How to build a UML model

RUP
Steriotypes, packages, and object diagrams
Case study

Announcements
- HW3 – Phase 1 due on Feb 6th, 5:00pm (need to create new pairs, accounts)
- Feedback on M2: turn procedural code into OO code, Planning game (show tables with features, subtasks, estimates, actuals, pair-programming partners)
- Register for the Feb 18 Industry Reception

Rational Unified Process
- Designed to work with UML
- No longer being promoted by IBM
- Roles - (out of 20 or so)
  - Architect
  - UI designer
  - Use case specifier
  - Use case engineer
  - Component engineer

How RUP builds a model
- Gather use cases from customer
- Make initial object model
- For each use case:
  - step through use case,
  - note the objects it requires
  - note the operations it uses
- Clean up the model

Architect
- Determine which use cases need to be developed first.
- High priority use cases
  - describe important and critical functionality
    - security
    - database
  - hard to retrofit later

UI design
- Logical design
  - Which user-interface elements are needed for each use case?
  - What information does the actor need to receive from or give to the system?
- Prototyping
  - Often is on paper.
  - Test on real users
Requirements Specification

- Not all requirements go in a use case.
  - Example: security
  - Example: global performance
- Requirements document describes all other requirements that are not suitable for use cases.

Analysis model

- Class diagrams
  - vague interfaces ("responsibilities")
  - vague associations (ignore navigability)
- stereotype classes:
  - boundary - UI, associated with actor
  - control - control associated with a use case
  - entity - persistent, the "real" objects
- Use-case realization (Analysis)

Stereotypes

- **Class diagrams**
  - vague interfaces ("responsibilities")
  - vague associations (ignore navigability)
- **stereotype classes**:
  - boundary - UI, associated with actor
  - control - control associated with a use case
  - entity - persistent, the "real" objects

Packages

- **Logical grouping**
  - divide large system into smaller subsystems
  - show dependencies between subsystems
- **Can contain**
  - class diagrams or packages
  - use cases, sequence diagrams, etc.

Packages and dependencies

- **Reduce coupling (so that teams can work independently)**
- **Increase cohesion**
  - In packages
    - cohesion is between classes in a package
    - coupling is between classes in different packages
  - In classes
    - cohesion is between methods in a class
    - coupling is between methods in different classes
**Architect**

- Responsible for the integrity of analysis model
- Makes sure packages fit together
- Makes sure each package is good
- Identifies obvious entity classes
- Lets other classes be defined during use-case realizations and component analysis

**Architect**

- Identify common special requirements
  - Persistence
  - Distribution and concurrency
  - Security
  - Fault tolerance
  - Transaction management

**Use case engineer**

- Identify analysis classes needed by use-case
  - Boundary classes, control classes, entity classes
- Distribute behavior of use-case to classes
- Make use-case realization: a precise description of use-case
  - sequence diagram
  - collaboration diagram

**Component Engineer**

- Analyze classes
  - Gather information from use cases
  - Make sure class is coherent
  - Make model as simple as possible, but no simpler.
- Analyze a package
  - Relationships between classes
  - Relationships between packages

**Outline of RUP process for analysis**

- Find use cases
- Architect determines order
- Repeatedly,
  - take next use case
  - change class diagram to accommodate use case
  - simplify class diagram

**Object diagram**

- Snapshot of objects in a system at a point in time
- If there is just one object of each class, the class diagram and the object diagram are the same
- As classes become more reusable, object diagram becomes more interesting
Class diagram

Class and object diagrams

Summary

Analysis/Design in XP

Modeling example in XP

Why we model incrementally?
The Viking

- A direct marketing system
- Sends customized mail and e-mail
- http://designfest.acm.org/Problems/Viking/Viking_00.pdf
- Description consists of a set of use cases

Generating letters

A user selects a set of customers to whom they wish to send letters and a template that defines the letter format. The Viking then generates a letter per customer that is based on filling in the “pluggable” information for the template with customer-related information. The user then previews the result of expanding the template list for each of the customers.

Template

Dear <<Proper Salutation>> <<Customer Name>>

Thank you very much from ordering from us on <<Date of most recent order>>. We recently received several thousand cans of the special ingredient, ...

If you are interested, please click on http://theHappyViking.com/SpecialOrder/<<special order number>>/Order.html
Use template to create a letter for a customer

letter := empty letter;
for each component c of template, add c.stringFor(customer) to letter

If c is a constant, c.stringFor() is c.value.

If c is a field, c.stringFor(customer) is customer.valueOf(c.name)

Sending letters

A user chooses from among the generated letters and decides which ones to send out and by which MailingSystem to send them. The Viking should already have information associated with each customer so it can properly distribute the letter by a particular MailingSystem; the user should not need to enter this information as part of the sending process.

Adding Customer Information

A user knows of a new customer that she wants to add to The Viking. The user can create a new customer entry and record relevant information (Name, Salutation, Address, Recent Purchase, and anything needed by other parts of The Viking) for that customer.
**Template Creation**

A user creates a new template either from scratch or by copying an existing template. A template needs to support both constant and “pluggable” information, and a user should be able to create a template and preview its appearance.

**User interfaces**

- Generate letters (select a set of customers and a template) and preview them
- Select letters to send and the mailing system to use
- Create new customer and enter info
- Create new template and preview it
Delivery Monitoring

A user reviews the letters to see which have been sent and whether any of them have “failed delivery”. If so, the user can choose to resend them either by the same Mailing System or by a different one. Also, to support this story, the Mailing System must be able to tell The Viking which sent letters “failed delivery”.

Mail delivery failure

- Some failures are permanent (“no such user”) and some are temporary (“mailbox full”). Need different status for each.
- Can have secondary addresses.
- “Select permanent failures and send to secondary address” is special case of “select messages and send them”
- Need UI for changing status

Customer selection by criteria

Upgrade the template from Story#1 to support selecting the set of customers by various search criteria. For example, select all customers who spent more than US$100 on their most recent order, or all customers who have ever bought a particular product.

- What query language?
  - SQL?
  - Dialog box?
- What criteria?
  - Any history?
  - Orders, products
- Change the UI for “select customers”
Conclusion

- Only half done
- Process to finish is the same
- Each step makes progress
- Keep track of open issues and make sure they get resolved
- Amount you write down depends on how much you remember

Next time

- Design Fest in class: CRC, UML