Process and Requirements

CMPUT 401—Module 02

Department of Computing Science
University of Alberta
Ken Wong, 2007

Software Life Cycle

- Questions:
  - what activities and tasks?
  - what dependencies between activities?
  - how to schedule activities?
  - standards?
    - e.g., IEEE 1074 (1995)

Software Life Cycle

<table>
<thead>
<tr>
<th>Process group</th>
<th>Activity</th>
<th>Task</th>
<th>Resource</th>
<th>Participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Phase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IEEE 1074

- **Quality assurance:**
  - verification
    - making sure you develop the system right (i.e., according to the requirements)
  - validation
    - making sure you develop the right system (i.e., what the customer really wanted)

- how?

Process Maturity

- **Notes:**
  - the standard specifies the complete set of activities, but does not define which should be chosen for a project
  - a software process is mature if the activities are defined and if management has some control over the management of the project

Capability Maturity

- **CMM levels:**
  - initial
    - ad hoc or chaotic, depends on heroism
  - repeatable
    - basic project planning, reviews
  - defined
    - process is documented, and institutionalized
  - managed
    - quality goals, measurement
  - optimizing
    - process itself changes over time

Life Cycle Models

- **Many variations:**
  - waterfall model
    - Royse (1970)
  - V model
    - Jensen & Tonies (1979)
  - sawtooth model
    - Rowen (1990)
  - spiral model
    - Boehm (1987)
  - ...
**Waterfall Model**

- Requirements Specification
- Architectural Design
- Detailed Design
- Coding and Unit Testing
- Integration and Testing
- Delivery and Installation
- Maintenance and Support

**Sawtooth Model**

- System Requirements Analysis
- Prototype Demonstration
- Detailed Design
- Coding and Unit Testing
- Integration and Testing
- Delivery and Installation
- Maintenance and Support

**V Model**

- System Requirements Analysis
- Software Requirements Elicitation
- Requirements Analysis
- Preliminary Design
- Detailed Design
- Implementation
- Unit Test
- Component Integration & Test
- System Integration & Test
- Client Acceptance
- Operation

**Spiral Model**

- Each cycle has phases:
  - define goals, alternatives, and constraints
  - evaluate alternatives, identify and resolve risks
  - develop and test prototype
  - plan next cycle
Spiral Model

- Cycles (inner to outer):
  - concepts
  - requirements
  - system design
  - detailed design
  - code
  - test
  - integration
  - acceptance
  - evolution

Unified Process

- Aspects:
  - use case driven
    - capture requirements from a user point of view, in natural language, with traceability
  - architecture-centric
    - organization of the whole system, including static and dynamic elements, connectors and protocols, overall architectural style, and quality constraints

Requirements

- Functional:
  - a feature that the system must have
    - e.g., interactions between end user and system
    - e.g., services provided by the system

- Non-functional:
  - a constraint that the system must satisfy to be accepted by a stakeholder
    - e.g., quality factors, accuracy, response time ...

Constraints

- Consider:
  - user interface and human factors
    - kind of interface, kind of expertise
  - documentation
    - level of product documentation
  - hardware, language, platform
    - compatibility requirements
  - performance characteristics
    - responsiveness, number of concurrent users
  - ...

**Constraints**

- Consider:
  - error handling and extreme conditions
  - how to handle exceptions, safety constraints
  - system modifications
    - anticipated scope of future changes
  - physical environment
    - where the system is deployed
  - security issues
    - protect against intrusions or malicious users
  - resource issues
    - constraints on resource consumption

**Requirements Elicitation**

- **Goal:**
  - specify the system in terms that the “users” can understand
    - who is your user? an end-user? a developer?
    - terminology issues
  - establish agreement between users and system developers
    - push versus pull
    - users can’t always express what they want

**Requirements**

- Requirements should be:
  - correct
    - requirements represent user view
  - complete
    - all possible scenarios are described
  - consistent
    - requirements do not contradict
  - clear
    - no ambiguities
  - realistic
    - can be done by mere mortals

**Requirement Elicitation**

- Also want:
  - traceability
    - implemented functionality and tests can be traced to a requirement
  - verifiability
    - a repeatable test can be designed to show that the system fulfills the requirement
Requirements Elicitation

- Verifiable?
  - The product shall have a *good* user interface.
  - The product shall respond to the user within 1 second for *most* cases.
  - The product shall be *error free*.

Requirements Elicitation

- Activities:
  - 1) identify actors
    - different types of entities to be supported
  - 2) identify scenarios
    - develop concrete scenarios for typical functionality to be provided
  - 3) identify use cases
    - define system scope with abstractions that describe all possible cases
  - 4) refine use cases
    - ensure completeness by considering potential exceptional conditions

Requirements Elicitation

- Bridging the gap:
  - scenario
    - an example of using the system in terms of interactions between the user and the system
  - use case
    - an abstraction that describes a class of scenarios
  - natural language, communication
  - user point of view

Requirements Elicitation

- Activities:
  - 5) identify relationships among use cases
    - consolidate and eliminate redundancies
  - 6) identify initial analysis objects needed
    - determine “business” objects, domain model
  - 7) identify non-functional requirements
    - determine qualities
Elicitation Activities

1) Identify actors:
   - what are the external entities? (human and non-human systems)

   ![Diagram of actors]

   - FieldOfficer
   - Dispatcher

2) Identify scenarios:
   - a scenario is a concrete, focused, informal description of a single feature from the viewpoint of a specific actor

   Example:
   - scenario name: warehouseOnFire
   - participating actor instances:
     - bob, alice: FieldOfficer
     - john: Dispatcher
   - flow of events:
     - Bob, driving down main street in his patrol car, notices smoke ...

Actors

- Sample questions:
  - Who is supported by the system?
  - Who administers the system?
  - What external hardware or software does the system interact with?
Scenario

- Sample questions:
  - What are the tasks that the actor wants the system to perform?
  - What info does the actor access? Who creates that data? Can it be modified or removed? By whom?
  - What external changes does the actor need to inform the system about? How often? When?
  - What events does the actor need to be informed of? With what latency?

Elicitation Activities

3) Identify and 4) refine use cases:

- more general than scenarios

- should be active voice, present tense, and actor point of view ...

Use Case

- Example:
  - use case name
    - ReportEmergency
  - participating actor
    - initiated by FieldOfficer
    - communicates with Dispatcher
  - entry condition
    - the FieldOfficer activates the “Report Emergency” function of her terminal
  - ...

- Example:
  - step-by-step flow of events
    - FRIEND responds by presenting a form ...
  - exit condition
    - the FieldOfficer receives the acknowledgement and the selected response
  - special requirements
    - the FieldOfficer's report is acknowledged within 30 seconds ...
Elicitation Activities

5) Identify relationships:
   - represent the flow of information
   - reduce or remove redundancies from the use case model, to eliminate potential inconsistencies

Relationship

- Communication:
  - between actors and use cases
  - start from initiating actor

Relationship

- extend relationships:
  - between use cases
  - for exceptional behavior

Relationship

- include relationships:
  - between use cases
  - for shared behavior, subtasks
Quality Factors

- Categories of factors:
  - product operation
    - i.e., software development
  - product revision
    - i.e., software maintenance
  - product transition
    - i.e., software migration

— McCall (1977)

Quality Factors

- Product operation:
  - correctness
    - extent to which the software performs its required function and meets the customer’s needs
  - reliability
    - extent to which the software can be expected to perform its function with the required precision
  - ...
Quality Factors

- Product transition:
  - portability
    - ease of effort needed to move the software from one platform to another
  - reusability
    - extent to which the software or its parts can be used in other applications
  - interoperability
    - ease of effort required to integrate one system to another

References

- Object-Oriented Software Engineering
  - B. Bruegge & A. Dutoit
  - Prentice-Hall, 2000
  - [b4, b12]