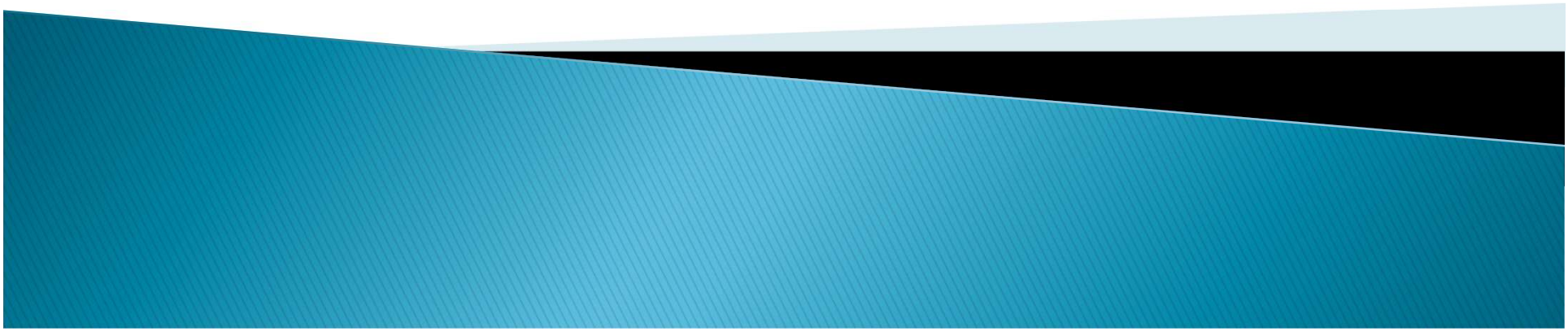
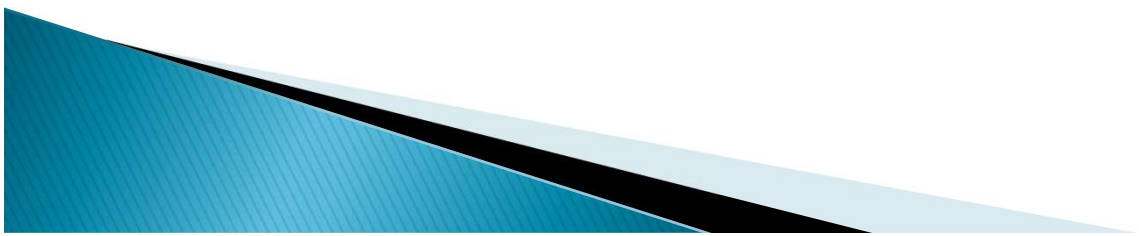
Unit-I

Introduction to Software Engineering



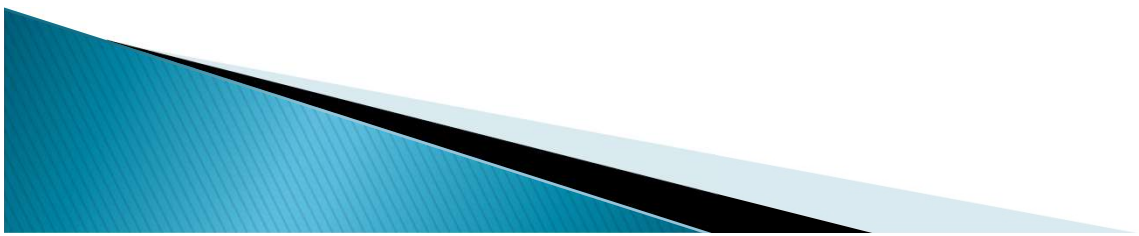
Software Engineering

- The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software
- Software engineering is defined as the systematic approach for development of (industrial strength) software



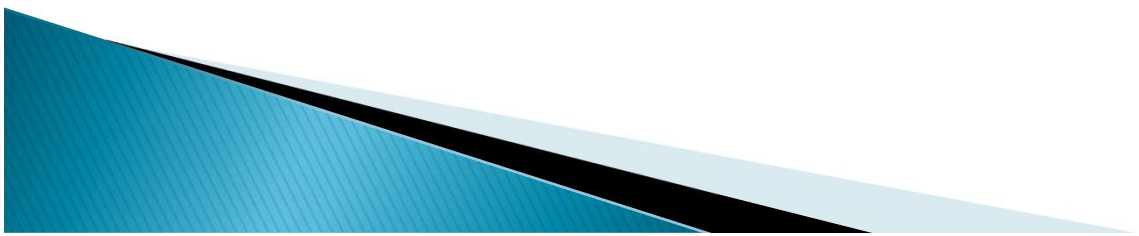
Software Process

- Process: A sequence of steps performed to achieve some goal
- Software Process: The sequence of steps performed to produce software with high quality, within budget and schedule
- Many types of activities performed by diff people in a software project
- Better to view software process as comprising of many component processes



Components of Software Process

- 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 engineering process
  - project managers execute management process



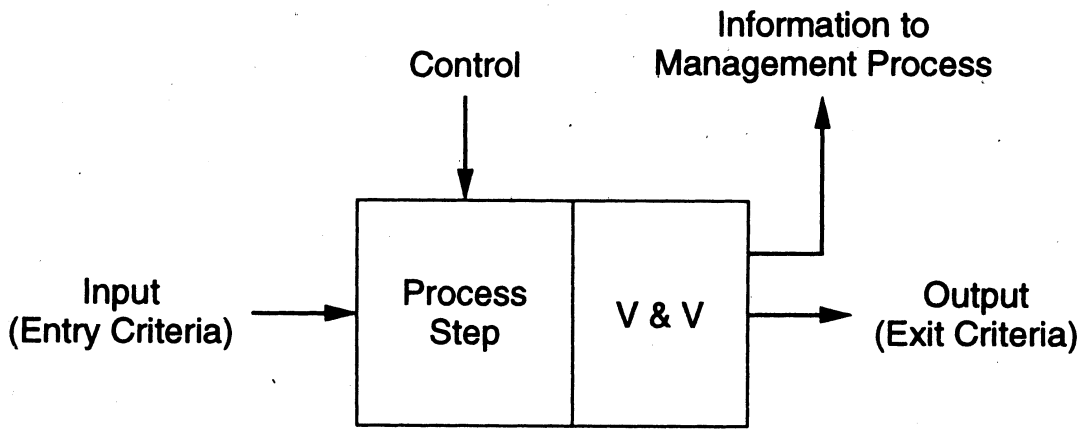
Process Specification

- Process is generally a set of phases
- Why have phases
  - To employ divide and conquer
  - each phase handles a different part of the problem
  - helps in continuous validation
- 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
- Methodologies (How to execute) of particular phases may be well defined

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 information for management

ETVX approach



General S/W Development Process

- Software development process commonly has these activities:
  - Requirements analysis
  - Design
  - Coding
  - Testing
  - Delivery
- Different models perform them in different manner

Waterfall Model

- Linear sequence of stages/phases
- Requirements – Design– Code – Test – Deploy
- A phase starts only when the previous has completed; no feedback
- The phases partition the project, each addressing a separate concern
- Each phase has output in the form of specific documents

Waterfall Model

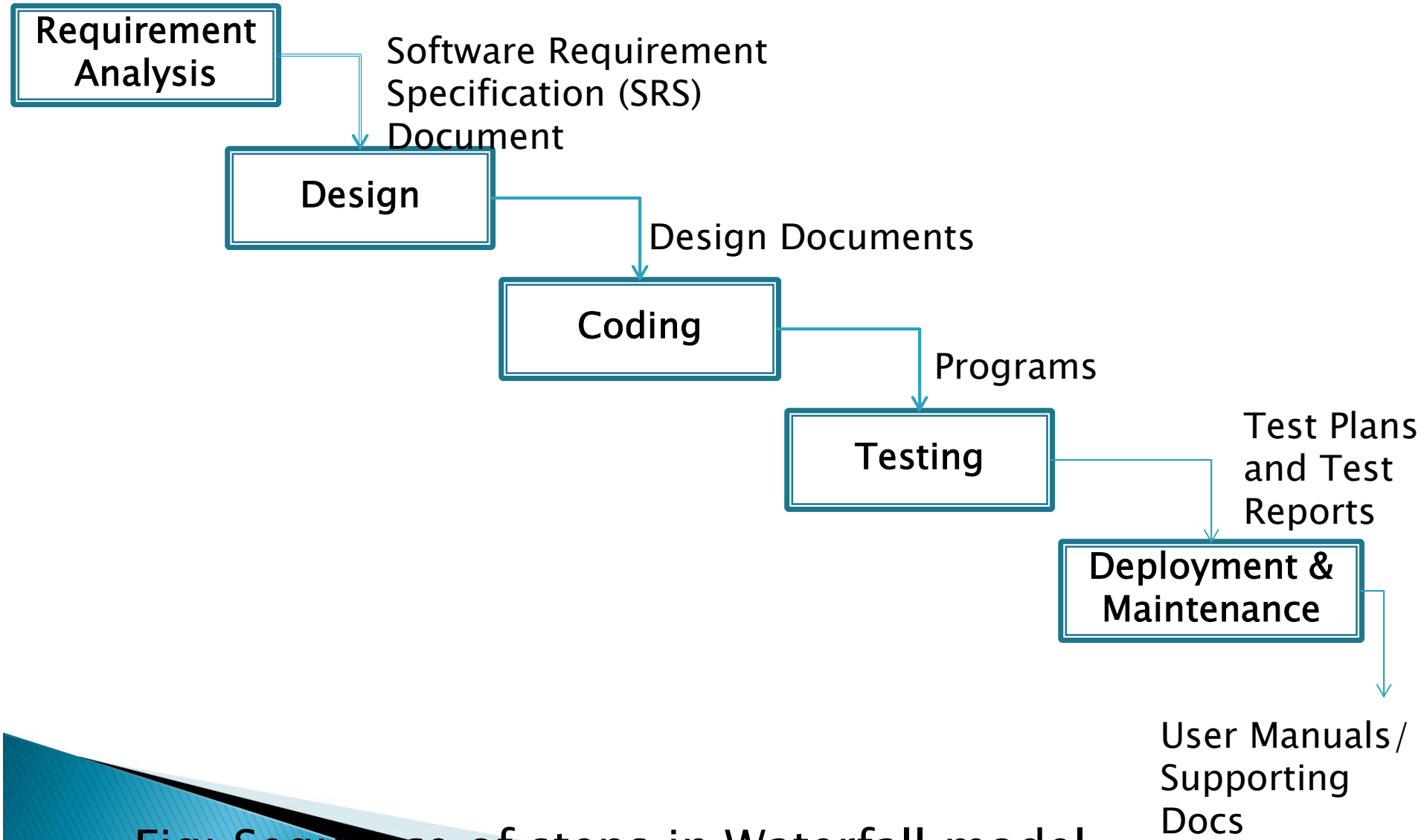
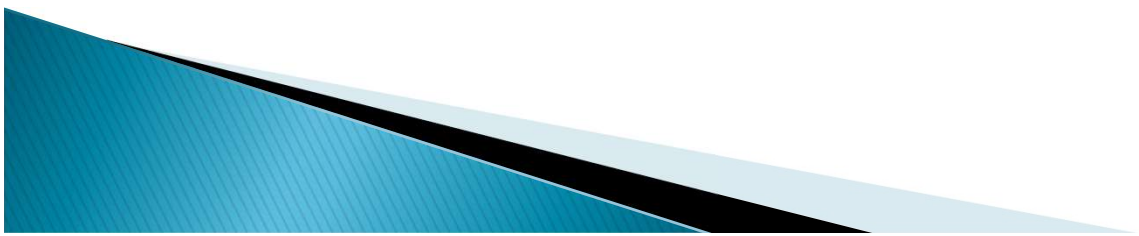


Fig: Sequence of steps in Waterfall model

Waterfall Model

- This model is easy to understand and reinforces the notion of “define before design” and “design before code”
- The model expects complete & accurate requirements early in the process



Waterfall Advantages

- Conceptually simple, cleanly divides the problem into distinct phases that can be performed independently
- Natural approach for problem solving
- Easy to administer – each phase is a milestone

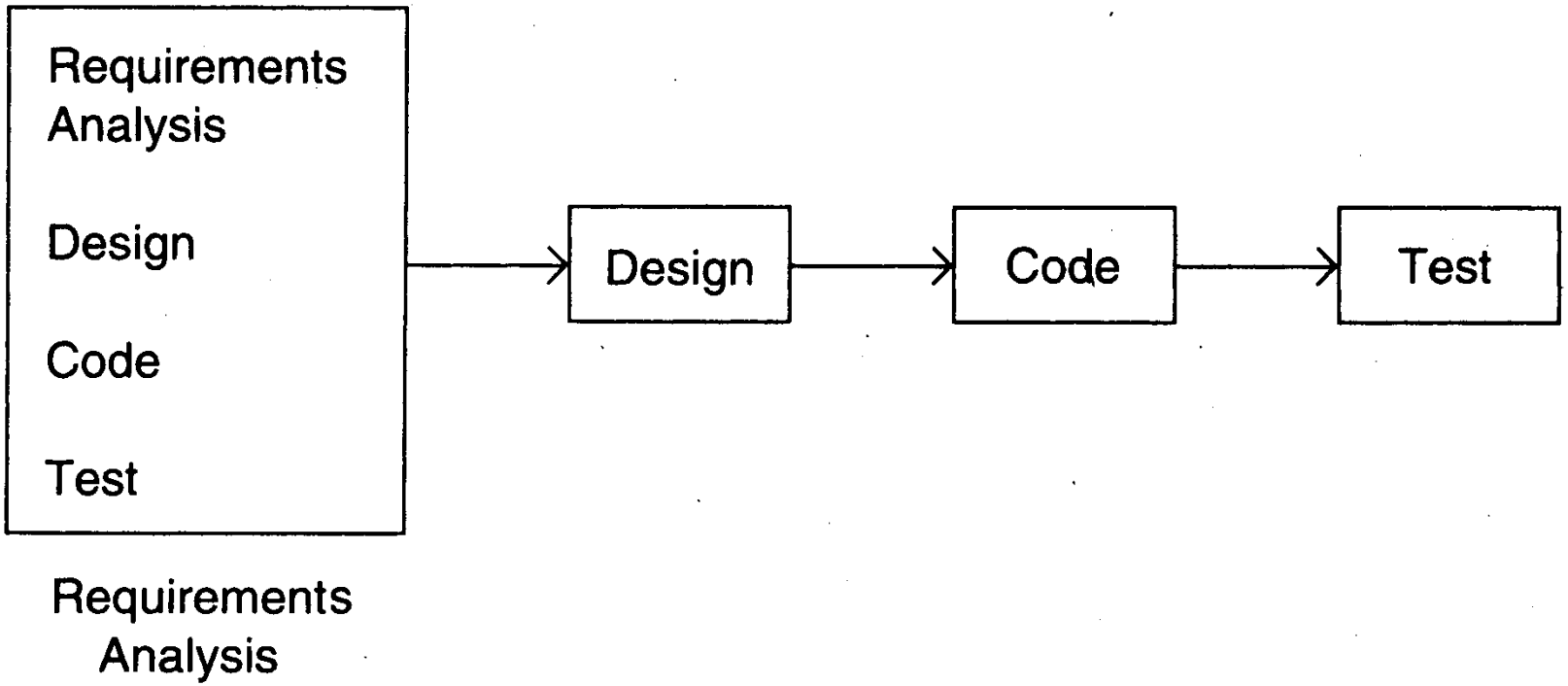
Waterfall disadvantages

- Assumes that requirements can be specified and frozen early which is unrealistic
- not suitable for accommodating any change
- 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
- Real projects are rarely sequential.

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 incorrect requirements risk
- A small waterfall model replaces the requirements stage

Prototyping



Prototyping

- Development of prototype
  - Starts with initial requirements
  - Only features which are unclear are included in prototype
  - Feedback from users taken to improve the understanding of the requirements
  - The code for the prototype is generally thrown away. However experience gathered helps in developing the actual system.

Prototyping

- The development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.
- “quick and dirty” – quality not important
- Things like exception handling, recovery, standards are omitted
- Learning in prototype building will help in building the software, besides improved requirements

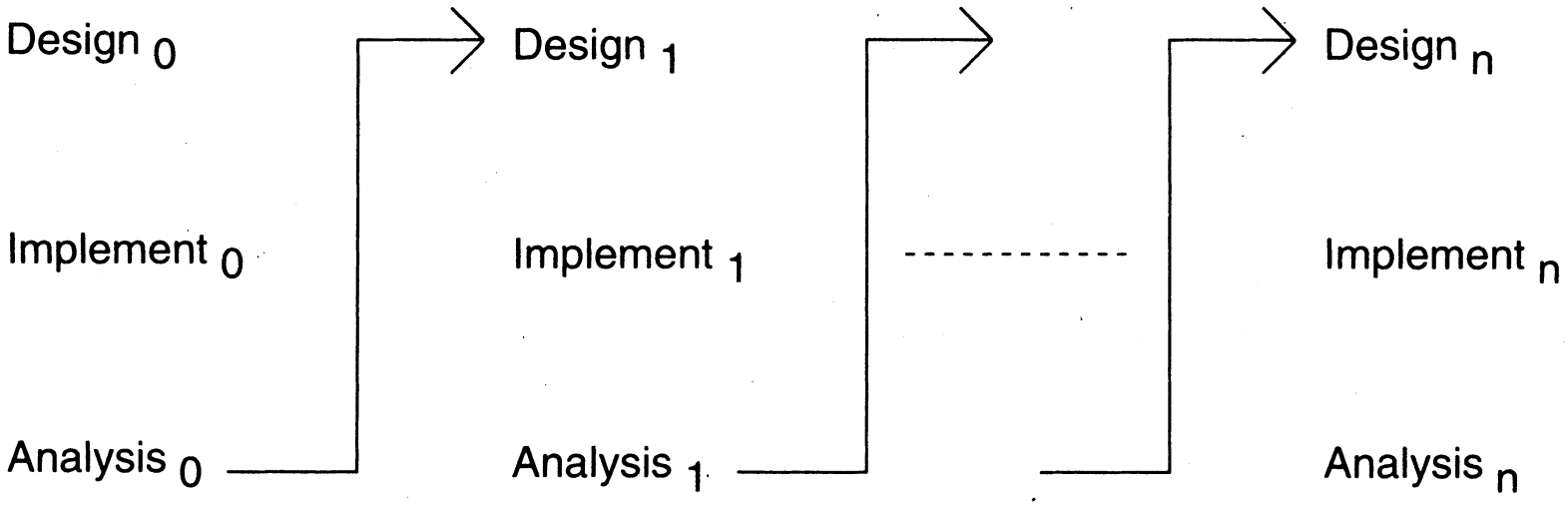
Prototyping

- Advantages
  - Allows flexible and unclear requirement, requirements frozen later
  - Experience helps in the main development
- Disadvantages
  - Potential hit on cost and schedule
- Applicability:
  - When requirements are hard to elicit and confidence in requirements is low; i.e. where requirements 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
- Newer approaches like XP, rely on iterative development

Iterative Enhancement



An iteration: A set of requirements is taken and development is done in the Waterfall way

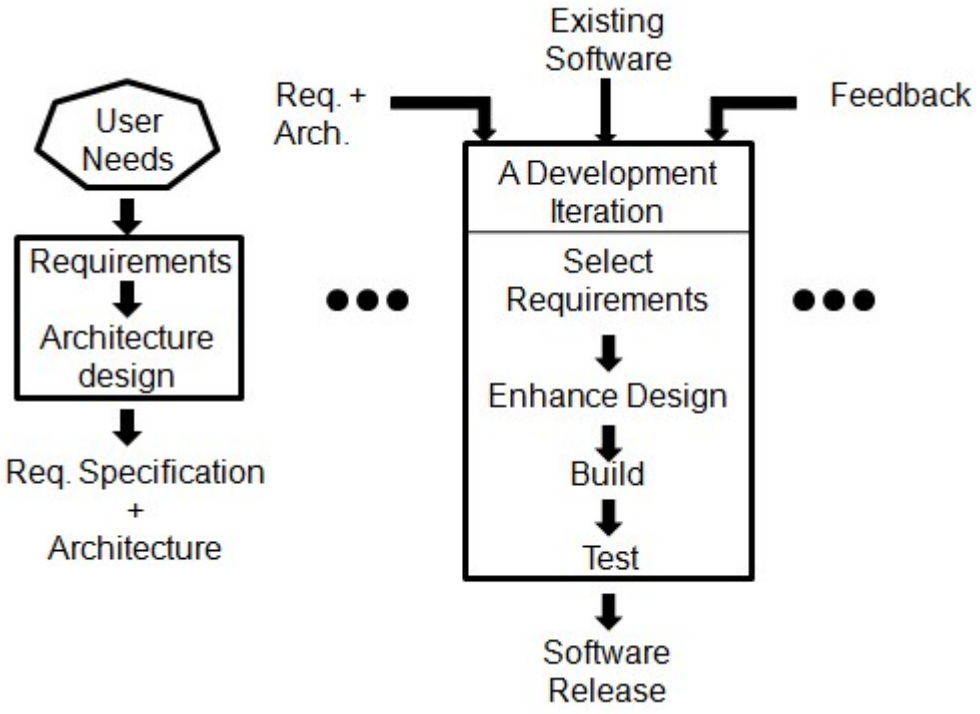
Iterative Development

- 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 requirements not known

Another Form of Iterative

- The first iteration does the requirements and architecture (Design) in the waterfall way (No iteration this far)
- The development and delivery is done incrementally in iterations
- From the set of completed requirements+design, select a set and develop it

Another form of Iteration…



Timeboxing

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

Timeboxing – Taking Time Boxed Iterations Further

- 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 is divided in fixed stages
- Each stage performs a clearly defined task that can be done independently
- Each stage is 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

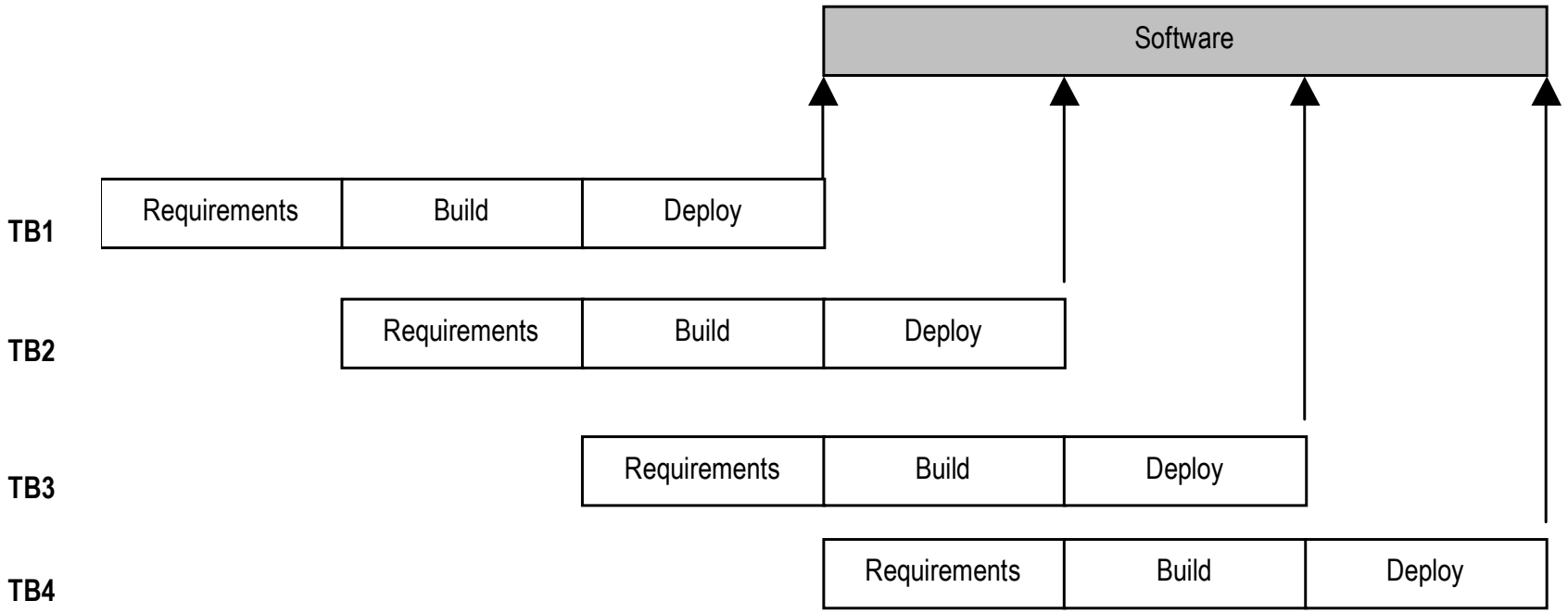
Example

- 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+2 T/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

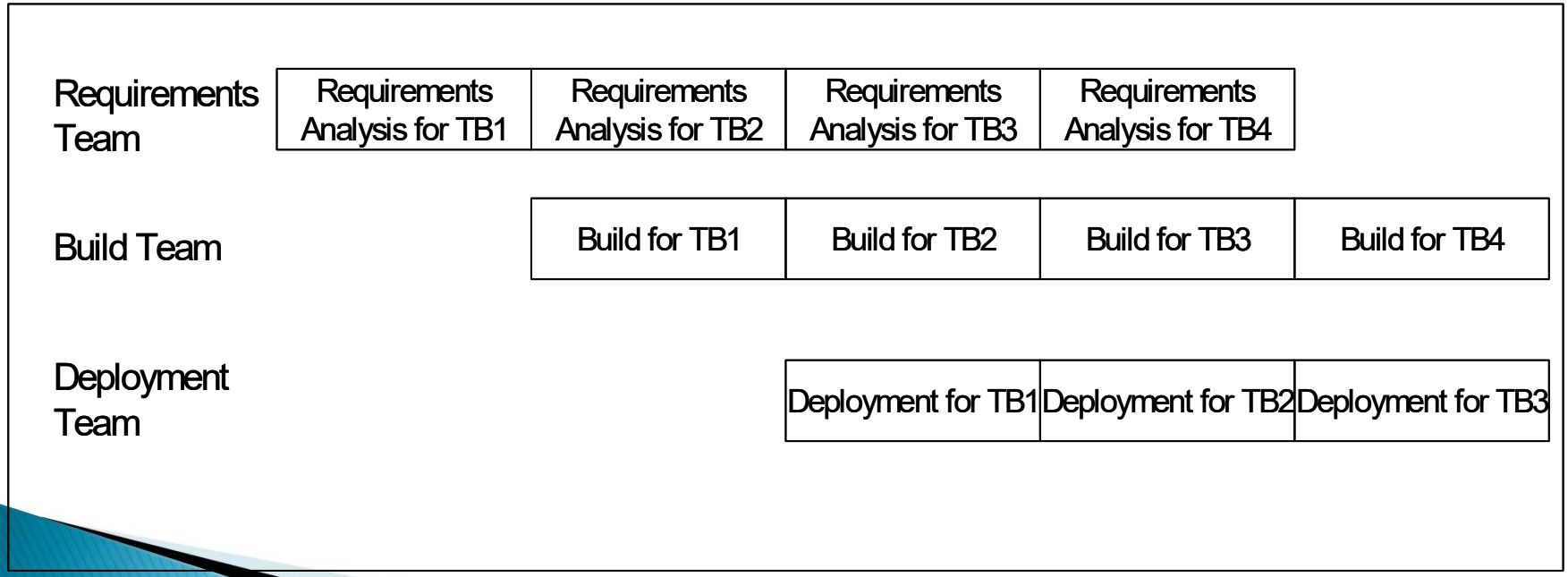
Team Size

- 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



Timeboxing

- Advantages
  - Shortened delivery times
  - other advantages of iterative
  - distributed execution
- Disadvantages
  - Larger teams, project management is harder
  - high synchronization needed
- Applicability
  - When delivery time is short
  - architecture is stable; flexibility in feature grouping

Agile Process Model

- Agile approaches developed in 90s as a reaction to document driven approaches
- Most agile approaches have some common principles
  - Working software is the measure of progress
  - Software should be delivered in small increments
  - Even late changes should be allowed
  - Prefer face to face communication over documentation
  - Continuous feedback and customer involvement is necessary
  - Prefer simple design

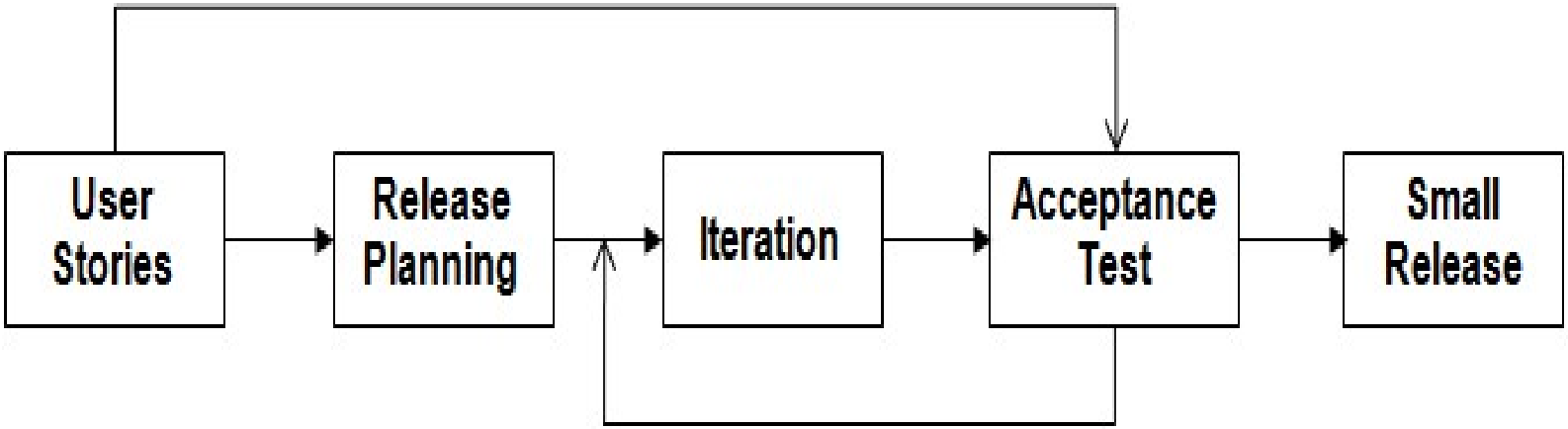
Extreme Programming (XP)

- Many agile methodologies have been proposed; extreme programming (XP) is one of the most popular
- Other agile methodologies are **Scrum, Lean, Kanban, Crystal, Dynamic Systems Development Method (DSDM), Feature-Driven Development (FDD)**
- An XP project starts with user stories, which are short descriptions of user needs
  - Details are *not* included
  - User stories can be combined in different ways

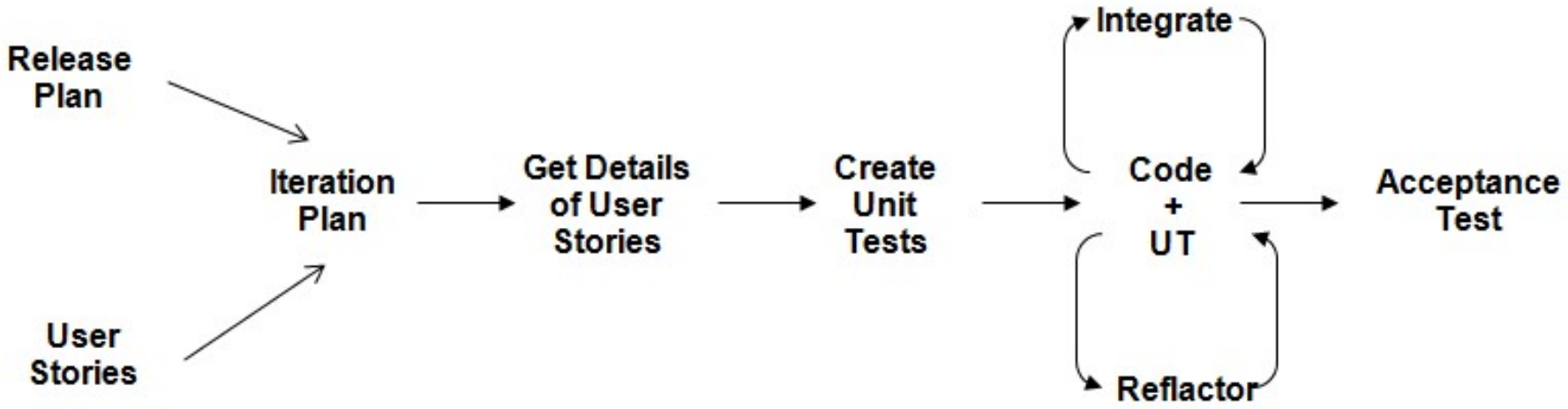
Overall Process

- Team estimates how long it will take to implement a user story
  - Estimates are rough
- Release planning is done
  - Defines which stories are to be built in which release, and dates for release
  - Frequent and small releases encouraged
  - Acceptance tests also built from user stories; used to test before release
  - Bugs found in AT are fixed in next release

XP – Overall Process



An Iteration



XP – Summary

- Well suited for situations where volume and pace of requirements is high
- Customer is willing to engage heavily with the team
- The team is collocated and is not too large (less than 20 or so)
- Requires strong capability in team members

Summary – waterfall

Strength	Weakness	Types of Projects
Simple Easy to execute Intuitive and logical	All or nothing – too risky Req frozen early Disallows changes No feedback from users	Well understood problems, automation of existing manual systems

Summary – Prototyping

Strength	Weakness	Types of Projects
Helps req clarification Reduces risk Better and more stable final system	Possibly higher cost and schedule Encourages requirement modification Disallows later change	Systems with novice users; or areas with req uncertainity.

Summary – Iterative

Strength	Weakness	Types of Projects
Regular deliveries, leading to business benefit Can accommodate changes naturally Allows user feedback Naturally prioritizes req 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;

Summary – Timeboxing

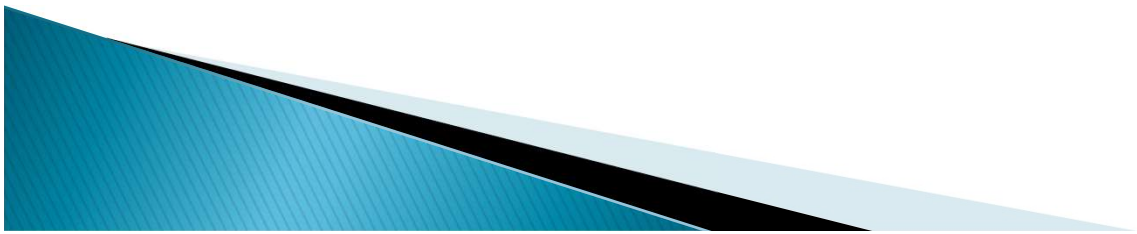
Strength	Weakness	Types of Projects
All benefits of iterative Planning for iterations somewhat easier Very short delivery times	Project Management becomes more complex Team size is larger Complicated – lapses can lead to losses	Where very short delivery times are very important Projects which can afford high costing (Multiple teams)

Summary – XP

Strength	Weakness	Types of Projects
Agile and responsive Short delivery cycles Continuous feedback can lead to better acceptance Reduced documentation	Can tend to become ad-hoc Lack of documentation can be an issue Continuous code change is risky	Where requirements are changing a lot, customer is deeply engaged in development, and where the size of the project is not too large

Rapid Application Development (RAD) Model

- Prototype based SDLC model
- Encouragement of code reuse, which means less manual coding, less room for errors, and shorter testing times
- High level of client engagement like other agile techniques



Rapid Application Development (RAD) Model

- Steps of RAD
  - Requirements (Flexible) Analysis
  - Design
  - Build a prototyope (Demonstrate to user and keep refining it as per user satisfaction)
  - Testing
  - Deployment