Models [5]
Prototypes [2.3.1]
Process repetition [2.3.2, 2.4]
Agile processes [3]

Models

- Abstraction
  - remove detail
  - focus on viewpoint
- Dynamic
  - interaction with environment
  - control flow
  - data flow
- Static
  - physical
  - database schema
  - object structure

Why?
- Improve understanding
- all viewpoints
- "coathanger" - requirements refer to it
- validation

Warnings
- try to avoid design
- avoid massive documentation
**Traceability**

- Caller can erase message
- Stores up to 100 messages
- < 0.1% messages lost
- Requirements becomes model

**Prototypes**

- Used for:
  - Requirements elicitation
  - Requirements validation
  - Proof of concept
  - User training
  - Back to back testing
  
  \{ Risk reduction \}

**Warning**

- A prototype is not the product!
- It may differ in:
  - Functionality
  - Performance
  - Reliability
  - Scalability
  - Maintainability
  - User-interface, "finish"

**Throw away prototype**

- Explorer development
- Working system
- Known parts
- Grow into the system
- Product

**Risks**

- Not thrown away
- Too low quality prevents evaluation
- Bad structure
- Low process visibility
- Contractual problems

**Repetition in processes**

- Iterative
- Incremental
- Evolutionary

**Iterative process**

1. Develop
2. Requirements
3. Version 2
4. System
5. Version 1
6. Paper prototype
7. Evaluate
Iterative process

- Each iteration
  - functionality can be added
  - requirements can be changed

- Problem: when does it end?
  - budget
  - contract

Incremental (divide and conquer)

1. Requirements
   - Design, code, unit test
   - Test

2. Requirements
   - Design, code, unit test
   - Test

3. Requirements
   - Design, code, unit test
   - Test

4. Requirements
   - Design, code, unit test
   - Test

Advantages of incremental development:
- The most important parts are most tested
- Later increments benefit from more domain knowledge
- Can even out work load for specialists (in UCSD)
- After each increment there is a working (incomplete) system
  - could be delivered if money or time runs out
  - could be used for user training

Top-down vs. Bottom-up

- Traditional "from scratch" development
- Hierarchical system
  - "The Cathedral"

- Reuse based development
  - "network" system
  - component-based

  "The Bazaar"
Agile manifesto
We have come to value:

- Individuals and interactions
- Working Software
- Customer collaboration
- Responding to change
- Processes and tools
- Comprehensive documentation
- Contract negotiation
- Following a plan

Traditional vs. Agile

- Follow a plan
- Change costs
- Frozen requirements contract
- Documentation
- Deliverables at a deadline
- People
- Embrace change
- User stories, tests, customer involvement
- Working software
- Time-boxed
- smaller increments

<table>
<thead>
<tr>
<th>Agile principle</th>
<th>Scrum</th>
<th>Extreme Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental planning and development</td>
<td>Sprints</td>
<td>Implement user stories</td>
</tr>
<tr>
<td></td>
<td>Sprint backlog</td>
<td>Story cards</td>
</tr>
<tr>
<td></td>
<td>Planning poker</td>
<td>Planning poker</td>
</tr>
<tr>
<td>Customer involvement</td>
<td>Product owner</td>
<td>Customer representative in development team</td>
</tr>
<tr>
<td></td>
<td>Demo at end of sprint</td>
<td></td>
</tr>
<tr>
<td>People, not process</td>
<td>Scrum meetings</td>
<td>Pair programming</td>
</tr>
<tr>
<td></td>
<td>Sustainable pace (time-boxed)</td>
<td>Collective ownership of code</td>
</tr>
<tr>
<td>Embrace change</td>
<td>Change occurs from one sprint to the next</td>
<td>Continuous integration and release</td>
</tr>
<tr>
<td>Maintain simplicity</td>
<td>Refactoring</td>
<td>Refactoring</td>
</tr>
<tr>
<td></td>
<td>No anticipation of future requirements</td>
<td>No anticipation of future requirements</td>
</tr>
</tbody>
</table>