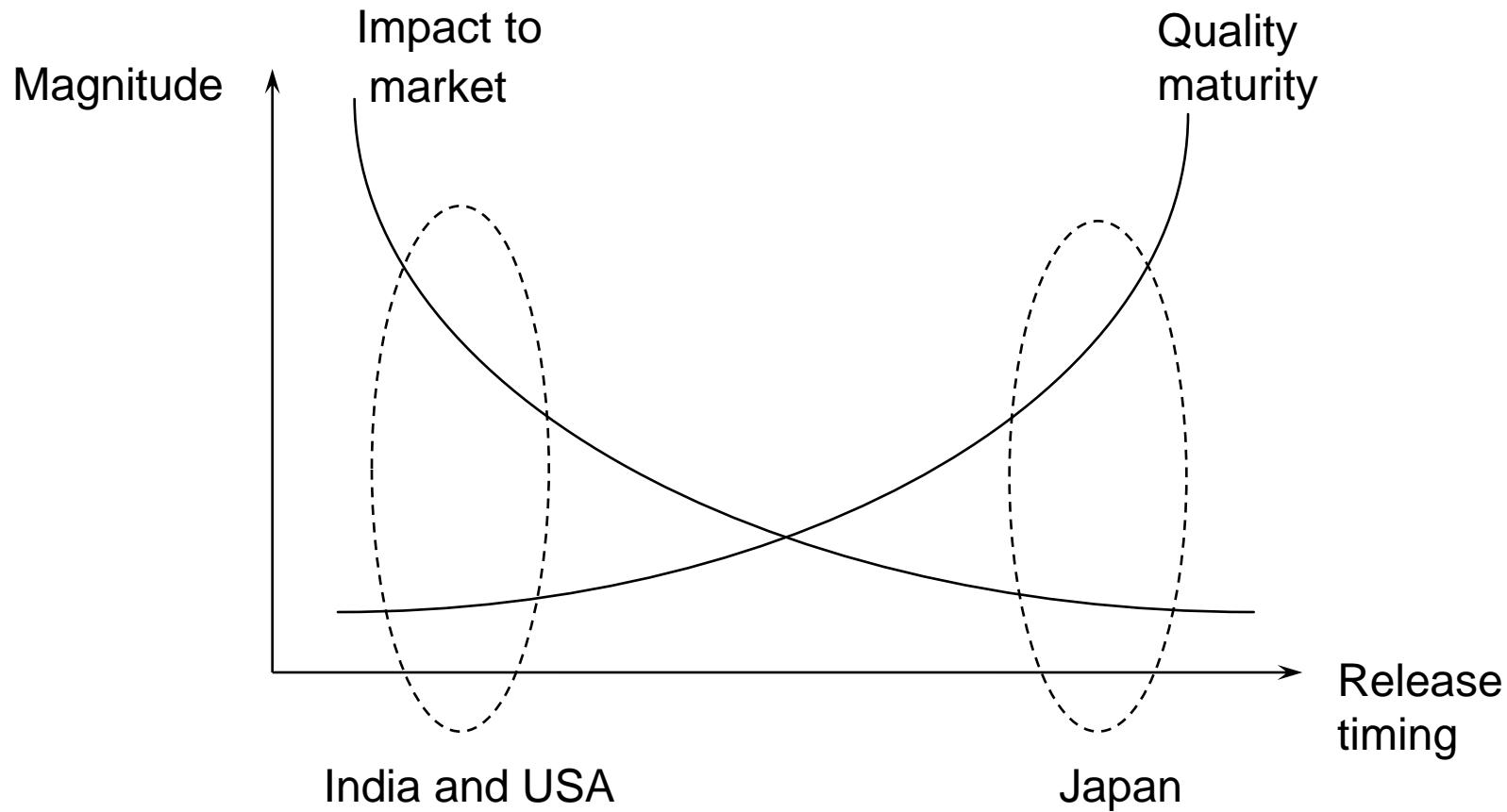
# The cost of the quality

- **A 12-month project with 10 developers** (working on C-based software with 100,000 LOC using Water-Fall Life Cycle Model)

Requirement specs.	2 months
Design	3 months
Coding	2 months
Debugging	3 months
Testing (by QA engrs.)	2 months

- **Details of the 3-month debugging phase**
  - 100,000 LOC software needs 10,000 test cases (1,000 per developer).
  - 10 days to design 10,000 test cases (100 a day per developer)
  - 10 days to check 1,000 test cases in code inspection (10 a day per developer)
  - 40 days to check 10,000 test cases in machine debugging (25 a day per dvlp)

# Differing Priorities/Strategies



# Summary

- Relationship First, Business Next
- Patience. Aptitude. Respect
- Continuous Improvement
- Honouring Commitments
- Constant Communications / Negotiations
- Speak Language of Metrics
- Analysis. Learnings. Way-Forward
- Apologize. Apologize. Apologize

かっぱ寿司  
のお寿司は  
全品一皿  
**100円**

わさび入り

のお寿司です。  
まわさび抜きのご注文も承ります。

わさび抜き

のお寿司です。

デザート150円

のお皿です。

インターホンは……



流れていった商品の御注文の際に  
お使い下さい。

お会計ボタンは……



会計時、1回だけ押してお待ち下さい。  
係のものがお伺いします。  
(2回押すと消えてしまいますのでご注意下さい)

注文品のお渡し方法

注文品

にのせて、一段高くして  
お流しいたしますのでお取り  
下さい。

★ビールの御注文は席までお持ちいたします★

★お冷やはお近くの従業員にお申し付け下さい★

◆御注文の品が届かない場合は、お近くの従業員に  
お尋ね下さい。



生ビール(中)  
みそ汁 150円 茶碗蒸 750円 ビール(中) 450円



一度お取りになった商品はレーンに  
戻さないようお願いいたします。



携帯電話の店内でのご使用はご遠慮  
下さい。



当店は「全席禁煙」となっております。  
※喫煙の際は「喫煙コーナー」にてお願い致します。

お客様のご理解とご協力をお願い申し上げます

かっぱ寿司 従業員一同

Arigatō Gozaimashita

