Afronding SO2

Voorwaarden:
• Voldoende serieuze deelname aan GiPHouse
• Aanwezigheid college, actieve deelname
Eindcijfer wordt bepaald aan de hand van:
• Eindverslag voor college met reflectie op
GiP activiteiten gerelateerd aan literatuur
(met name software ontwikkelings proces)
• Mondelinge evaluatie

College:

maandag, 13:45 - 15:30

1. 3 februari 2004 Sheets in pdf op WWW
2. 16 februari 2004 A0002
3. 8 maart 2004 A0002
4. 22 maart 2004 A0002
5. 5 april 2004 A0013
6. 19 mei 2004 A0013
7. 10 mei 2004 A0002
8. 24 mei 2004 A0002

Volgende College – 8 maart 2004

Alle 4 GiP projecten geven presentatie over:
• Globale idee requirements, stake holders,
  business case, actors, use cases
• Risico analyse
• Keuze software ontwikkelproces
  (hoe produkt maken en waarom zo)
Tijd: ca 10 minuten per project, daarna evaluatie
Example Iterative Development

Build embedded communication protocol following 7-layer ISO standard

- Application Layer: Application services
- Presentation Layer: Data encode/decode
- Session Layer: Organized extended dialogs
- Transport Layer: Connection-oriented and Connectionless transportation
- Network Layer: Packet routing
- Data Link Layer: Error free frame transmission, flow control
- Physical Layer: Encapsulate and abstract physical media characteristics

Example: develop early prototypes

ROPES process

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Representation</th>
<th>Basic?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements document</td>
<td>Test</td>
<td>Yes</td>
<td>A formal description of the system's constraints, interfaces to external actors, and externally visible behavior, including constraints and fairly requirements.</td>
</tr>
<tr>
<td>Use cases</td>
<td>Use case diagram</td>
<td>Yes</td>
<td>Identifications of the major functional aspects of the system and the interaction of actors with use cases.</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Sequence diagrams</td>
<td>No</td>
<td>Some sets of use cases will be recursive, i.e. have same behavior. The full behavior of the set of use cases may be captured in scenarios.</td>
</tr>
<tr>
<td>External Event List</td>
<td>A spreadsheet describing the properties of events excited by the system or sensed from the system. This includes properties such as period and jitter (for periodic events), minimum and maximum interevent times, and so on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Diagram</td>
<td>No</td>
<td>A hierarchical use of a UML object diagram that contains the system “object” and actors that interact with the system. The messages and events passing between actors and the system are identified.</td>
<td></td>
</tr>
<tr>
<td>Use case Scenarios</td>
<td>Sequence diagrams</td>
<td>Yes</td>
<td>As paths through individual use cases, these represent scenarios of use of the system. They show specific paths through a use case including messages sent between the use case and its associated actors.</td>
</tr>
<tr>
<td>Timing diagrams</td>
<td>No</td>
<td>As another representation of scenarios, timing diagrams are also instances of use cases. Normally applied only to reactive use cases, they show view along the vertical axis and linear time along the horizontal axis.</td>
<td></td>
</tr>
<tr>
<td>Hazard Analysis</td>
<td>No</td>
<td>This is normally a spreadsheet format document which identifies the key hazards for the system and their properties, such as fault tolerance time, severity and probability of the hazard, and its corrected risk.</td>
<td></td>
</tr>
<tr>
<td>Test Vectors</td>
<td>Test cases</td>
<td>Yes</td>
<td>Specification of tests to validate system against requirements.</td>
</tr>
</tbody>
</table>
Unified Modeling Language

Most current methods, including ROPES, use UML to describe various views on software using mainly visual notations.

Most methods contain some class diagram:

Development of the UML

the Unified Modeling Language

- ‘89-’94: more than 50 modeling languages for Object Oriented analysis and design. Including methods of Grady Booch, Jim Rumbaugh (OMT - Object Modeling Technique), Ivar Jacobson (OOSE - Object Oriented Software Engineering).

Development of the UML

the Unified Modeling Language

- Oct’94: at Rational Software, Grady Booch (Booch’93) and Jim Rumbaugh (OMT) start unification of their methods.
- Oct’95: draft 0.8 of Unified Method, Ivar Jacobson (OOSE) joins Rational, aiming at Unified Modeling Language.

Example

The ESU University wants to computerize the registration system

- The Registrar sets up the curriculum for a semester
  - One course may have multiple course offerings
  - Students select 4 primary courses and 2 alternate courses
  - Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
  - Students may use the system to add/drop courses for a period of time after registration
  - Professors use the system to receive their course offering rosters
  - Users of the registration system are assigned passwords which are used at login validation.
• An actor is someone or some thing that must interact with the system under development

• A use case is a pattern of behavior the system exhibits
  – Each use case is a sequence of related transactions performed by an actor and the system in a dialogue

• Actors are examined to determine their needs
  – Registrar -- maintain the curriculum
  – Professor -- request roster
  – Student -- maintain schedule
  – Billing System -- receive billing information from registration

Documenting Use Cases

• A flow of events document is created for each use case
  – Written from an actor point of view
• Details what the system must provide to the actor when the use case is executed
• Typical contents
  – How the use case starts and ends
  – Normal flow of events
  – Alternate flow of events
  – Exceptional flow of events

Maintain Curriculum -- Flow of Events

• This use case begins when the Registrar logs onto the Registration System and enters his/her password. The system verifies that the password is valid (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the professor to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
• If the activity selected is ADD, the S-1: Add a Course subflow is performed.
• If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.
• If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.
• If the activity selected is QUIT, the use case ends.

• Use case diagrams are created to visualize the relationships between actors and use cases

• Use case diagram presents an outside view of the system
• Interaction diagrams describe how use cases are realized as interactions among societies of objects
• Two types of interaction diagrams
  – Sequence diagrams
  – Collaboration diagrams
Sequence Diagram

- A sequence diagram displays object interactions arranged in a time sequence

Collaboration Diagram

- A collaboration diagram displays object interactions organized around objects and their links to one another

History of UML

- June’96: UML 0.9
- Oct’96: UML 0.91

UML development

- Jan. 1997: UML 1.0
- Sept. 1997: UML 1.1 with contribution from new partners, clarification semantics
- June 1999: UML 1.3
- 2001: UML 1.4, minor revision
  Added in 2002: Action Language
- 2004: UML 2.0, major revision

Almost all current CASE tools claim UML compatibility, e.g.

- Rational Rose, Rose Real-Time, ARTiSAN, Rhapsody,TogetherI, Argo UML

Many industries use UML and UML-based development

- Philips Medical System, Oce, Chess, ICT, Philips Semiconductors, NLR, ASML, …, web-development, …

UML tool support: Rational Rose

- Use case view: Use case diagram, Sequence diagram, State/activity diagram.
- Logical view: Class diagram, OCL, State diagram, Sequence diagram.
- Implementation diagrams: Component diagram, Deployment diagram.
UML-based development

Jacobson, Booch and Rumbaugh defined a unified software development process, the Rational Unified Process (RUP)

Basically, iterative, incremental development, similar to spiral model
Different from waterfall process

Waterfall Process Assumptions

- Requirements are known up front before design
- Requirements rarely change
- Users know what they want, and rarely need visualization
- Design can be conducted in a purely abstract space, or trial rarely leads to error
- The technology will all fit nicely into place when the time comes
- The system is not so complex

Waterfall Process Limitations

- Big Bang Delivery Theory
- The proof of the concept is relegated to the very end of a long singular cycle. Before final integration, only documents have been produced
- Late deployment hides many lurking risks:
  - technological (well, I thought they would work together...)
  - conceptual (well, I thought that's what they wanted...)
  - personnel (took so long, half the team left)
  - user doesn't get to see anything real until the very end, and they always hate it
  - system testing doesn't get involved until later in process

An Iterative Development Process...

- Recognizes the reality of changing requirements
- Caspers Jones’s research on 8000 projects:
  40% of final requirements arrived after the analysis phase, after development had already begun
- Promotes early risk mitigation, by breaking down system into mini-projects and focusing on the riskier elements first
- Allows to “plan a little, design a little, and code a little”
- Encourages all participants, including testers, integrators, and technical writers to be involved earlier on
- Allows to correct errors sooner and put into practice lessons learned in the prior iteration
- Focuses on component architectures, not final big bang deployments
An Incremental Development Process...

• Allows for software to evolve, not be produced in one huge effort
• Allows software to improve, by giving enough time to the evolutionary process itself
• Forces attention on stability, for only a stable foundation can support multiple additions
• Allows the system (a small subset of it) to actually run much sooner than with other processes
• Allows interim progress to continue through the stubbing of functionality
• Allows for the management of risk, by exposing problems earlier on in the development process

Goals and Features of Each Iteration

• The primary goal of each iteration is to slowly chip away at the risk facing the project, namely:
  – performance risks
  – integration risks (different vendors, tools, etc.)
  – conceptual risks (ferret out analysis and design flaws)
• Perform a “miniwaterfall” project that ends with a delivery of something tangible in code, available for scrutiny by the interested parties, which produces validation or correctives
• Each iteration is risk-driven
• The result of a single iteration is an increment—an incremental improvement of the system, yielding an evolutionary approach

The First Iteration

• The first iteration is usually the hardest
  – Requires the entire development environment and most of the development team to be in place
  – Many tool integration issues, team-building issues, staffing issues, etc. must be resolved
• Teams new to an iterative approach are usually overly-optimistic
• Be modest regarding the amount of functionality that can be achieved in the first iteration
  – Otherwise, completion of the first iteration will be delayed,
  – The total number of iterations reduced, and
  – The benefits of an iterative approach reduced

There Is No Silver Bullet

• Remember the main reason for using the iterative life cycle:
  – You do not have all the information you need up front
  – Things will change during the development period
• You must expect that
  – Some risks will not be eliminated as planned
  – You will discover new risks along the way
  – Some rework will be required; some lines of code developed for an iteration will be thrown away
  – Requirements will change along the way

Risk Management

“If you don’t actively attack the risks, they will actively attack you” – Tom Gilb

Identify risks:
• Technology risks
• People risks
• Organizational risks
• Tools risks
• Requirements risks
• Estimation risks

Risk analysis

Estimate for each risk:
• Probability, e.g. low, medium, high
• Impact, e.g. negligible, marginal, critical, catastrophic

If possible qualify impact of risk concerning:
  cost, performance, schedule, support

Overall risk, e.g. none, low, moderate, high

is derived from probability and impact
Risk Planning and Monitoring

Define strategies to deal with risks, e.g. reorganize, training, trace requirements, …
Next monitor the risks, discuss, re-plan, adapt strategies, etc.