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.

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.

More models can be found on the following slides.

The waterfall model is a linear sequence of stages/phases. Requirements – System Design – Detailed 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.

Typical Student Process Model:
- Get problem statement – code – do some testing – deliver/demo
- Why this process model cannot be used for commercial projects:
  - Cannot ensure desired quality for industrial-strength software

Common Process Models:
- Waterfall – the oldest and widely used
- Prototyping
- Iterative – currently used widely
- Timeboxing
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 (Software Requirement Specification), 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
- 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

- 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

- 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

Advantages: requirements will be more stable and 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

- 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
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.

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.

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 – 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**

- 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 approximately equal in many situations
  - Can adjust durations by determining the boundaries 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**

- 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.

- Merely by increasing the team size we cannot reduce cycle time - Brook’s law.
  - Adding manpower to a late software project makes it later.
  - 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 of Teams

<table>
<thead>
<tr>
<th>Requirements Team</th>
<th>Build TB1</th>
<th>Build TB2</th>
<th>Build TB3</th>
<th>Build TB4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Team</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment Team</td>
<td>Deployment TB1</td>
<td>Deployment TB2</td>
<td>Deployment TB3</td>
<td>Deployment TB4</td>
</tr>
</tbody>
</table>

### Timeboxing

- Advantages: Shortened delivery times, other adv of iterative, distributive execution.
- Disadvantages: Larger teams, proj mgmt is harder, high synchronization needed.
- Applicability: When short delivery times very important; architecture is stable; flexibility in feature grouping.

### Summary

- Process is a means to achieve project objectives of high quality and productivity.
- 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

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
<th>Types of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>All or nothing – too risky</td>
<td>Well understood problems, short duration projects, automation of existing manual systems</td>
</tr>
<tr>
<td>Easy to execute</td>
<td>Requirement frozen early</td>
<td></td>
</tr>
<tr>
<td>Intuitive and logical</td>
<td>May chose outdated hardware.tech</td>
<td></td>
</tr>
<tr>
<td>Easy contractually</td>
<td>Disallows changes No feedback from users Encourages requirement bloating</td>
<td></td>
</tr>
</tbody>
</table>
### Summary – Prototyping

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
<th>Types of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helps requirement elicitation</td>
<td>Front heavy</td>
<td>Systems with novice users; or areas with requirement uncertainty.</td>
</tr>
<tr>
<td>Reduces risk</td>
<td>Possibly higher cost and schedule</td>
<td>Heavy reporting based systems can benefit from UI prototyping.</td>
</tr>
<tr>
<td></td>
<td>Encourages requirement bloating</td>
<td>Disallows later change.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Summary – Iterative

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
<th>Types of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help requirement elicitation</td>
<td>Regular deliveries, leading to biz benefit</td>
<td>For businesses where time is imp; risk of long projects cannot be taken; requirement not known and evolve with time.</td>
</tr>
<tr>
<td>Can accommodate changes</td>
<td>Can accommodate changes naturally</td>
<td></td>
</tr>
<tr>
<td>Encourages</td>
<td>Allows user feedback</td>
<td></td>
</tr>
<tr>
<td>requirement</td>
<td>Avoids requirement bloating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Naturally prioritizes requirement</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Allows reasonable exit points</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduces risk</td>
<td></td>
</tr>
</tbody>
</table>

### Summary – Timeboxing

<table>
<thead>
<tr>
<th>Strength</th>
<th>Weakness</th>
<th>Types of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>All benefits of iterative</td>
<td>Project Management becomes more complex</td>
<td>Where very short delivery times are very important.</td>
</tr>
<tr>
<td>Planning for iterations</td>
<td>Team size is larger</td>
<td>Where flexibility in grouping features is stable.</td>
</tr>
<tr>
<td>somewhat easier</td>
<td>Complicated - lapses can lead to losses</td>
<td></td>
</tr>
<tr>
<td>Very short delivery times</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>