Software Development Process

Software Dev Process

- Process is distinct from product
  - products are outcomes of executing a process on a project

- SW Eng. focuses on process

- Premise: Proper processes will help achieve project objectives of high QP

Software Process…

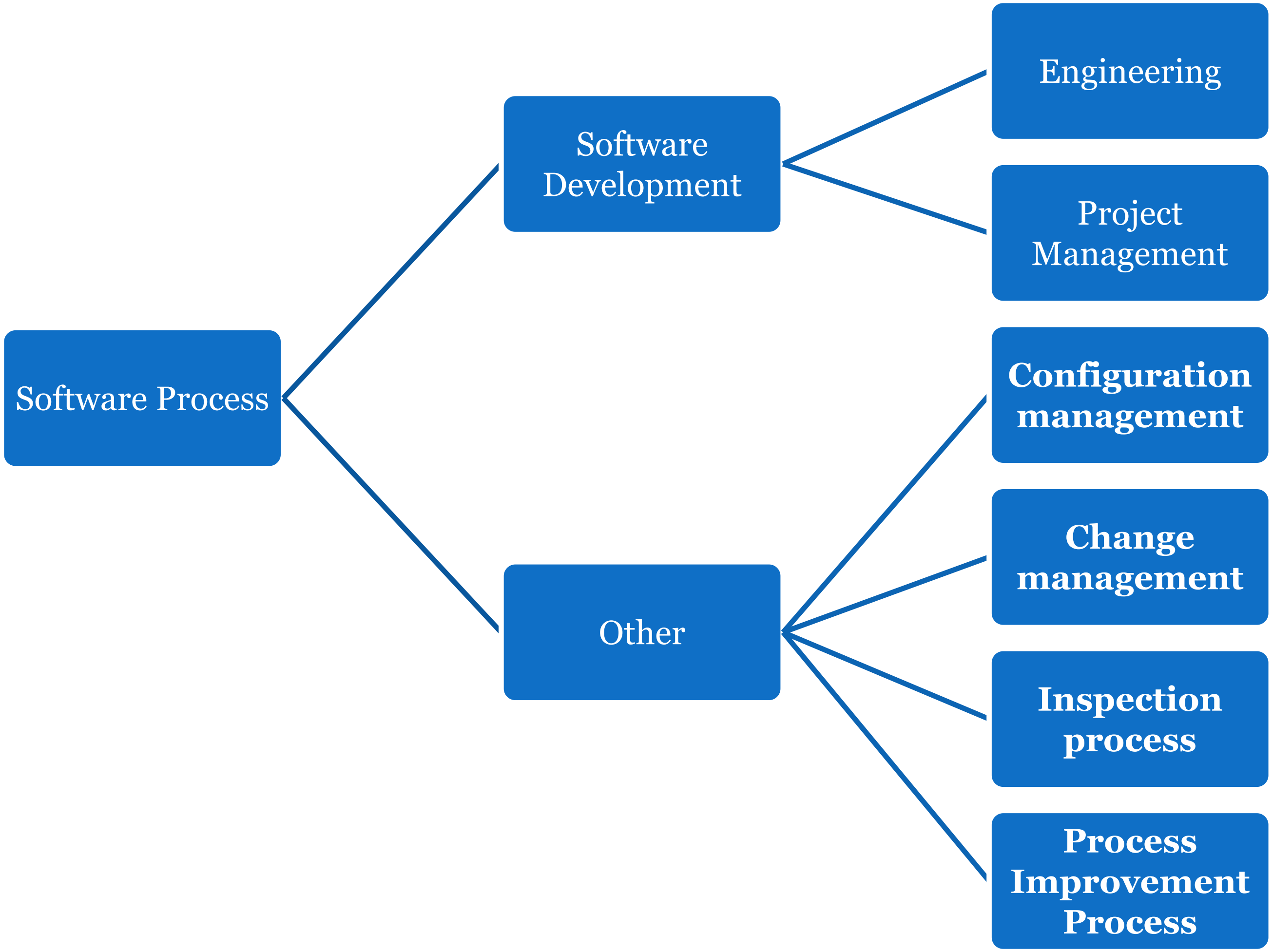
Process: A particular method, generally involving a number of steps

Software Process: A set of steps, along with ordering constraints on execution, to produce software with desired outcome

Many types of activities performed by diff people

Software process is comprising of many component processes

Key Processes



Software Dev Processes

Two major processes

- **Engineering**– “development and quality steps needed to engineer the software”
- **Project management** – “planning and controlling the development process”

Key Roles

- Developers execute Engineering process
  - Software architects, lead developers, …
- Project manager(s) executes the mgmt process

Other Processes…

Other processes

- **Configuration management**: manages the evolution of artifacts
- **Change management**: how changes are incorporated
- **Inspection process**: how inspections are conducted on artifacts
- **Process Improvement Process:**

Process Specification

Phase 1 → **Phase 2** → **Phase 3**

Phase 1
- Step A
- Step B
- Step C

Phase 2
- Step A
- Step B
- Step C

Phase 3
- Step A
- Step B
- Step C

Process is generally a set of phases

Each phase performs a well defined task and generally produces an output

Intermediate outputs – ***work products***

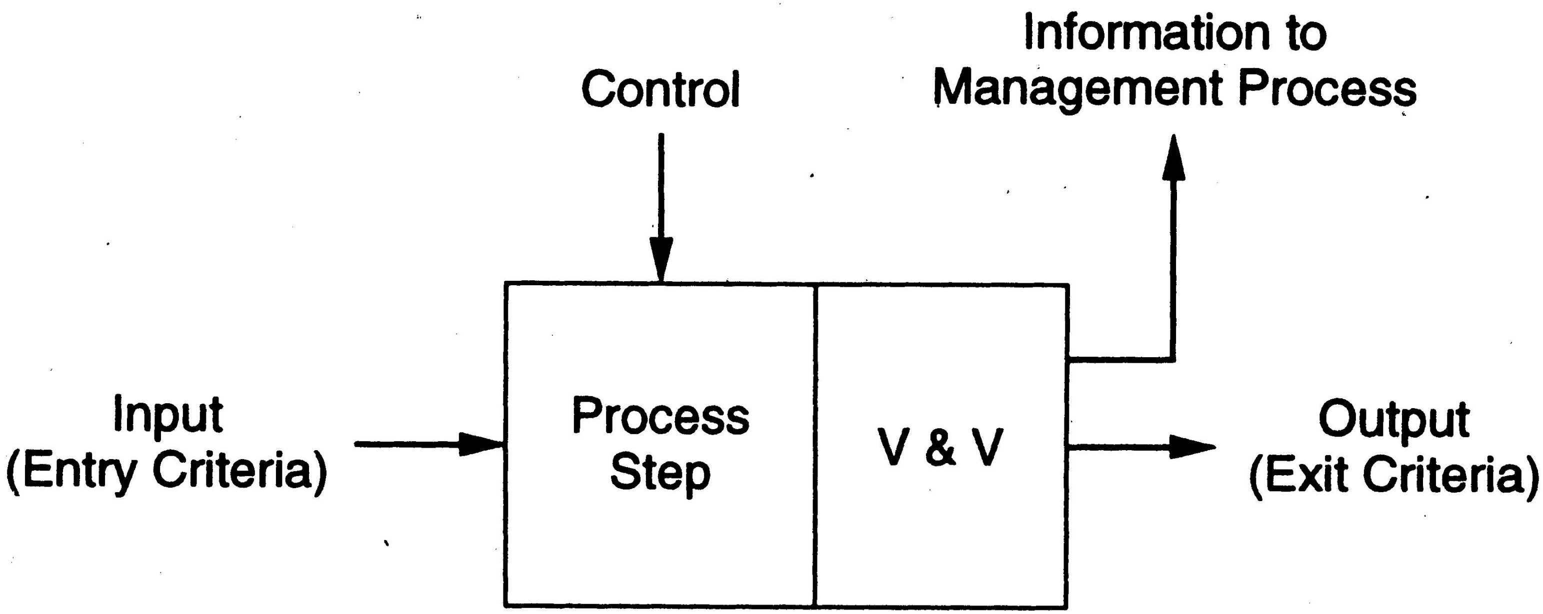
At top level, typically few phases in a process

Variety of methodologies have been proposed for each phase

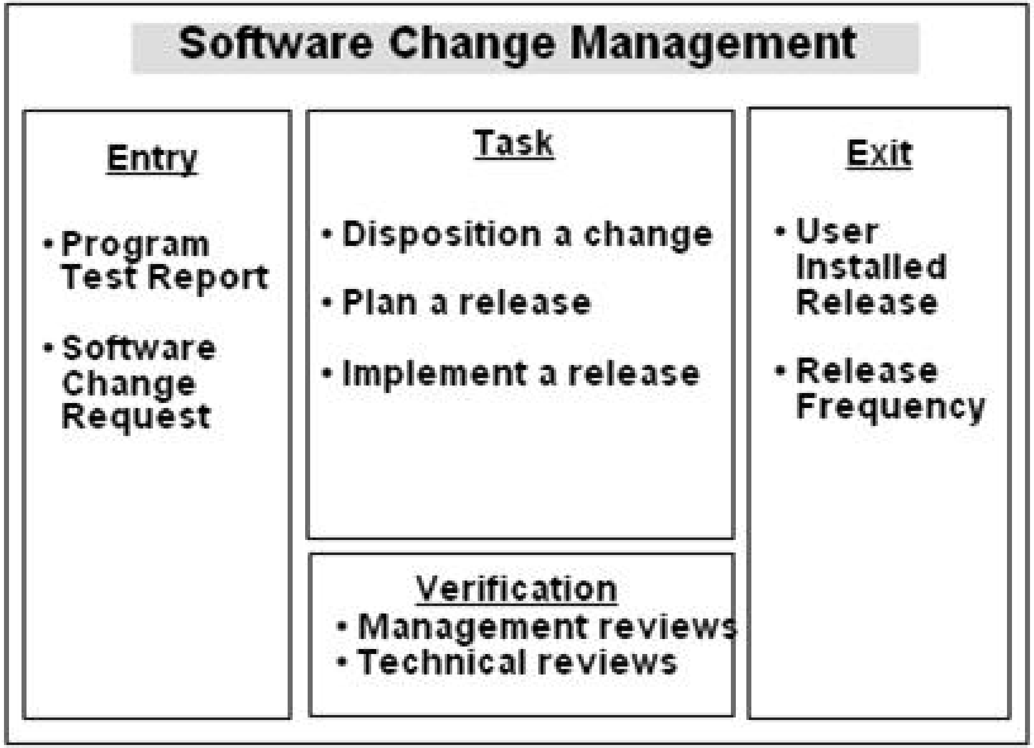
ETVX Specification

- ETVX approach to specify a phase
  - Entry criteria: what conditions must be satisfied for initiating this phase
  - Task: what is to be done in this phase
  - Verification: the checks done on the outputs of this phase
  - eXit criteria: when can this phase be considered done successfully
- A phase also produces info for mgmt

ETVX approach



ETVX Example: Change management



Key Dev Process Properties

Provide high Q&P by

1. Early defect removal
2. Execute in a Predictable and repeatable manner
3. Support Change

1) Early Defect Removal…

- Provide high Q&P
  - Support testability
    - testing can consume 30 to 50% of total development effort
  - Support maintainability
    - maintenance can be more expensive than development;
    - over life up to 80% of total cost
  - Remove defects early, as cost of removing defects increases with latency

1) Early Defect Removal…

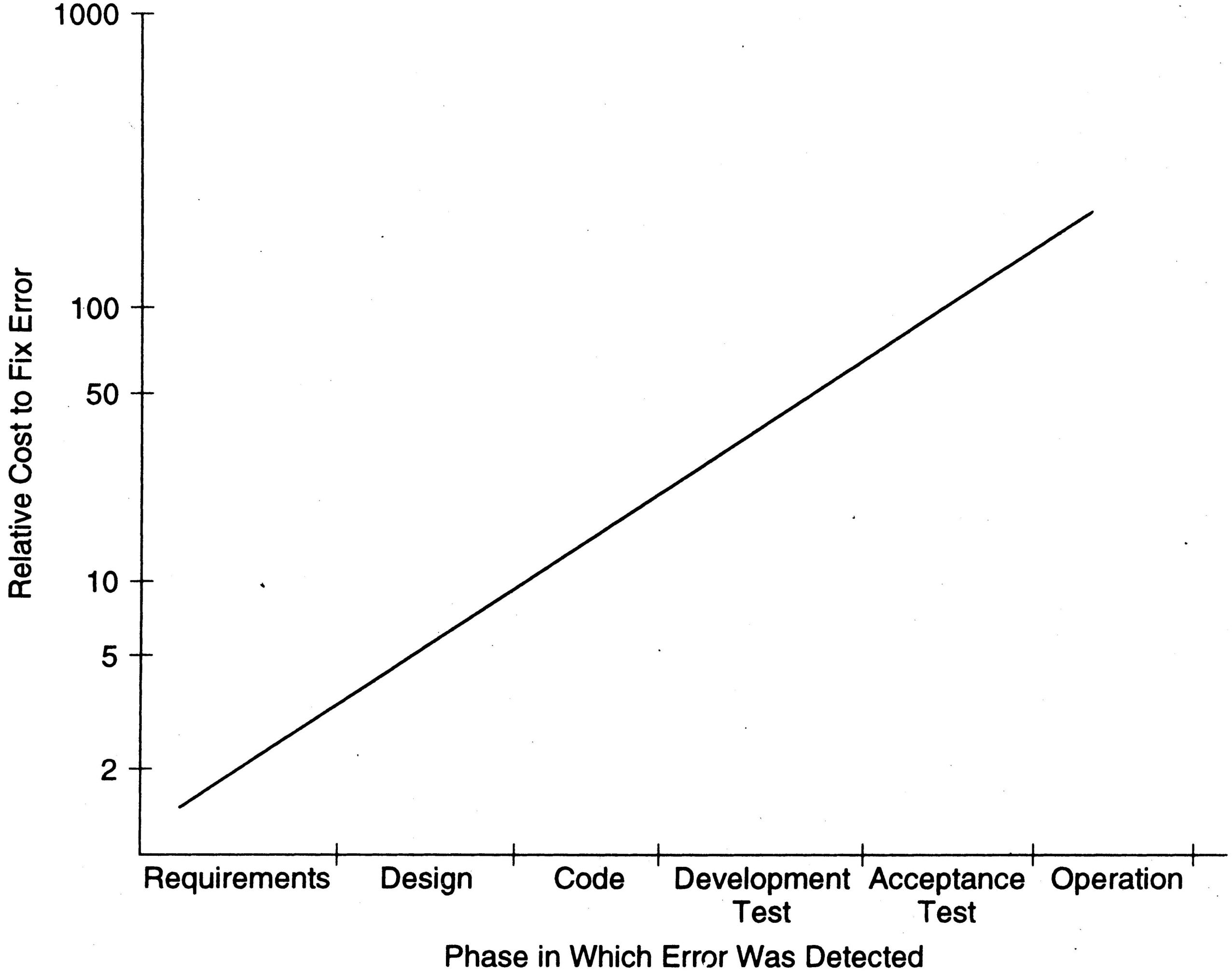
Cost of a defect increases with latency

- fixing a req defect
  - During requirements: x
  - During operation: 100x

the process must support early defect removal

That is why there is a V in ETVX, and quality control tasks in the sw process

3) Early Defect Removal…



2) Predictability and repeatability

Process should repeat its performance when used on different projects

outcome of using a process should be predictable!

Without predictability,

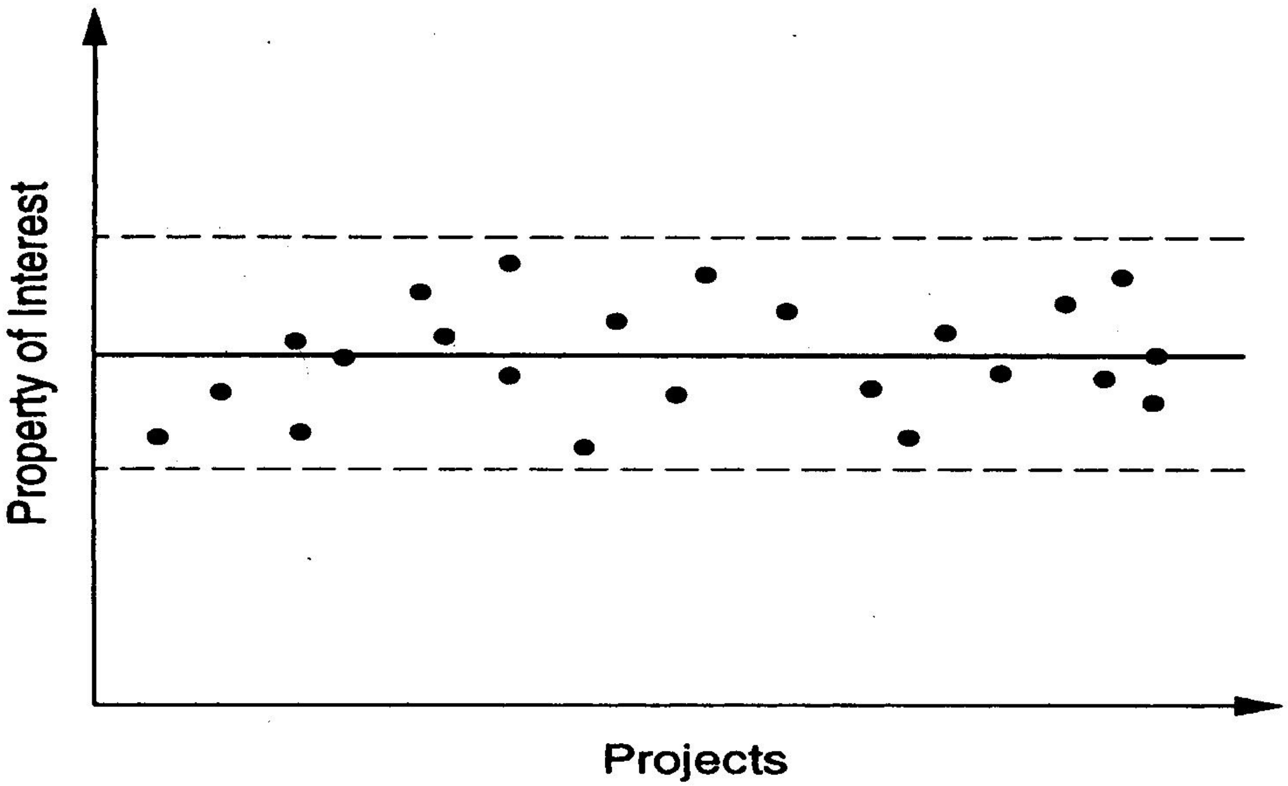
can’t estimate effectively,

can’t say anything about quality or productivity

With predictability,

past performance can be used to predict future performance

2) Predictability…



2) Predictability…

- Predictable process is said to be under statistical control
  - Repeatedly using the process produces similar results
  - Results – properties of interest like quality, productivity, …
- To consistently develop sw with high Q&P, process must be in control

3) Support Change

Software changes for various reasons

- Requirements change
- Infrastructure changes
- Designs change

Changes in Requirements

- cannot be wished away or treated as “bad”
- must be accommodated in the SW development process

Summary

Process – method for doing something

- Process typically has stages, each stage focusing on an identifiable task
- Stages have methodologies

Software process - methods for developing software

- Best to view it as comprising of multiple processes

Models of the Development Process

How are we going to go about it?

Development Process

- A set of phases and each phase being a sequence of steps
- For each phase there are
  - A variety of methodologies
  - Corresponding to different sequence of steps for a phase
- Why have phases?
  - To employ divide and conquer
  - Each phase handles a different part of the problem
  - Helps in continuous validation

Development Process

Commonly has these activities:

1. Requirements analysis,
2. Design
3. Coding,
4. Testing,
5. Delivery

Different models perform them in different manner!

1. Requirement Analysis

State the problem precisely!

Forms the basis of agreement between user and developer

Specifies "**what**" not "**how**"

Hard task - needs often not understood well

Requirement specifications of even medium systems can be many hundreds of pages

Output is the SW Requirements Spec (SRS) document

2. Design

A major step in moving from problem domain to solution domain

Three main tasks

1. **Architecture design** – components and connectors that should be there in the system
2. **High level design** – modules and data structures needed to implement the architecture
3. **Detailed design** – logic of modules

Most methodologies focus on architecture or high level design

Outputs are arch/des/logic design docs

3) Coding

Converts design into code in specific language

Goal: Implement the design with simple and easy to understand code

Coding phase affects both testing and maintenance

- Well written code reduces testing and maintenance effort

Output is code

4) Testing & Quality Assurance

- Defects are introduced in each phase
- Must be found and removed to achieve high quality
- **Goal:** Identify most of defects
- Very expensive task; must be properly planned and executed
- Outputs are
  - Test plans/results, and
  - the final tested (hopefully reliable) code

4) Typical Effort Distribution ?

Req. - ?

Design - ?

Coding - ?

Testing - ?

4) Typical Effort Distribution

Distribution of effort :

Req. - 10-20%

Design - 10-20%

Coding - 20-30%

Testing - 30-50%

Coding is not the most expensive!

4) Distribution of effort…

How programmers spend their time?

Writing programs	?
Reading programs and manuals	?
Job communication	?
Others	?

4) Distribution of effort…

How programmers spend their time

Writing programs	- 13%
Reading programs and manuals	- 16%
Job communication	- 32%
Others	- 39%

Programmers spend more time in reading programs than in writing them.

Writing programs is a small part of their lives.

4) Delivery

What the “Operations” group does.

Varies by distribution model

- Shrink Wrapped Software
- In house software
- Web-based
- Software As A Service (SaaS)
- …

From a users perspective my be as important as design!

When are defects introduced?

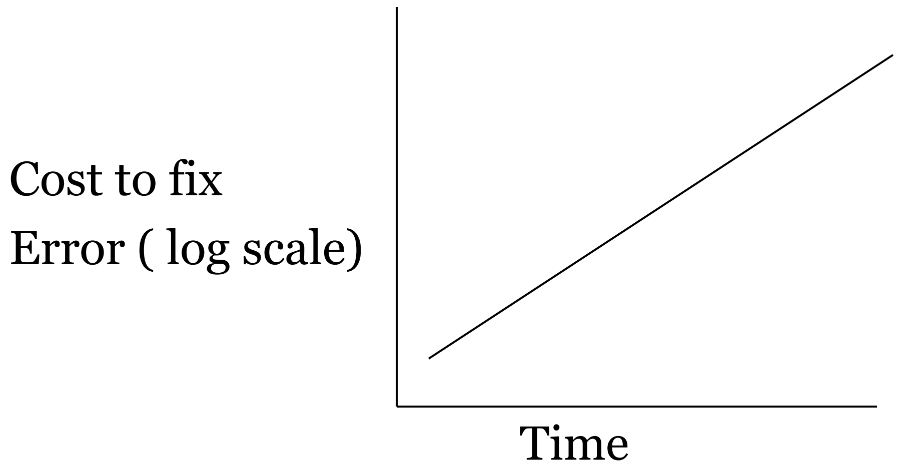
Distribution of error occurrences by phase is

Req.	- 20%
Design	- 30%
Coding	- 50%

Defects can be injected at any of the major phases.

Cost of latency: Cost of defect removal increases exponentially with latency time.

Defects…



Cheapest way to detect and remove defects close to where it is injected.

Hence must check for defects after every phase.

CSE Methodologies

Common Methods

- **Waterfall** – the oldest and widely used
- **Prototyping** – Prototype, followed by Waterfall
- **Iterative** – used widely in product dev
- **Timeboxing** – Iterative 2.0
- **Agile** - Lightweight" methodologies (Speaker)

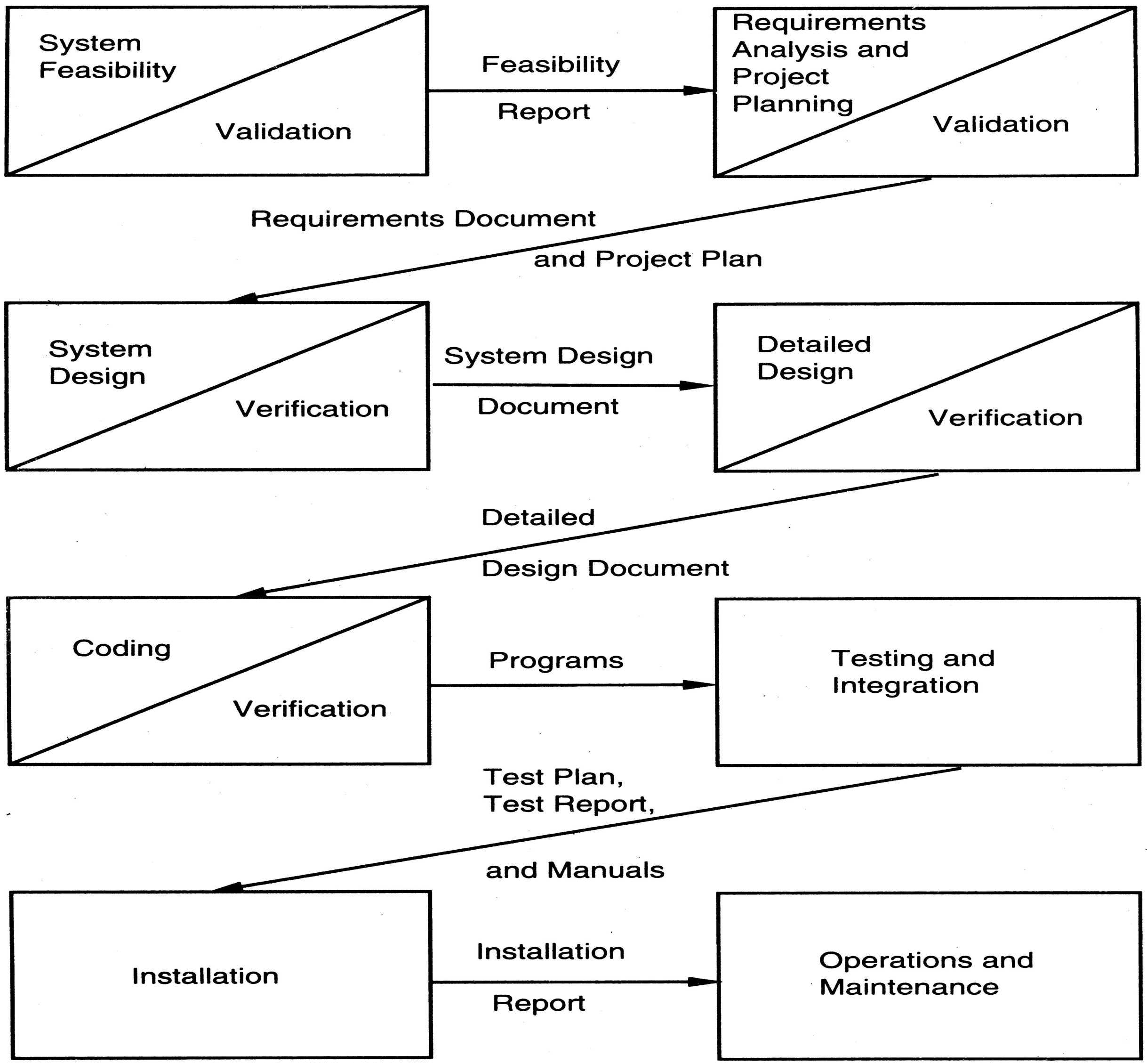
Waterfall Model

Linear sequence of stages/phases

Requirements -> HLD –> DD –> Code –> Test –> Deploy

A phase starts only when the previous has completed; no feedback!

The phases partition the project, each addressing a separate concern


System Feasibility
Validation
Feasibility
Report
Requirements Analysis and Project Planning
Validation
Requirements Document
and Project Plan
System Design
Verification
System Design
Document
Detailed Design
Verification
Detailed
Design Document
Coding
Verification
Programs
Testing and Integration
Test Plan,
Test Report,
and Manuals
Installation
Installation
Report
Operations and Maintenance

Waterfall…

Linear ordering implies each phase should have some output

The output must be validated/certified

Outputs of earlier phases: work products

Common outputs of a waterfall: SRS, project plan, design docs, test plan and reports, final code, supporting docs

Waterfall Advantages

Natural approach for problem solving

Conceptually simple, cleanly divides the problem into distinct independent phases

Easy to administer in a contractual setup – each phase is a milestone

Waterfall disadvantages

Assumes that requirements can be specified and frozen early

May fix hardware and other technologies too early

Follows the “big bang” approach – all or nothing delivery; too risky

Very document oriented, requiring docs at the end of each phase

Waterfall Usage

Well suited for projects where requirements can be understood easily and technology decisions are easy

Has been used widely

For standard/familiar type of projects it still may be the most optimum

Well suited to the out sourcing model

Prototyping

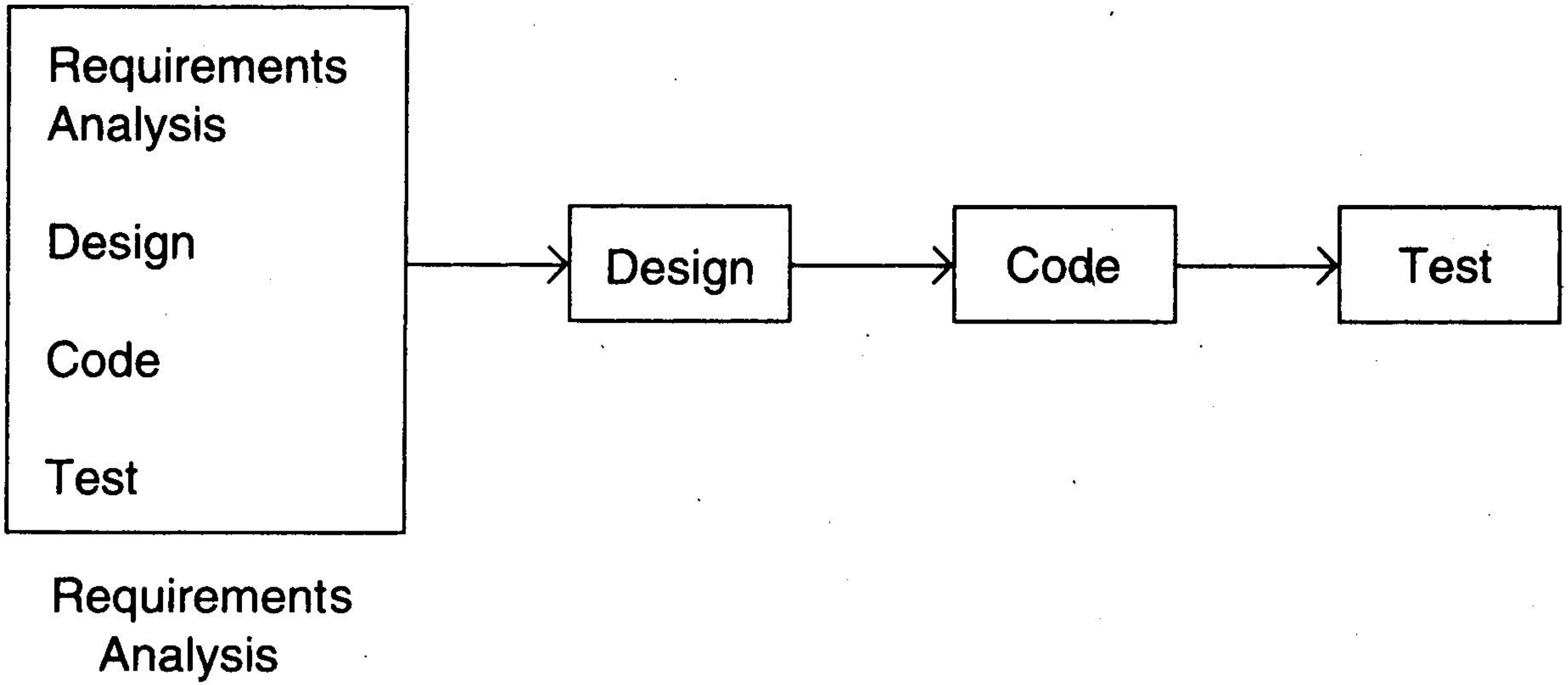
Addresses the requirement specification limitation of waterfall

Instead of freezing requirements only by discussions, a prototype is built to understand the requirements

Helps alleviate the requirements risk

A small waterfall model replaces the requirements stage

Prototyping



Development of prototype

Starts with initial requirements

Only key features which need better understanding are included in prototype

No point in including those features that are well understood

Feedback from users taken to improve the understanding of the requirements

Prototyping

- Cost can be kept low
  - Build only features needing clarification
  - “quick and dirty” – quality not important, scripting etc can be used
  - Things like exception handling, recovery, standards are omitted
  - Cost can be a few % of the total
  - Learning in prototype building will help in building, besides improved requirements

Prototyping

Advantages

- Requirement will be more stable and more likely to satisfy user needs
- Early opportunity to explore scale/performance issues
- Ability to modify or cancel the project early
- Enhanced user engagement

Disadvantages:

- Potential hit on cost and schedule
- Potential false sense of security if prototype does not focus on key (high risk) issues

Prototyping

Applicability:

- When req are hard to elicit
- When confidence in reqs is low
- Where reqs are not well understood
- When design is driven by user needs

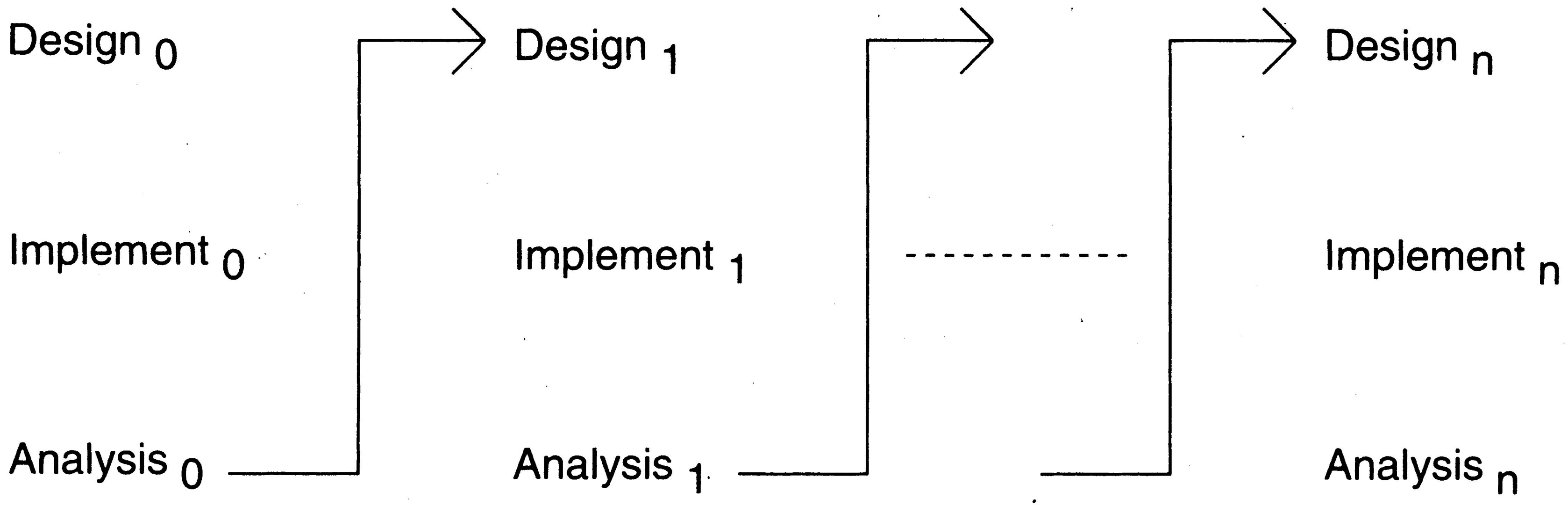
Variants

- Paper Prototypes
- UI Prototypes
- Technology Proving
- Rapid Prototyping environments

Iterative Development

- Counters the “all or nothing” drawback of the waterfall model
- Combines benefit of prototyping and waterfall
- Develop and deliver software in increments
- Each increment is complete in itself
- Can be viewed as a sequence of waterfalls
- Feedback from one iteration is used in the future iterations

Iterative Enhancement



Iterative Development

- Most Software Products follow it
- Used commonly in customized development also
  - Businesses want quick response for sw
  - Cannot afford the risk of all-or-nothing
- Newer approaches like XP, Agile,… all rely on iterative development

Iterative Development

- Benefits
  - Get-as-you-pay
  - feedback for improvement

- Drawbacks
  - Architecture/design may not be optimal
  - Amount of refactoring may increase
  - Total cost may increase

Iterative Development

Applicability

where response time is important,

risk of long projects cannot be taken,

all req not known

Execution

Each iteration is a mini waterfall – decide the specs, then plan the iteration

Length of iteration driven by amount of new functionality to be added in an iteration

Timeboxing

Time boxing is like Iterative development but

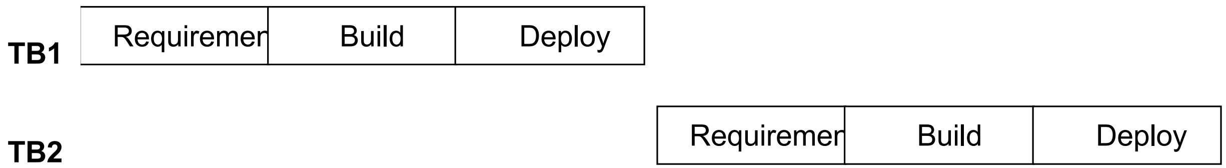
fix an iteration duration, then determine the specs

Divide iteration in a few equal stages

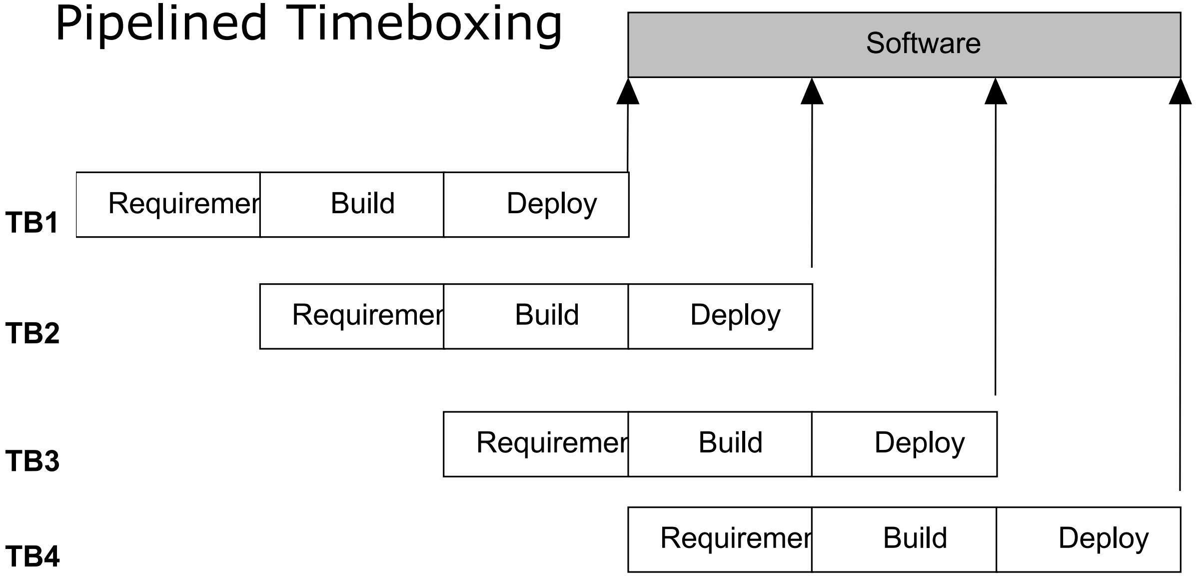
Use pipelining concepts to execute iterations in parallel

Timeboxing Execution

Linear Timeboxing



Pipelined Timeboxing



Time Boxed Iterations

General iterative development – fix the functionality for each iteration, then plan and execute it

In time boxed iterations – fix the duration of iteration and adjust the functionality to fit it

Completion time is fixed, the functionality to be delivered is flexible

Linear Timeboxing

- This itself very useful in many situations
- Has predictable delivery times
- Overall product release and marketing can be better planned
- Makes time a non-negotiable parameter and helps focus attention on schedule
- Prevents requirements bloating
- Overall dev time is still unchanged

Pipelined Timeboxing

- Multiple iterations executing in parallel
- Can reduce the average completion time by exploiting parallelism
- For parallel execution, can borrow pipelining concepts from hardware
- This leads to Pipelined Timeboxing Process Model

Pipelined Timeboxing Model – Basics

- Development is done iteratively in fixed duration time boxes
- Each time box divided in fixed stages
- Each stage performs a clearly defined task that can be done independently
- Each stage approximately equal in duration
- There is a dedicated team for each stage
- When one stage team finishes, it hands over the project to the next team

Example

- An iteration with three stages – Analysis, Build, Deploy
  - These stages are appx equal in many situations
  - Can adjust durations by determining the boudaries suitably
  - Can adjust duration by adjusting the team size for each stage
- Have separate teams for A, B, and D

Pipelined Execution

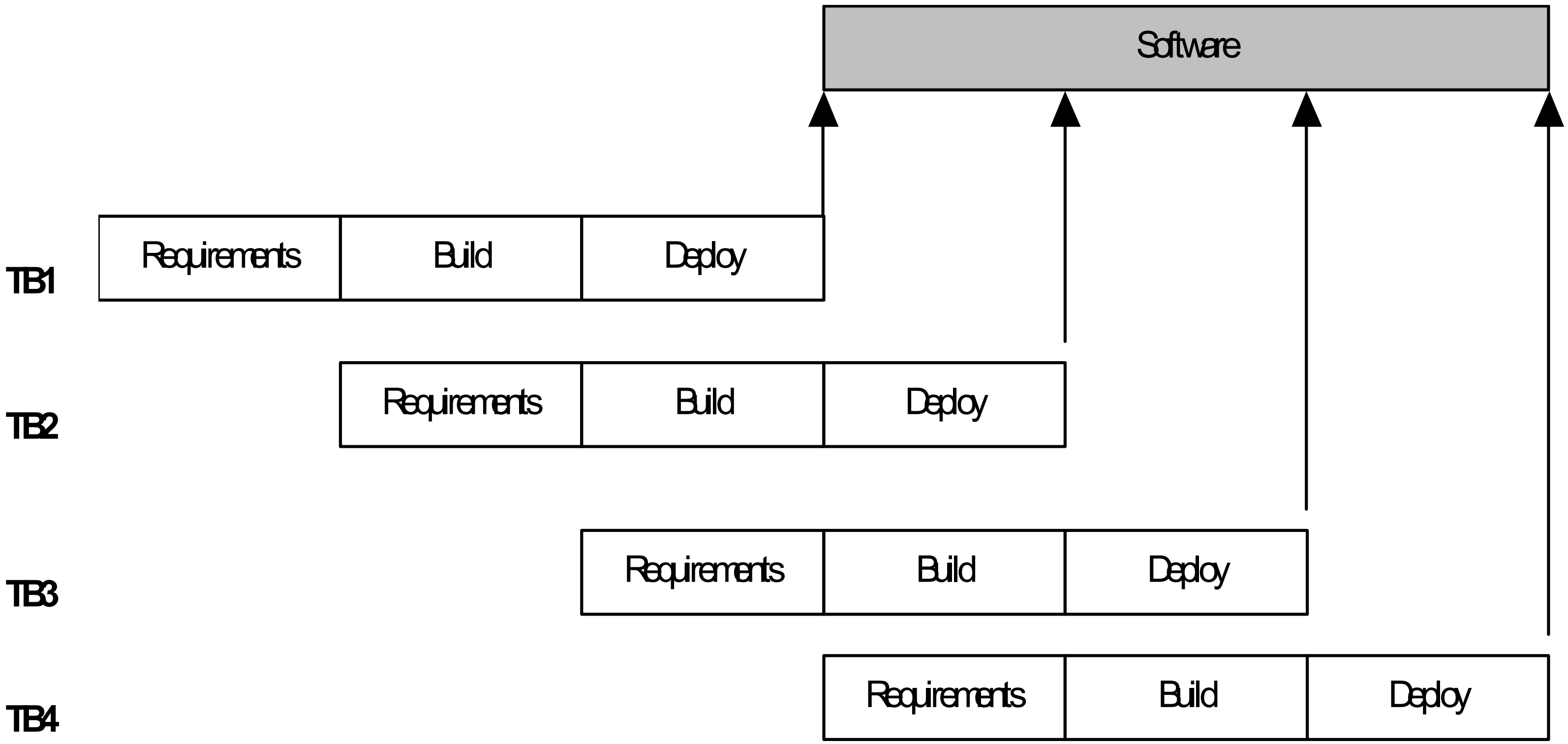
A Team starts executing it-1

A Team finishes, hands over it-1 to B Team, starts executing it-2

A Team finishes it-2, hands over to B Team; B Team finishes it-1, hands over to D Team; AT starts it-3, B Team starts it-2 (and D Team, it-1)

…

Timeboxing Execution



Timeboxing execution

Duration of each iteration still the same

Total work done in a time box is also the same

Productivity of a time box is same

Yet, average cycle time or delivery time has reduced to a third

Magic?

- Are we getting something for nothing?

Team Size

No magic, bigger teams

In linear execution of iterations, the same team performs all stages

If each stage has a team of size S, in linear execution the total team size is S

In pipelined execution, the total team size is three times (one for each stage)

Total team size in timeboxing is larger; and this what reduces cycle time

Brook’s Law

“Adding manpower to a late software project makes it later".

Fred Brooks, The Mythical Man-Month (1975)

http://en.wikipedia.org/wiki/Brooks's_law

Team Size

- Merely by increasing the team size we cannot reduce cycle time - Brook's law
- Timeboxing allows structured way to add manpower to reduce cycle time
- Note that we cannot change the time of an iteration – Brook's law still holds
- Work allocation different to allow larger team to function properly

Work Allocation of Teams

Requirements Team	Requirements Analysis for TB1	Requirements Analysis for TB2	Requirements Analysis for TB3	Requirements Analysis for TB4	
Build Team		Build for TB1	Build for TB2	Build for TB3	Build for TB4
Deployment Team			Deployment for TB1	Deployment for TB2	Deployment for TB3

Timeboxing

Advantages:

Shortened delivery times,
other adv of iterative,
distr. Execution

Disadvantages:

Larger teams,
proj mgmt is harder,
high synchronization needed,
CM is harder

Applicability

When short delivery times
When architecture is stable
Flexibility in feature grouping
For larger heavily managed teams

Summary

Process is a means to achieve project objectives of high Q&P

Process models define generic process, which can form basis of project process

Process typically has stages, each stage focusing on an identifiable task

Many models for development process have been proposed

Summary – waterfall

Strength	Weakness	Types of Projects
Simple Easy to execute Intuitive and logical Easy contractually	All or nothing – too risky Req frozen early May chose outdated hardware/tech Disallows changes No feedback from users Encourages req bloating	Well understood problems, short duration projects, automation of existing manual systems

Summary – Prototyping

Strength	Weakness	Types of Projects
Helps req elicitation Reduces risk Better and more stable final system	Front heavy Possibly higher cost and schedule Encourages req bloating Disallows later change	Systems with novice users; or areas with req uncertainity. Heavy reporting based systems can benefit from UI proto

Summary – Iterative

Strength	Weakness	Types of Projects
Regular deliveries, leading to biz benefit Can accommodate changes naturally Allows user feedback Avoids req bloating Naturally prioritizes req Allows reasonable exit points	Overhead of planning each iteration Total cost may increase System arch and design may suffer Rework may increase	For businesses where time is imp; risk of long projects cannot be taken; req not known and evolve with time

Summary – Timeboxing

Strength	Weakness	Types of Projects
All benefits of iterative Planning for iterations somewhat easier Very short delivery times	PM becomes more complex Team size is larger Complicated – lapses can lead to losses	Where very short delivery times are very important Where flexibility in grouping features Arch is stable