14 Initial development

• First stage in software lifespan
• Makes the fundamental decisions
• Consequences of these decisions will be reflected in the software throughout its life span.
Tasks of initial development

- Software plan
- Initial product backlog
- Initial design
- Implementation of the first version
Software plan

- Important management tool
- Overview of the entire software project before the start
- Serves managers to decide whether to start the project
- Estimates of the size of the project is an important component
Software Project Management Plan

• Introduction
  – Project overview
  – Project deliverables
  – Evolution of SPMP
  – Reference materials
  – Definitions and acronyms

• Project organization
  – Process model
  – Organizational structure
  – Organizational boundaries and interfaces
  – Project responsibilities

• Managerial process
  – Management objectives and priorities
  – Assumptions, dependencies and constraints
  – Risk management
  – Monitoring and controlling mechanisms
  – Staffing plan
SPMP cont...

• Technical process
  – Methods, tools and techniques
  – Software documentation
  – Project support functions

• Work packages, schedule and budget
  – Work packages
  – Dependencies
  – Resources requirements
  – Budget and resource allocation
  – Schedule

• Additional components
  – (subcontractor management plans, security plans, training plans, …)
Initial product backlog

• There are no current product users
• Future users usually consider the project too far in the future
• Participation of the users at this stage is lukewarm
Requirements elicitation

• Identify the stakeholders
  – future users, current or future project staff, current or future project managers, veterans from similar past projects, experts on the project domain

• Specialized techniques and methods
  – questionnaires, interviews, surveys, focus groups
Prototyping

- Quick implementation of the basic functionalities
- Emphasis on the user interface
- Demonstrates to the future users the look-and-feel
- Users test this prototype and comment
  - these comments become the requirements of the initial backlog.
Scope of the initial backlog

• Scope must be limited
• It must cover the first important decisions
  – conflicting considerations affect the selection of the requirements for the initial backlog.
  – user needs
    • requirements that are most central to the future users get the highest priority
  – project needs
Project needs

• Project risks
• Address the most serious risks early on
  – untried algorithm is to be used
  – the collaboration with outside software…
• Address the most risky aspects of the project in the initial development
  • there is still a time to resolve them
  • in extreme case to abandon the project before too many resources are wasted
Avoid expensive rework

• To postpone certain issues too far into evolution may mean that the evolution will require substantial rework
  – introduction of a project database too late in the project
  – transactions to be conducted with other already existing software, and so forth
Limited size of initial development

• After initial development, the programmers receive the first feedback from the other stakeholders
  – it should not be postponed too long
  – the longer this feedback is postponed, the larger the resulting rework may be

• The user and project needs increase the initial backlog
  – compromise must be found
Requirements creep

• Initial development can become big
• Example: London Ambulance System
  – ambitious
  – untried solutions that backfired
    • assignment of the nearest free ambulance to the call
• Initial development should deal with basics
  – remaining features should be postponed into evolution
Initial design

- Find classes
- Assign responsibilities
- Find dependencies
Find classes

• Extract the set of concepts used in the initial backlog
• Delete *insignificant* concepts
  – intended for the communication with the programmers
  – lie outside of the scope of the program
• *Significant* concepts become classes
• *Trivial* concepts become class members
Class responsibilities

• Each trivial concept is assigned to the most closely related class

• List of actions that are to be performed
  – extracted from the initial backlog
    • “order new items if supply runs low”
    • extracted assigned as a responsibility to a class
Find dependencies

• Some of the responsibilities are too large
  – classes need a help
  – contract with other classes
  – the other classes assume new responsibilities
    • that establishes dependencies among the classes

• Level of the detail in the design document
  – detailed description of contracts
  – just an associations among the classes
### CRC cards

<table>
<thead>
<tr>
<th>Class name: Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Responsibilities:</strong></td>
</tr>
<tr>
<td>Order more items</td>
</tr>
<tr>
<td><strong>Cooperations:</strong></td>
</tr>
<tr>
<td>Item</td>
</tr>
</tbody>
</table>
Phone directory

• The same initial backlog with the relevant concepts underlined is the following: A person's telephone number is found by searching a directory, which contains records of several persons. The user enters names and the program returns the respective phone numbers. The session is terminated by entering string "xxx".
<table>
<thead>
<tr>
<th>Concept name in backlog</th>
<th>Concept name in code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A person's telephone number</td>
<td>phoneNumber</td>
</tr>
<tr>
<td>Searching</td>
<td>search</td>
</tr>
<tr>
<td>directory</td>
<td>Directory</td>
</tr>
<tr>
<td>record</td>
<td>Record</td>
</tr>
<tr>
<td>user</td>
<td>UserInterface</td>
</tr>
<tr>
<td>enters</td>
<td>enterName</td>
</tr>
<tr>
<td>name</td>
<td>Name</td>
</tr>
<tr>
<td>return the phone numbers</td>
<td>returnPhone</td>
</tr>
<tr>
<td>terminated by entering string &quot;xxx&quot;</td>
<td>terminate</td>
</tr>
</tbody>
</table>
## CRC Cards

<table>
<thead>
<tr>
<th>Class name: Directory</th>
<th>Class name: UserInterface</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Responsibilities:</strong></td>
<td><strong>Responsibilities:</strong></td>
</tr>
<tr>
<td>search</td>
<td>main</td>
</tr>
<tr>
<td>list</td>
<td></td>
</tr>
<tr>
<td>addRecord</td>
<td></td>
</tr>
<tr>
<td>deleteRecord</td>
<td></td>
</tr>
<tr>
<td><strong>Cooperation:</strong></td>
<td><strong>Cooperations:</strong></td>
</tr>
<tr>
<td>Record</td>
<td>Directory</td>
</tr>
<tr>
<td>Name</td>
<td>Record</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class name: Record</th>
<th>Class name: Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Responsibilities:</strong></td>
<td><strong>Responsibilities:</strong></td>
</tr>
<tr>
<td>phoneNumber</td>
<td>enterName</td>
</tr>
<tr>
<td>returnPhone</td>
<td>terminate</td>
</tr>
<tr>
<td>putRecord</td>
<td>nameString</td>
</tr>
<tr>
<td></td>
<td>compare</td>
</tr>
<tr>
<td><strong>Cooperations:</strong></td>
<td><strong>Cooperations:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Implementation

• Writing code and testing it
• Bottom-up implementation
• Guided by the dependency graph.
  – first implement the classes that do not have any suppliers
  – then implement their clients, the clients of the clients, and so forth
• Verification by the test drivers
Dependency graph

Sequence:
Name, Record, List, UserInterface
Cycles in dependency graph
Skeletons of the code

• Design in UML class diagram
  – substantial overlap between the information in the design documents and the code
    • class names, class members, their arguments, etc.
    • can be directly copied into the code

• Tools that generate code skeletons
  • skeletons contain blanks that need to be filled in
  • skeletons save the programmers’ work.
Team organization

- Tasks of initial development have a very dissimilar character
  - different skills are required
- Different teams with different qualifications may participate in these different tasks
- Or the same team uses the help of different specialists
- Opportunity to recruit people into the team
Implementation

• Done iteratively
  – modified SIP, AIP, DIP, and CIP
  – Backlog consists of classes to be implemented
Transition to evolution

• Evolution fills in the missing requirements
• Reacts to the volatility that accumulated during the initial development
• Updated product backlog
Empty backlog after initial development

• Software is small, short-lived, or unusually stable domain
  – washing machine control program

• Waterfall model
  – there will be no evolution
  – next stage is servicing, phase-out, or close-down.