The Software Life Cycle

A Software Crisis

Denver International Airport (DIA)
Construction started in 1989 • 53 sq miles
• Planned: 1.7 bio USD costs, opening 1993

Code and Fix
(1950–)
Build first version
Modify until client is satisfied
Operate
Retirement
Code and Fix: Issues

- No process steps – no specs, docs, tests…
- No separation of concerns – no teamwork
- No way to deal with complexity

Waterfall Model
(1968)

Communication
- project initiation
- requirements

Planning
- estimating
- scheduling
- tracking

Modeling
- analysis
- design

Construction
- code
- test

Deployment
- delivery
- support
- feedback
Waterfall Model (1968)

Communication
- project initiation
- requirements gathering

Planning
- estimating
- scheduling
- tracking

Modeling
- analysis
- design

Construction
- code
- test

Deployment
- delivery
- support
- feedback

Deployment

QUESTIONNAIRE

- Very often
- Often
- Sometimes
- Rarely
**Waterfall Model**

**(1968)**

- Communication
  - project initiation
  - requirements gathering

- Planning
  - estimating
  - scheduling
  - tracking

- Modeling
  - analysis
  - design

- Construction
  - code
  - test

- Deployment
  - delivery
  - support
  - feedback

---

**Waterfall Model**

**(1968)**

- Real projects rarely follow a sequential flow
- Hard to state all requirements explicitly
- No maintenance or evolution involved
- Customer must have patience
- Any blunder can be disastrous

---

**Boehm’s first law**

Errors are most frequent during requirements and design activities and are the more expensive the later they are removed.

---

This and other laws are found in Endres/Rombach: Handbook of Software and Systems Engineering. Evidence: Several studies before 1974.
Problem Cost

- Relative cost of problem per phase

<table>
<thead>
<tr>
<th>Phase</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>0</td>
</tr>
<tr>
<td>Unit test</td>
<td>7.5</td>
</tr>
<tr>
<td>Component test</td>
<td>15.0</td>
</tr>
<tr>
<td>System test</td>
<td>22.5</td>
</tr>
<tr>
<td>Field</td>
<td>30.0</td>
</tr>
</tbody>
</table>

Incremental Model

- Each linear sequence produces a particular “increment” to the software
- First increment typically core product; more features added by later increments
- Allows flexible allocation of resources
Prototyping

- Communication
- Quick Plan
- Quick Design
- Prototype Construction
- Deployment and Feedback

Prototypes

Top Layer (GUI)

Vertical Prototype

Bottom Layer

Horizontal Prototype
Prototypes

• A horizontal prototype tests a particular layer (typically the GUI) of the system
• A vertical prototype tests a particular functionality across all layers
• Resist pressure to turn a prototype into a final result!
Spiral Model
(1988)

- System is developed in series of evolutionary releases
- Milestones for each iteration of the spiral
- Process does not end with delivery
- Reflects iterative nature of development
Inception

- Encompasses communication with user + planning
- Results in a set of use cases
- Architecture is just a tentative outline

Elaboration

- Refines and expands preliminary use cases
- Provides architecture and initial design model

Construction

- Builds (or acquires) software components according to architecture
- Completes design model
- Includes implementation, unit tests, acceptance tests
Transition

- Software given to end users for beta testing
- Feedback reports defects and changes
- Support information written

Production

- Software is deployed
- Problems are monitored

Re-Iteration

- Feedback results in new iteration for next release
Unified Process

- Draws on best features of conventional process models
- Emphasizes software architecture and design
- Integrates with UML modeling techniques (more on this later)

Extreme Programming
(1999–)

- Planning
- Design
- Test
- Coding
- Software Increment
Planning

- In XP, planning takes place by means of stories
- Each story captures essential behavior

Extreme Programming

- Design is made on the fly, using the KISS (keep it simple) principle
- Virtually no notation besides CRC cards (object sketches) and spike solutions (prototypes)
Extreme Programming

- Each story becomes a unit test that serves as specification
- The program is continuously refactored to have the design match the stories

Coding

- To ensure continuous review, XP mandates pair programming
Extreme Programming

- Planning
- Design
- Test
- Coding

Software Increment

Testing

- Planning
- Design
- Test

Unit tests
- detect errors
- find missing functionality
- measure progress

Software Increment

Extreme Programming

- Planning
- Design
- Test

Software Increment

- The resulting prototypes result in new stories
Extreme Programming is fast – with multiple deliverables per day!