Engineering

- The science concerned with putting scientific knowledge to practical use.
  
  Webster’s Dictionary

- Physics versus Electrical Engineering
Software Engineering

- The science concerned with putting computer science knowledge to practical use.

- Computer Science versus Software Engineering
Software Engineering - IEEE

1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

2. The study of approaches as in 1.

One of the largest efforts in Software Engineering has been the design of Ada Programming Language!
<table>
<thead>
<tr>
<th>No.</th>
<th>Activity</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirements</td>
<td>3.84%</td>
</tr>
<tr>
<td>2</td>
<td>Prototyping</td>
<td>4.50%</td>
</tr>
<tr>
<td>3</td>
<td>Architecture</td>
<td>2.25%</td>
</tr>
<tr>
<td>4</td>
<td>Project Plans</td>
<td>1.33%</td>
</tr>
<tr>
<td>5</td>
<td>Initial Design</td>
<td>3.84%</td>
</tr>
<tr>
<td>6</td>
<td>Detailed Design</td>
<td>4.50%</td>
</tr>
<tr>
<td>7</td>
<td>Design Reviews</td>
<td>3.02%</td>
</tr>
<tr>
<td>8</td>
<td>Coding</td>
<td>13.50%</td>
</tr>
<tr>
<td>9</td>
<td>Reuse acquisition</td>
<td>1.13%</td>
</tr>
<tr>
<td>10</td>
<td>Package Purchase</td>
<td>1.69%</td>
</tr>
<tr>
<td>11</td>
<td>Code Inspection</td>
<td>4.50%</td>
</tr>
<tr>
<td>12</td>
<td>Independent V&amp;V</td>
<td>5.42%</td>
</tr>
<tr>
<td>13</td>
<td>Configuration Mgt.</td>
<td>0.41%</td>
</tr>
<tr>
<td>14</td>
<td>Integration</td>
<td>2.71%</td>
</tr>
<tr>
<td>15</td>
<td>User Documentation</td>
<td>9.67%</td>
</tr>
<tr>
<td>16</td>
<td>Unit Testing</td>
<td>4.50%</td>
</tr>
<tr>
<td>17</td>
<td>Function Testing</td>
<td>4.50%</td>
</tr>
<tr>
<td>18</td>
<td>Integration Testing</td>
<td>3.84%</td>
</tr>
<tr>
<td>19</td>
<td>System Testing</td>
<td>3.38%</td>
</tr>
<tr>
<td>20</td>
<td>Beta Testing</td>
<td>3.02%</td>
</tr>
<tr>
<td>21</td>
<td>Acceptance Testing</td>
<td>1.94%</td>
</tr>
<tr>
<td>22</td>
<td>Independent Testing</td>
<td>3.38%</td>
</tr>
<tr>
<td>23</td>
<td>Quality Assurance</td>
<td>4.50%</td>
</tr>
<tr>
<td>24</td>
<td>Installation and Training</td>
<td>1.94%</td>
</tr>
<tr>
<td>25</td>
<td>Project Management</td>
<td>6.75%</td>
</tr>
</tbody>
</table>
Effort Breakdown of ~10000 Projects - Capers Jones

- Project Management: 8.08%
- Requirements: 14.43%
- Design: 11.36%
- Coding: 13.50%
- SQA: 30.64%
- SCM: 13.02%
- Integration: 6.54%
- Misc.: ~3%
Software Development

- Requirements
- Design
- Coding
- Testing
- Maintenance
- etc.

- project planning and management
- configuration management
- quality assurance
- installation and training
- etc.

Management

Construction
Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product
- implementing the product
- testing the parts of the product
- integrating the parts and testing them as a whole
- maintaining the product
The Four P’s of Software Engineering

- **Project** – the task at hand
- **People** – by whom it is done
- **Process** – the manner it is done
- **Product** – the artifacts produced
Well-Engineered Software

- Provides the required functionality
- Maintainable
- Reliable
- Efficient
- User-friendly
- Cost-effective
Well-Engineered Software - contd.

- These requirements may be conflicting:
  - Cost vs. Efficiency
  - Cost vs. Reliability
  - Efficiency vs. User-interface
- Law of diminishing returns.
- Challenge is to balance these requirements.
Software Life-Cycle Models

- The way you organize your activities
- The steps through which the product progresses
  - Requirements phase
  - Specification phase
  - Design phase
  - Implementation phase
  - Integration phase
  - Maintenance phase
  - Retirement
Software Processes

- Build-and-fix model
- Waterfall model
- Rapid prototyping model
- Incremental model
- Extreme programming
- Synchronize-and-stabilize model
- Spiral model
- Object-oriented life-cycle models
- Comparison of life-cycle models
Build and Fix Model

- Problems
  - No specifications
  - No design
- Totally unsatisfactory
- Need life-cycle model
  - “Game plan”
  - Phases
  - Milestones
Waterfall Model (contd)

- Characterized by
  - With or without feedback loops
  - Documentation-driven
- Advantages
  - Documentation
  - Maintenance easier
- Disadvantages
  - Client feedback
Rapid Prototyping Model

- Linear model
- “Rapid”
- Horizontal versus vertical prototyping
Three Key Points

- Do not turn into product
- Rapid prototyping may replace specification phase—never the design phase
- Comparison:
  - Waterfall model—try to get it right first time
  - Rapid prototyping—frequent change, then discard
Waterfall and Rapid Prototyping Models

- **Waterfall model**
  - Many successes
  - Client needs

- **Rapid prototyping model**
  - Cannot be used for robust applications

- **Solution**
  - Rapid prototyping for requirements phase
  - Waterfall for rest of life cycle
Incremental Model

- Divide project into builds
Incremental Model (contd)

- Waterfall, rapid prototyping models
  - Operational quality complete product at end
- Incremental model
  - Operational quality portion of product within weeks
- Less traumatic
- Smaller capital outlay, rapid return on investment
- Need open architecture—maintenance implications
- Variations used in object-oriented life cycle
Incremental Model (contd)

- Problems
  - Build-and-fix danger
  - Contradiction in terms
Incremental Model (contd)

- More risky version—pieces may not fit
  - CABTAB and its dangers
Code a bit, test a bit – CABTAB

- Haphazard undisciplined approach
Extreme Programming

- Somewhat controversial new approach
- Stories (features client wants)
- Estimate duration and cost of each story
- Select stories for next build
- Each build is divided into tasks
- Test cases for task are drawn up first
- Pair programming
- Continuous integration of tasks
Unusual Features of XP

- Computers are put in center of large room lined with cubicles
- Pair programming with shared computers
- Client representative is always present
- Cannot work overtime for 2 successive weeks
- No specialization
- Refactoring
Evaluating XP

- XP has had some successes
- Good when requirements are vague or changing
- Too soon to evaluate XP
Dilbert on eXtreme Programming

EXTREME PROGRAMMING

I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.

Copyright © 2003 United Feature Syndicate, Inc.
Synchronize-and Stabilize Model

- Microsoft’s life-cycle model
- Requirements analysis—interview potential customers
- Draw up specifications
- Divide project into 3 or 4 builds
- Each build is carried out by small teams working in parallel
Synchronize-and Stabilize Model (contd)

- At the end of the day—synchronize (test and debug)
- At the end of the build—stabilize (freeze build)
- Components always work together
  - Get early insights into operation of product
Spiral Model

- Simplified form
  - Waterfall model plus risk analysis

- Precede each phase by
  - Alternatives
  - Risk analysis

- Follow each phase by
  - Evaluation
  - Planning of next phase
If risks cannot be resolved, project is immediately terminated.
Full Spiral Model

- Radial dimension: cumulative cost to date
- Angular dimension: progress through the spiral
Full Spiral Model (contd)
Analysis of Spiral Model

- **Strengths**
  - Easy to judge how much to test
  - No distinction between development, maintenance

- **Weaknesses**
  - For large-scale software only
  - For internal (in-house) software only
Object-Oriented Life-Cycle Models

- Need for iteration within and between phases
  - Fountain model
  - Recursive/parallel life cycle
  - Round-trip gestalt
  - Unified software development process

- All incorporate some form of
  - Iteration
  - Parallelism
  - Incremental development

- Danger
  - CABTAB
Conclusions

- Different life-cycle models
- Each with own strengths
- Each with own weaknesses
- Criteria for deciding on a model include
  - The organization
  - Its management
  - Skills of the employees
  - The nature of the product
- Best suggestion
  - “Mix-and-match” life-cycle model
Quality Assurance?

- There is NO QA phase
- QA is an activity performed throughout software production
- Verification
  - Performed at the end of each phase
- Validation
  - Performed before delivering the product to the client
Documentation Phase?

- There is NO documentation phase
- Every phase must be fully documented before starting the next phase
  - Postponed documentation may never be completed
  - The responsible individual may leave
  - The product is constantly changing—we need the documentation to do this
  - The design (for example) will be modified during development, but the original designers may not be available to document it
<table>
<thead>
<tr>
<th>Phase</th>
<th>Documents</th>
<th>QA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>• Rapid prototype, or Requirements document</td>
<td>• Rapid prototype</td>
</tr>
<tr>
<td>Definition</td>
<td></td>
<td>• Reviews</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>• Specification document (specifications)</td>
<td>• Traceability</td>
</tr>
<tr>
<td>Specification</td>
<td>• Software Product Management Plan</td>
<td>• FS Review</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Check the SPMP</td>
</tr>
<tr>
<td>Design</td>
<td>• Architectural Design</td>
<td>• Traceability</td>
</tr>
<tr>
<td></td>
<td>• Detailed Design</td>
<td>• Review</td>
</tr>
<tr>
<td>Coding</td>
<td>• Source code</td>
<td>• Traceability</td>
</tr>
<tr>
<td></td>
<td>• Test cases</td>
<td>• Review</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Testing</td>
</tr>
<tr>
<td>Integration</td>
<td>• Source code</td>
<td>• Integration testing</td>
</tr>
<tr>
<td></td>
<td>• Test cases</td>
<td>• Acceptance testing</td>
</tr>
<tr>
<td>Maintenance</td>
<td>• Change record</td>
<td>• Regression testing</td>
</tr>
<tr>
<td></td>
<td>• Regression test cases</td>
<td></td>
</tr>
</tbody>
</table>
# Traceability matrix

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Use Case ID</th>
<th>UID</th>
<th>Class/function</th>
<th>Test Case ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions?
Thank you!