CS3505/5020
Software Practice II

Software process overview
Sequence diagrams
Software Processes

● What is software process?
  – Process is the area of software engineering concerned with organizing software development
    » Process also known as method, methodology

● What is the goal of a software process?
● Why do we care about a software process?
● What are some kinds of processes that you know about?
Fundamentally, to develop software you must do what things?

- Requirements – gather what it is that the program is supposed to do
- Architecture – figure out the “shape” of the application
- Design – Specify how the program is to be implemented
- Implement – build it
- Test – make sure that it works both in terms of unit up through matching the requirements
- Deployment – package it up and send it to the customer

Each of the software processes do these, but vary in how they do them:

- Waterfall – do each part completely and in order
- Iterative – do a little bit of each at a time
Software Process - 3

- Other things involved in process
  - Management
  - Planning, estimating, organizing
  - Process engineering
  - Source code/document/configuration control
  - Software change management
  - Metrics and measurements
  - Reuse of libraries and components
  - Documenting - for user, developer, maintainer, reviewer

- What kind of process do you do at your software job?
Software Process - 4

- Process is a very touchy and controversial topic
  - Which belies its importance. How we develop DOES make a difference
    » Some claim that it is THE difference.
- There is no “one right process”
  - RUP, XP, ... process must be applied, and that takes skill and experience
- But, there are lots of good practices and ideas out there
- Software Development is very uncertain and risk prone compared to most other engineering feats
  - Thankfully, we aren’t exposed to the physical dangers of failure
  - But software is more and more important
  - And projects can fail spectacularly
    » See “Software Runaways” (Robert L. Glass) for some true horror stories.
Software process choices depend on goals
  - No one process suits every project
  - Easy to become ‘religious’ about a process
    » Software development is changing too rapidly
  - Process should facilitate goals, not facilitate software process!
Sequence Diagrams

- Goal is to show interactions between objects
  - Object as in class instance
  - Also can show actors, participants in a more general sense
  - Often is used to help understand what is happening in a ‘use case’ (defined next week)
  - Not precise enough to show activity inside of an object

- As with the other UML diagrams, can show as little or as much detail as you like
Participants

- **Actors**
  - Taken directly from use case

- **Roles**
  - Anonymous instance of a class
    - can also be specific instance
  - Box notation
  - Just use the name

- **Objects**
  - Box notation
  - Underline name
  - You will see:
    - Object : Class (either optional)

- **Participants drawn at top of diagram, left to right**
  - Extended downward from middle of participant is lifeline
    - A dashed line indicating activity of the participant
  - Time flows from top to bottom
**Messages**

- The interaction between objects is called a message
  - Line with an arrow head
  - Several different kinds of messages
    » See table
    » Does Object creation and found/lost make sense? – Maybe

- Lines horizontal
  - Some use diagonal lines to indicate that it takes a LONG time

- Returns are optional
  - But you should use them when you need to improve clarity

- To call yourself (self-call) just loop back to your lifeline
  - Most of the message types are possible

---

**Message Types**

- Asynchronous
- Method call
- Object Creation
- Return
- Found/Async
- Found/Call
- Lost/Async
- Lost/Call
- Self-Call
Message Labels

- Text placed above the message line
- Labels are optional
  - But mostly they are used
- Describes the method or operation being invoked
  - Can include arguments or types
  - UML 1.x allowed inclusion of a guard (in square brackets)
    » Guard = Boolean statement that indicates the message will happen if the guard evaluates to true
    » For example: [balance <> 0]
    » UML 2 uses a “combined fragment” (see later)
- Sometimes label includes a sequence number
  - Helps make it clear ordering of messages
- Returns can be labeled to describe what is being returned
Activation

- An activation is a long rectangle placed on top of the life line
  - Indicates when the object is active
    - That is, it is “on the stack”
- Activations are optional, but often make it very clear what is going on
  - When drawing on the board I usually leave them off
- If some kind of recursion happens, you nest the activations
- You may leave off the return if you are using activations
  - Because it is “obvious” when the return happens
Frame

- Rectangle with a name box in the upper left hand corner
- Newly added in UML 2.0
- Graphical boundary for diagrams
  - A visual border
  - Is also used a lot internally to sequence diagrams
    » It is a “combined fragment”
  - Can use frame boundary to indicate messages to and from the internal diagram
    » See “ref” combined fragment later
The problem with guards is that they could only be used on a single message

- What if you wanted to have several messages that were “guarded”?
- What you did in UML 1.x was duplicated the guard
  » We know that duplication of a single idea is a bad idiom

Solution – combined fragment

- A frame with a specific name
- Opt – optional (same as a guard)
- Alt – alternative (if then else type of statement)
- Loop – a way of indicating n occurrences
Combined Fragment Examples

- **opt**\[balance <> 0\]
- **alt**\[balance < 0\]
- **alt**\[balance < 100.00\]
- **alt**\[balance >= 100.00\]

**While loop**

---

CS 3505
**ref Combined Fragment**

- Innovation in 2.0 is the ref combined fragment
  - Allows you to compose sequence diagrams
  - Essentially “call” a sequence diagram

- Use ref as the keyword

- Draw messages in and out of the frame

- Text on the frame indicates information about the referenced sequence diagram
Creating and Deleting Objects

- Sometimes you wish to indicate object creation and deletion
- Use the object creation message and point at an object box
- Activation often extends directly from box
- Use X to indicate object deletion
  - Either by self-destructing (lifeline termination)
  - Or by a message arrival into the X
break and par Combined Fragments

- Last two – break and par
- Break just like opt except the enclosing sequence is NOT evaluated if the guard is true
  - Similar to a break statement in a programming language
  - Another way to look at it is, it is an inverted OPT
    » That is, opt is do this if the guard is true, break is don’t do this if the guard is true
- Par allows multiple parallel sequences
- Looks just like alt, but no guards
**UML 1.4 Example**

1. **Customer**
   - AddItemToCart

2. **StoreFront**
   - IsAvailable

3. **Warehouse**
   - [Inventory = 0] Order

4. **Distributor**

5. **Shipper**
   - ShipItem

6. **Customer**
   - CheckOut
   - ShipTo
UML 2.0 Example

sd Example

StoreFront  Cart  Inventory

loop
/items in cart/

AddItem  ReserveItem

Checkout

ConfirmOrder  ProcessOrder  PlaceItemInOrder
Sequence Diagrams

- Are probably the most popular means of showing interaction between objects in UML
  - Communication Diagrams are still preferred by some
    » You will have to dig into these on your own if you want to learn about them
    » See Chapter 12 for a brief introduction

- It is often useful to take select ‘use cases’ and build a sequence diagram in the design that reflects the use case
  - Insures that the design does cover use case functionality
  - Doesn’t need to be perfect, but enough detail to verify that the use case can be performed as required.
  - We will cover use cases shortly
Sequence Diagrams

- Complex systems often contain quite a bit of messaging
  - For example, most web systems pass between multiple classes to process/render web pages

- Sequence diagrams can show this interaction fairly well
  - Diagrams show how the flow of messages in structured

- See Chapter 4 for more details