



Software Process



Software Process

- Process is distinct from product – products are outcomes of executing a process on a project
- SW Engg. 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 in a software project
- Better to view software process as comprising of many component processes

Component Software Processes



- Two major processes
 - Development – focuses on development and quality steps needed to engineer the software
 - Project management – focuses on planning and controlling the development process
- Development process is the heart of software process; other processes revolve around it
- These are executed by different people
 - developers execute engg. Process
 - project manager executes the mgmt proces



Component Processes...

- Other processes
 - Configuration management process: manages the evolution of artifacts
 - Change management process: how changes are incorporated
 - Process management process: management of processes themselves
 - Inspection process: How inspections are conducted on artifacts



Process Specification

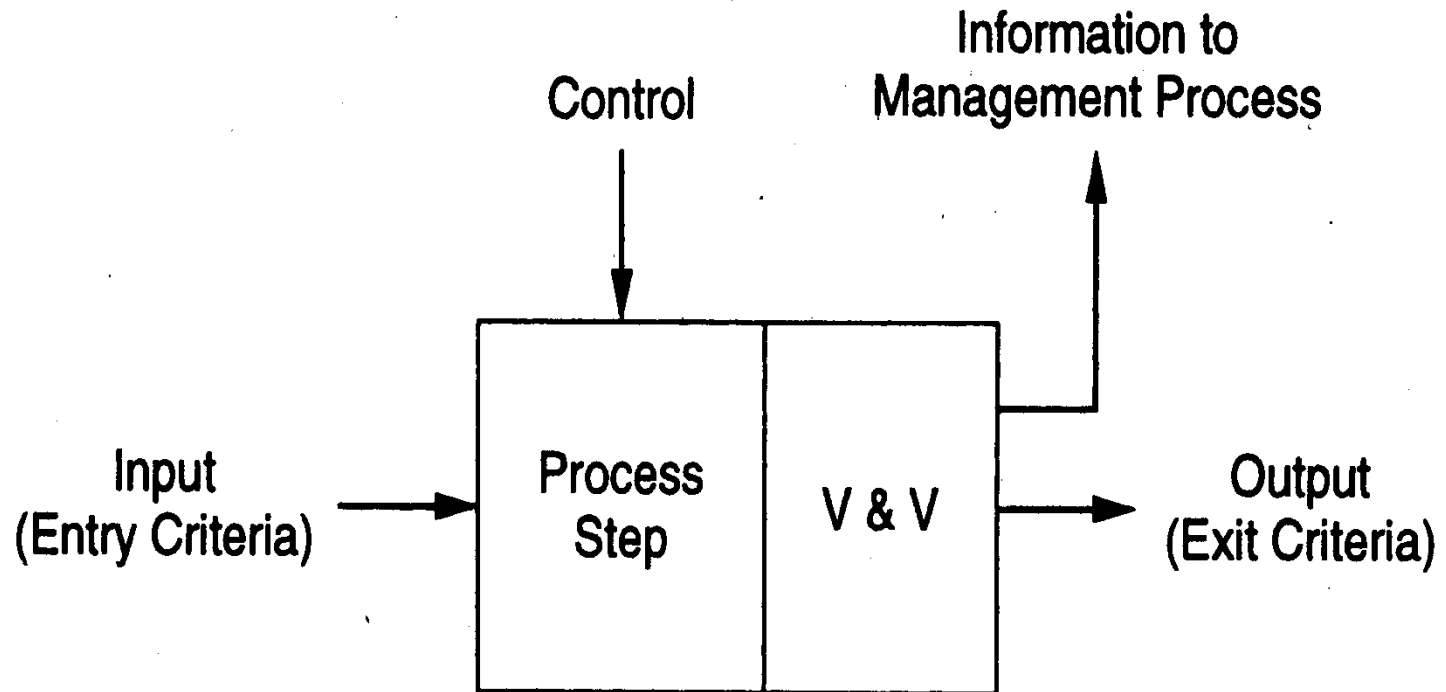
- Process is generally a set of phases
- Each phase performs a well defined task and generally produces an output
- Intermediate outputs – *work products*
- At top level, typically few phases in a process
- How to perform a particular phase – *methodologies* have been proposed



ETVX Specification

- ETVX approach to specify a step
 - 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





Desired Process Properties

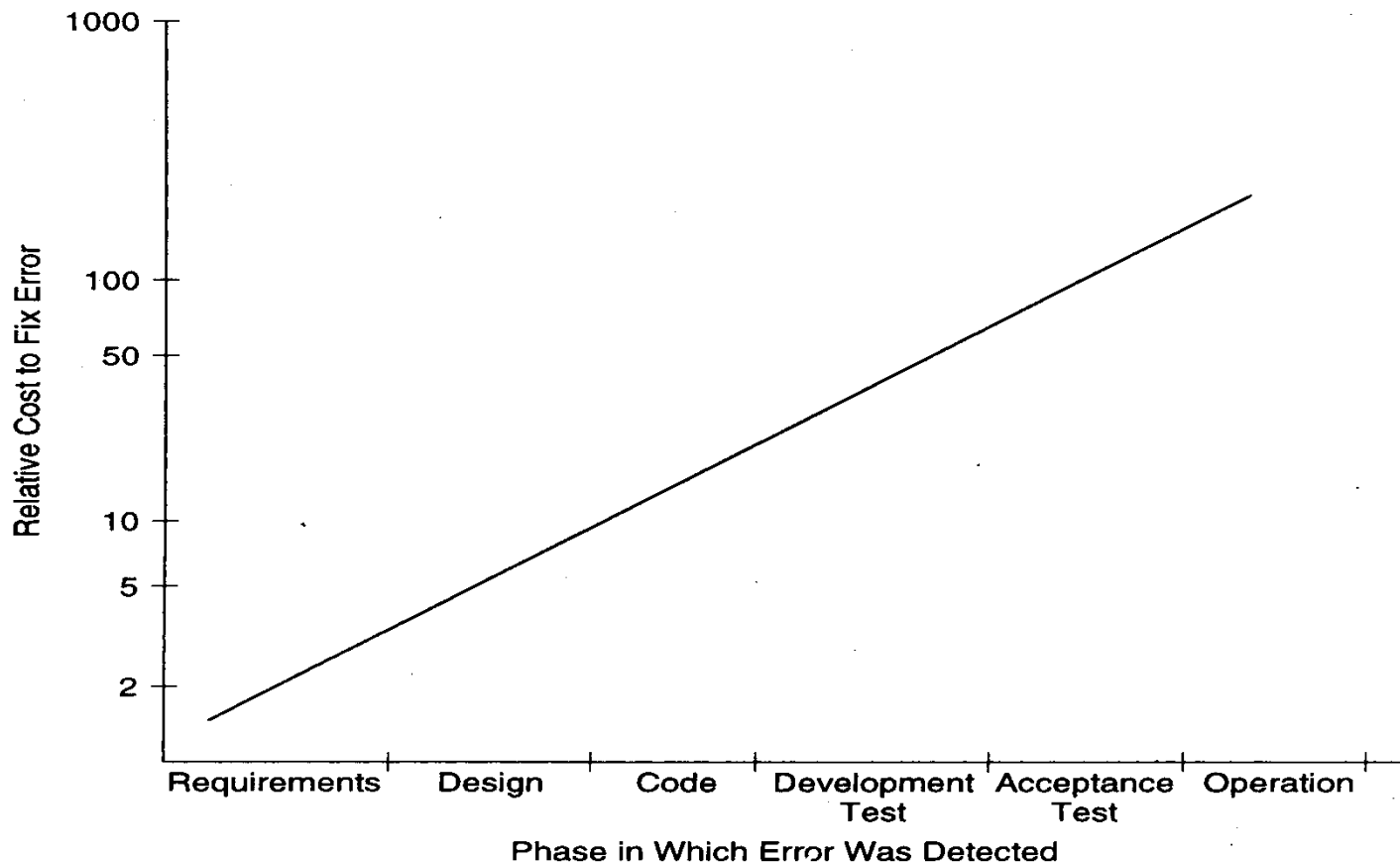
- Provide high Q&P
 - Support testability as testing is the most expensive task; testing can consume 30 to 50% of total development effort
 - Support maintainability as maintenance can be more expensive than development; over life up to 80% of total cost
 - Remove defects early, as cost of removing defects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



High Q&P: Early Defect Removal...

- Cost of a defect increases with latency
- I.e. fixing a req defect in operation can cost a 100 times the cost of fixing it in requirements itself
- Hence, for high Q&P, the process must support early defect removal
- That is why there is a V in ETVX, and quality control tasks in the sw process

Early Defect Removal...





Desired Properties...

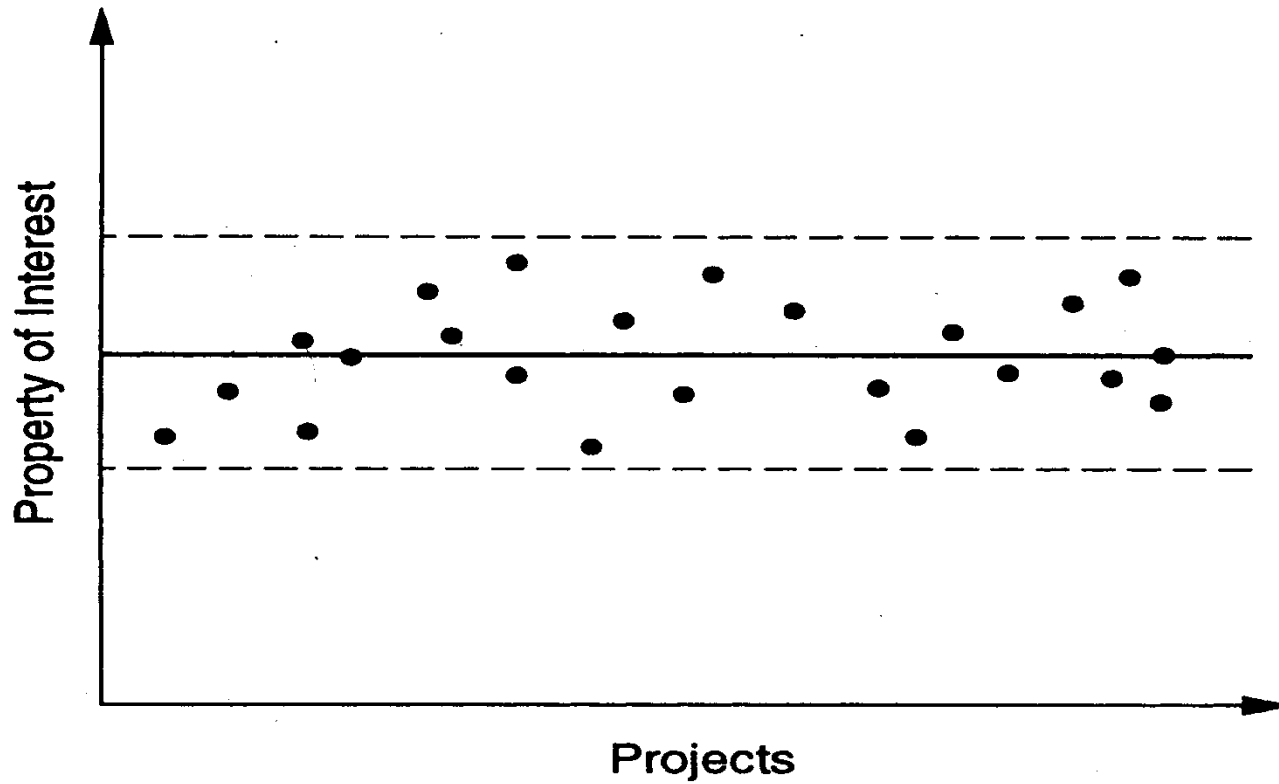
- Predictability and repeatability
 - Process should repeat its performance when used on different projects
 - I.e. outcome of using a process should be predictable
 - Without predictability, cannot estimate, or say anything about quality or productivity
 - With predictability, past performance can be used to predict future performance



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

Predictability...





Support Change

- Software changes for various reasons
- Requirements change is a key reason
- Requirement changes cannot be wished away or treated as “bad”
- They must be accommodated in the process for sw development



Summary

- Process – method for doing something
- Process typically has stages, each stage focusing on an identifiable task
- Stages have methodologies
- Software process is the methods for developing software
- Best to view it as comprising of multiple processes



Summary

- Goal is to produce software with high quality and productivity
- Process is the means
- Development process is central process
- Mgmt process is for controlling dev
- Other supporting processes
- Sw process should have high Q&P, predictability, and support for change



Development Process and Process Models



Software Project

- Project – to build a sw system within cost and schedule and with high quality which satisfies the customer
- Project goals – high Q and high P
- Suitable process needed to reach goals
- For a project, the process to be followed is specified during planning



Development Process

- A set of phases and each phase being a sequence of steps
- Sequence of steps for a phase - methodologies for that 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:
Requirements analysis, architecture, design, coding, testing, delivery
- Different models perform them in different manner



Requirement Analysis

- To understand state the problem precisely
- Forms the basis of agreement between user and developer
- specifies “ **what** ” , not “ **how** ”.
- Not an easy task, as needs often understood.
- Requirement specifications of even medium systems can be many hundreds of pages
- Output is the sw req spec (SRS) document



Design

- A major step in moving from problem domain to solution domain; three main tasks
 - Architecture design – components and connectors that should be there in the system
 - High level design – modules and data structures needed to implement the architecture
 - Detailed design – logic of modules
- Most methodologies focus on architecture or high level design
- Outputs are arch/des/logic design docs



Coding

- Converts design into code in specific language
- Goal: Implement the design with simple and easy to understand code.
 - Code should be simple and readable.
- The coding phase affects both testing and maintenance. Well written code can reduce the testing and maintenance effort.
- Output is code



Testing

- Defects are introduced in each phase
- They have to be found and removed to achieve high quality
- Testing plays this important role
- Goal: Identify most of defects
- Is a very expensive task; has to be properly planned and executed.
- Outputs are Test plans/results, and the final tested (hopefully reliable) code



Effort Distribution

- **Distribution of effort :**
 - Req. - 10-20%
 - Design - 10-20%
 - Coding - 20-30%
 - Testing - 30-50%
- Coding is not the most expensive.



Distribution of effort...

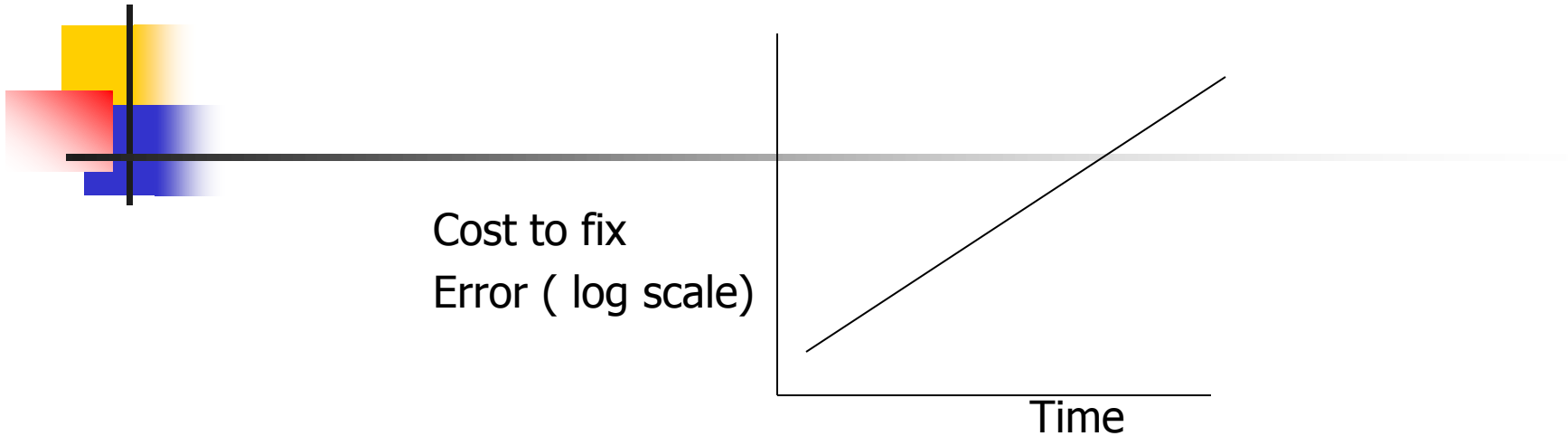
- How programmers spend their time
 - Writing programs - 13%
 - Reading programs and manuals - 16%
 - Job communication - 32%
 - Others - 39%
- Programmers spend more time in reading programs than in writing them.
- Writing programs is a small part of their lives.



Defects

- 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.



Process Models

- A process model specifies a general process, usually as a set of stages
- This model will be suitable for a class of projects
- I.e. a model provides generic structure of the process that can be followed by some projects to achieve their goals



Projects Process

- If a project chooses a model, it will generally tailor it to suit the project
- This produces the spec for the projects process
- This process can then be followed in the project
- I.e. process is what is actually executed; process spec is plan about what should be executed; process model is a generic process spec
- Many models have been proposed for the development process



Typical Student Process Model

- Get problem stmt – Code – do some testing – deliver/demo
- Why this process model cannot be used for commercial projects?
 - Produces student-software, which is not what we are after
 - Cannot ensure desired quality for industrial-strength software



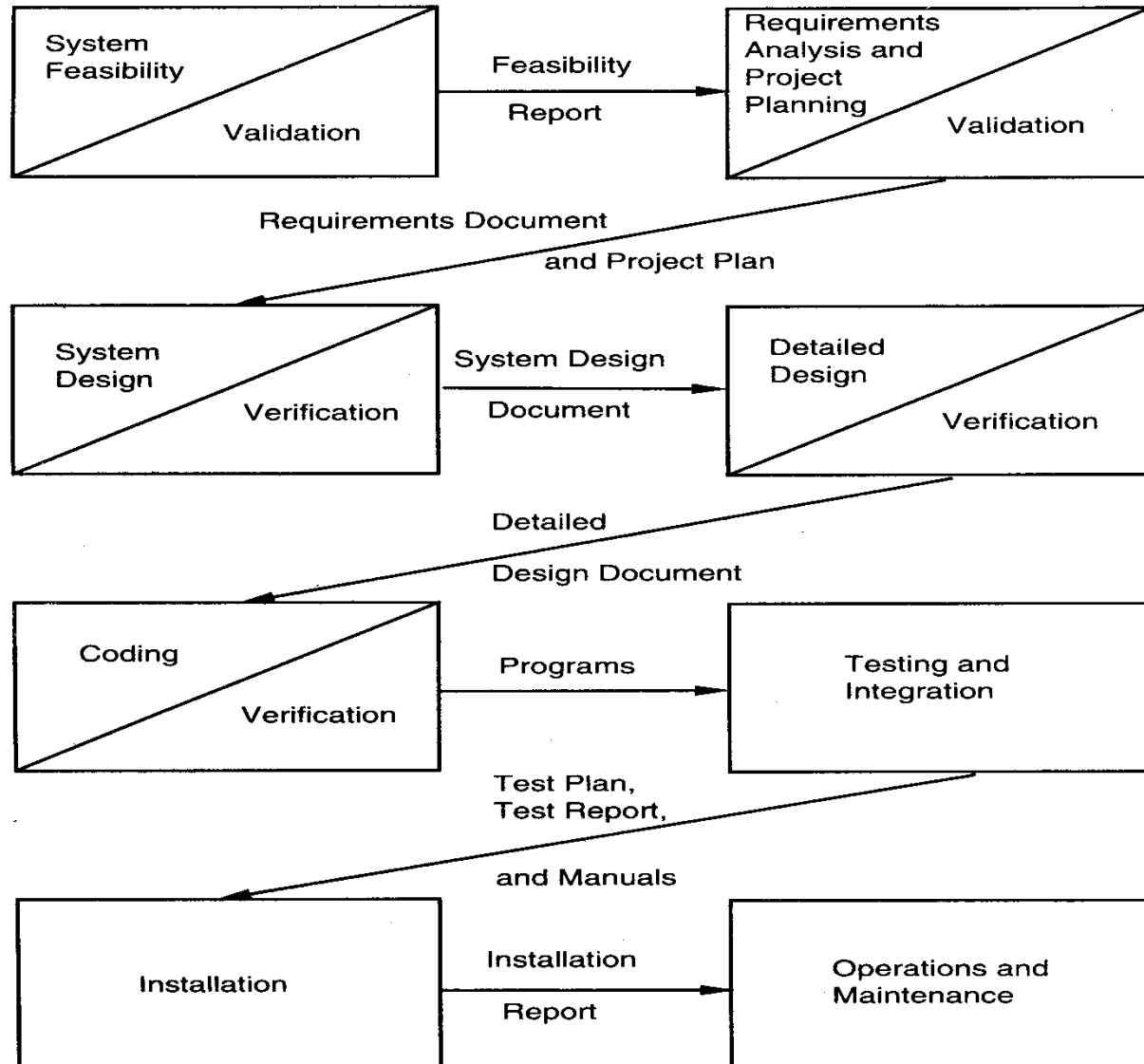
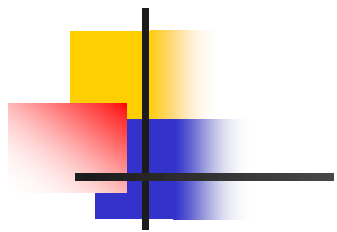
Common Process Models

- Waterfall – the oldest and widely used
- Prototyping
- Iterative – currently used widely
- Timeboxing



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





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

- Conceptually simple, cleanly divides the problem into distinct phases that can be performed independently
- Natural approach for problem solving
- 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

- Has been used widely
- Well suited for projects where requirements can be understood easily and technology decisions are easy
- I.e. for familiar type of projects it still may be the most optimum

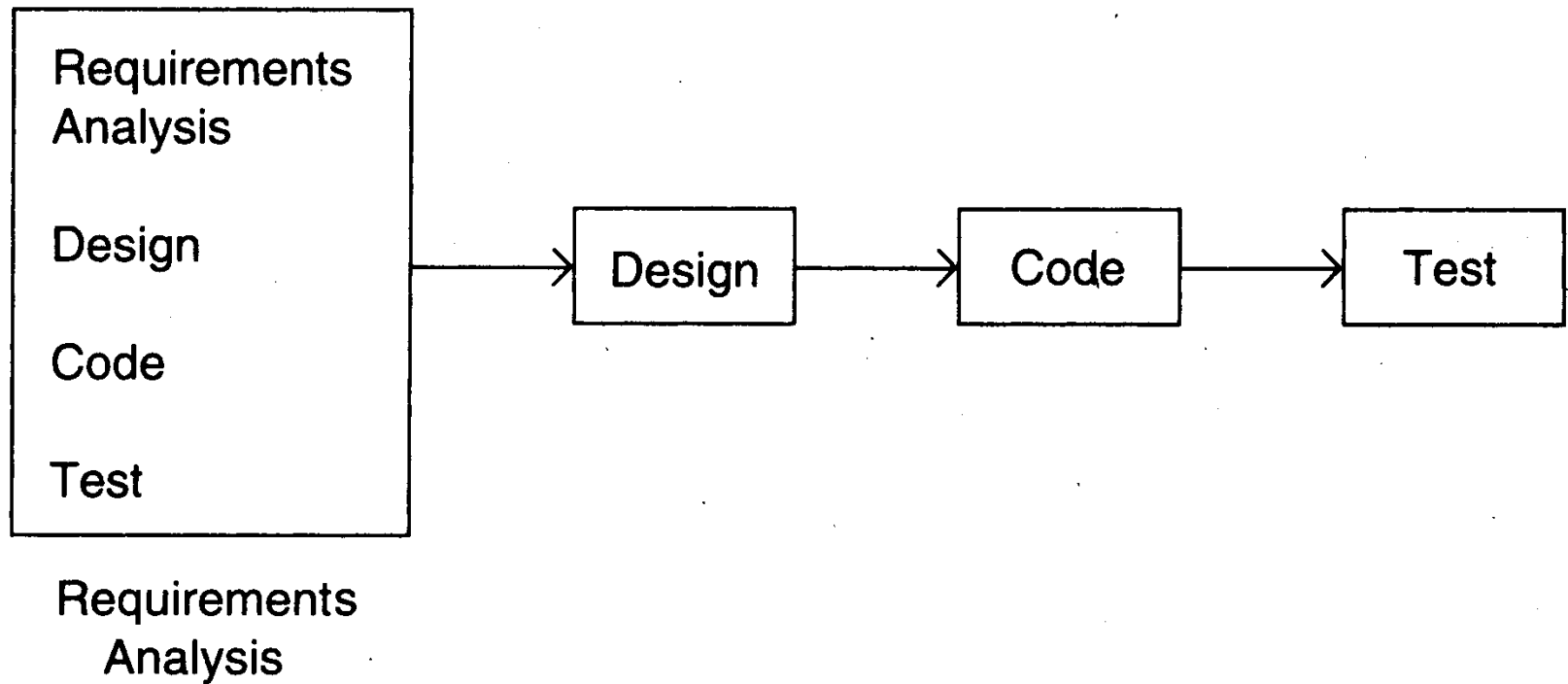


Prototyping

- 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





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: req will be more stable, req frozen later, experience helps in the main development
- Disadvantages: Potential hit on cost and schedule
- Applicability: When req are hard to elicit and confidence in reqs is low; i.e. where reqs are not well understood

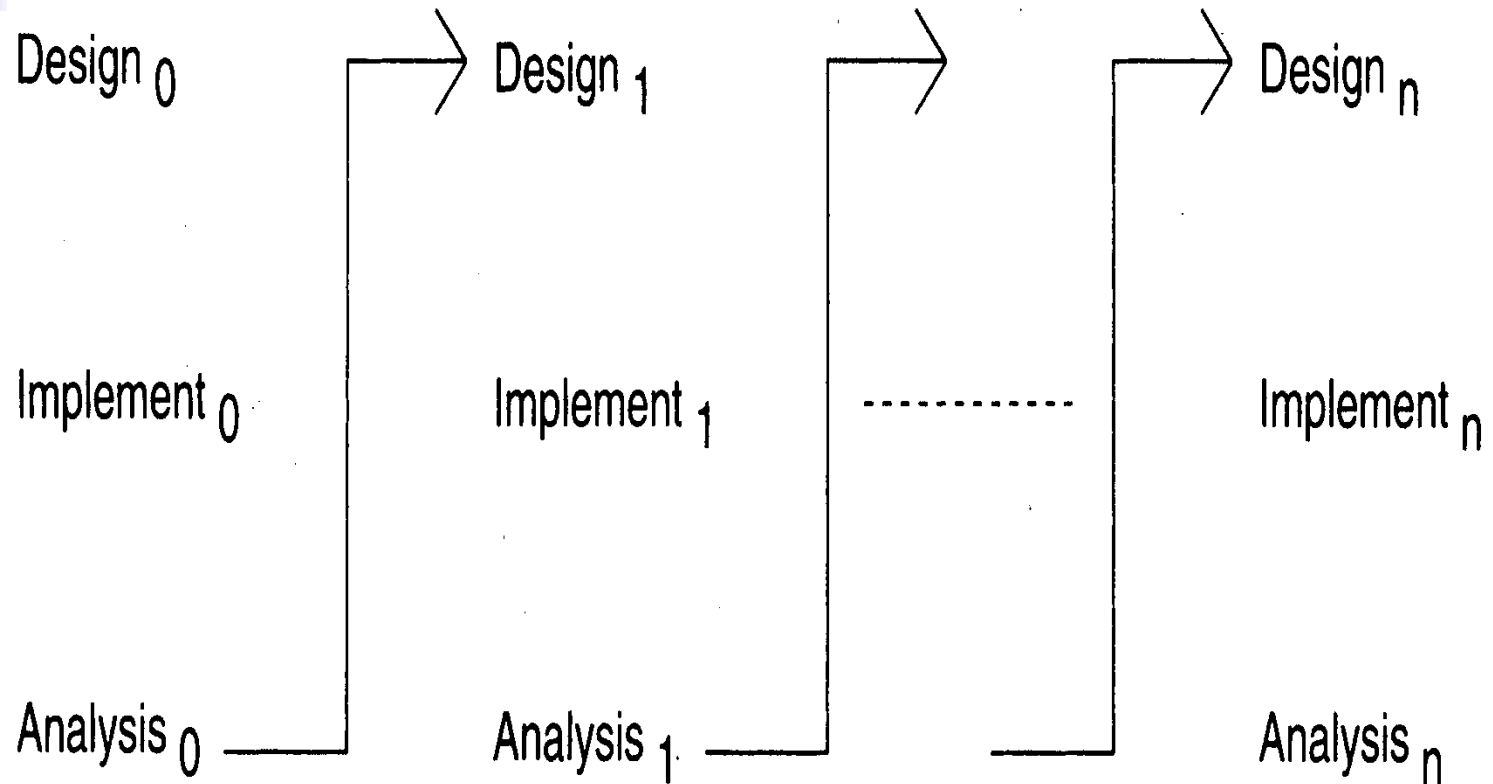


Iterative Development

- 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

- Products almost always 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, rework may increase, total cost may be more
- Applicability: where response time is important, risk of long projects cannot be taken, all req not known



Timeboxing

- Iterative is linear sequence of iterations
- Each iteration is a mini waterfall – decide the specs, then plan the iteration
- Time boxing – fix an iteration duration, then determine the specs
- Divide iteration in a few equal stages
- Use pipelining concepts to execute iterations in parallel



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



Time boxed Iteration

- 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 expanding
- Overall dev time is still unchanged



Timeboxing – Taking Time Boxed Iterations Further

- What if we have 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 Timeboxing Process Model



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



Timeboxing

- With this type of time boxes, can use pipelining to reduce cycle time
- Like hardware pipelining – view each iteration as an instruction
- As stages have dedicated teams, simultaneous execution of different iterations is possible



Example

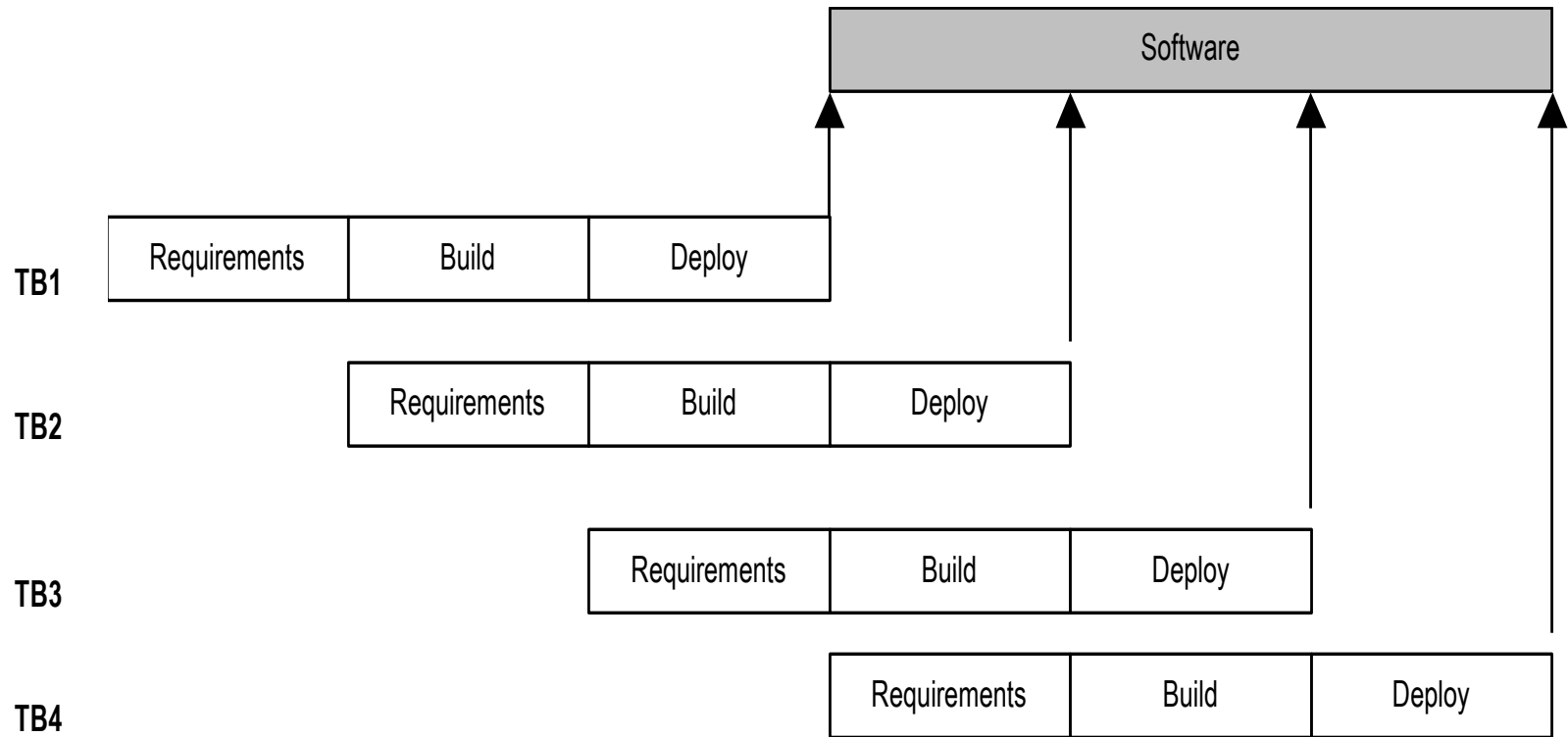
- An iteration with three stages – Analysis, Build, Deploy
 - These stages are appx equal in many situations
 - Can adjust durations by determining the boudaries suitably
 - Can adjust duration by adjusting the team size for each stage
- Have separate teams for A, B, and D



Pipelined Execution

- AT starts executing it-1
- AT finishes, hands over it-1 to BT, starts executing it-2
- AT finishes it-2, hands over to BT; BT finishes it-1, hands over to DT; AT starts it-3, BT starts it-2 (and DT, it-1)
- ...

Timeboxing Execution





Timeboxing execution

- First iteration finishes at time T
- Second finishes at $T+T/3$; third at $T+2T/3$, and so on
- In steady state, delivery every $T/3$ time
- If T is 3 weeks, first delivery after 3 wks, 2nd after 4 wks, 3rd after 5 wks,...
- In linear execution, delivery times will be 3 wks, 6 wks, 9 wks,...



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



Team Size

- In linear execution of iterations, the same team performs all stages
- If each stage has a team of S , in linear execution the team size is S
- In pipelined execution, the team size is three times (one for each stage)
- I.e. the total team size in timeboxing is larger; and this reduces cycle time



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



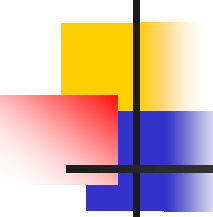
Timeboxing

- 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 v. imp.; architecture is stable; flexibility in feature grouping



Summary

- Process is a means to achieve project objectives of high QP
- 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 uncertainty. 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 Reduces risks	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