4 Software models

• Software is complex
  – often more than what people can handle
  – not necessary to know all details at all times
• Models offer simplified view
  – concentrate on the important issues and omit the clutter
• Different models are needed in different contexts
Roles of software models

• **Predictive (Designs)**
  
  • created up-front, before implementation starts
  • predict how the software will be like
  • helpful for resource planning and risk assessment

• **Extracted**

  • extracted from an existing system, by analysis of the properties of the software
  • help answer specific questions about the software
Roles of software models (2)

- Prescriptive
  - correspond to an existing system and its code
  - set of rules on how to evolve the software
  - software engineers must guarantee that the models remain valid after they change the code
  - define constraints that need to be preserved during evolution
Criteria for successful software modeling

• Need to understand which details are essential and which are not
  – the important properties need to be retained by the model

• Models that miss important information are misleading and can lead to mistakes
UML (Unified Modeling Language)

• Readable
  – serves as a communication between the customer and the developer
  – widely used

• Helps comprehension of the system
UML diagrams

• Structure diagrams
  – class diagrams
  – component diagram
  – package diagram
  – implementation diagram

• Behavior diagrams
  – activity diagram
  – sequence diagram
  – collaboration diagram
  – state diagram
  – use case diagram

• Supported by tools: Rational Rose, Visio, Violet …
UML class diagrams

• Structural diagrams
• Represent the classes in the system and their relationships
• Can have different levels of precision and completeness
• Independent of the software technologies
Class representation

• 3 compartments
  – class name
  – attributes (data members)
  – operations (function members)

• Supports different levels of detail
  – class name only

<table>
<thead>
<tr>
<th>Store</th>
<th>Inventory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-i : int</td>
</tr>
<tr>
<td></td>
<td>+update()</td>
</tr>
</tbody>
</table>
Association

• Associations are lines connecting class symbols

\[
\text{Store} \quad \text{Inventory}
\]

- \(i : \text{int}\)
- \(+\text{update}()\)
Adornments

- Adornments are at the ends of paths
  - arrow means navigation, i.e. direction
Generalization (inheritance)

- “Is-a” relationship
  - rectangle is a shape
  - ellipse is a shape
Generalization hierarchies

- Sensor
  - CalibratingSensor
    - currentValue()
    - calibrate()
  - HistoricalSensor
    - highestValue
    - lowestValue
    - resetHighest()
    - resetLowest()
  - TrendSensor
    - trend
  - TemperatureSensor
    - temp
    - currentTemp()
  - Barometer
    - pressure
    - currentPressure()
  - HumiditySensor
    - humidity
    - currentHumidity()
  - WindspeedSensor
    - speed
    - currentSpeed()
  - WindDirectionSensor
    - currentDirection()
Composite/component

• “Part-of” relationship
  – inventory is part of a store
class SaleLineItem
{
    private:
        int quantity;
        ProductDescription description;

    public:
        SaleLineItem(ProductDescription desc, int qty);
        int getSubtotal();
};
Activity diagram

- Activities are decomposed into actions

Activity diagram:

1. Find key
2. Unlock door
3. Enter

Toss coin
- [Head]: You win
- [Tail]: You lose

Check result
Swim lanes

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create midterm</td>
<td>Take midterm</td>
</tr>
<tr>
<td>Grade midterm</td>
<td>Read graded midterm</td>
</tr>
</tbody>
</table>

© 2012 Václav Rajlich
Software Engineering: The Current Practice  Ch. 4
Class dependency graphs (CDG)

• Depict classes and their dependencies

• Different from UML class diagrams
  – in small but significant details

• Extracted from the existing code

• Used during software evolution
Class responsibilities

• Each class in the program plays a role
  – assumes a certain responsibility
  – class Item is responsible for items sold in the store

• Supplier class
  – helps class Item to fulfill its responsibility
  – class Price is supplier class of Item
Clients, suppliers, dependencies

• Class B helps class A to fulfill its responsibilities:
  – B is supplier of class A
  – A is client of class B
  – there is dependency of class A on class B
    • denoted as (A,B)
Dependency examples

• If there is a part-of relation between classes A and B, there is also a dependency (A,B)
• If there is a non-polymorphic inheritance of X from Y, then (X,Y) is a dependence
• If there is a polymorphic inheritance between X and Y, then both (X,Y) and (Y,X) are dependencies.
Definition of CDG

• directed graph $G = (C,D)$
  – vertices $C$ are classes of the program
  – edges $D$ are dependencies
Supplier slice

• Set of all suppliers, suppliers of suppliers, ...

• Let \((C,D)\) be a CDG
  – \(A \in C\) be a class
  – then supplier slice \(S(A) = \{X \mid X = A \text{ or there is a dependency } <Y, X> \in D \text{ such that } Y \in S(A)\}\)
Class responsibilities, cont.

• *Local responsibility*
  – class assumes it alone and unaided

• *Composite responsibility*
  – assumed by the whole supplier slice
  – class itself may be unable to implement everything that is expected from it
  – it delegates some of the responsibilities to its suppliers
Contracts

• Composite responsibilities are sometimes expressed by *contracts*
  – between a class and its clients
  – clients request something
  – they have to make the request within reasonable bounds
    • they cannot request items that the store does not sell

• Bounds are described by the *precondition*

• Results of the action is *postcondition*
Examples of contracts

• Contract of “Average_calculator”
  – precondition: A non-empty list of student names and grades
  – postcondition: the grade point average

• Contract of “Checking_withdrawal”
  – precondition: amount to be withdrawn W, current balance in the account B, 0 ≤ W ≤ B
  – postcondition: the printed check, the new balance in the account.
Ariane 5 disaster

- Ariane is a series of rockets
- European space program
- Ariane 5 crashed after launch in 1996
- Loss approximately half billion dollars
- Cause: software error
Ariane 5 software error

• Inertial Reference System (IRS)
  – IRS is a supplier to the rest of the software
  – deals with “horizontal bias”
  – precondition: horizontal bias must fit within 16-bit integer
  – IRS was used in the earlier versions of Ariane

• Clients of IRS in Ariane 5 violated this precondition

• Boooom!!!!
Ariane 5
Formality of contracts

• Contracts are described by formal logic
  – special assertion language

• Plain English
  – verbal description of contracts

• Contracts are only tacit
  – not recorded anywhere
  – all programmers must rediscover them
    • misunderstandings and errors
    • very common practice
    • not recommended
CDG and UML

• CDG and UML class diagrams are similar
• Software engineers sometimes use the more popular UML class diagrams instead of CDG
  – they should make sure that the diagram contains all dependencies
  – direction of dependencies may be missing in UML, have to be filled in!