CS 307

Software Engineering

Lecture 1

Course outline

- Focus on practical software engineering techniques and tools for component-based object oriented system. The objective of this course is to learn how to design and construct OO systems.
- Main topic areas identified by IEEE Software Engineering Body of Knowledge:
  - **Software Engineering Process**: Extreme Programming
  - **Software Design**: Object oriented design with UML
  - **Software Construction**: OO coding practices, design patterns, refactoring, concurrency, aspect oriented programming
  - **Software Testing**: integrated testing with Junit
  - **Software Engineering Tools and Methods**: formal specification techniques, design by contract, and tools

Goals of this course

- Object-Oriented Software Engineering
  - How to design system of objects
  - How to exploit inheritance and subsumption to make systems generic and flexible
  - How and when to refactor systems to simplify their designs
- Software Quality
  - How to test and validate software
  - Good coding practices
  - How to document a design

Goals of this course

- Communication
  - How to work in groups
  - How to keep software as simple as possible
  - How to write software that communicates its design
- Skills, techniques and tools
  - How to use debuggers, version control systems, profilers, make, and others tools

Goals of this course

- Assignments (individual & project pairs)
- Project
  - A real software engineering project…

Workload

CS307 is a course with a fairly high workload, students should not expect a traditional bubbles-and-arrows SE:

- three assignments
- one project
- one midterm / final exam

Grading policy:

- Mid-nal 30%,
- Assignments 10%
- Project 60%

Academic Integrity

All work that you submit in this course must be your own.

Unauthorized group efforts are considered academic dishonesty.

You may discuss homework in a general way with others, but you may not consult any one else's written work. You are guilty of academic dishonesty if:

- You examine another's solution to a programming assignment (PA)
- You allow another student to examine your solution to a PA
- You fail to take reasonable care to prevent another student from examining your solution to a PA and that student does examine your solution.

Automatic tools will be used to compare your solution to that of every other current or past student. Don't con yourself into thinking you can hide your collaboration. The risk of getting caught is too high, and the standard penalty is way too high.

References

- **Lethbridge, Laganiere,** Object-oriented Software Engineering: Practical Software Development using UML and Java, MH 2001
- **Braude,** Software engineering: an object oriented perspective.
- **Gamma, Helm, Johnson, Vlissides,** Design Pattern — Elements of Reusable Object-Oriented Software, AW, 1995.
- **Beck,** extreme Programming explained — Embrace Change, AW 1999.
- **Fowler,** Refactoring: Improving the Design of Existing Code, AW 1999.
- **Pooley, Steven,** Using UML — Software Engineering with Objects and Components, AW, 1999.
- **Brooks,** The Mythical Man-Month, AW, 1982.

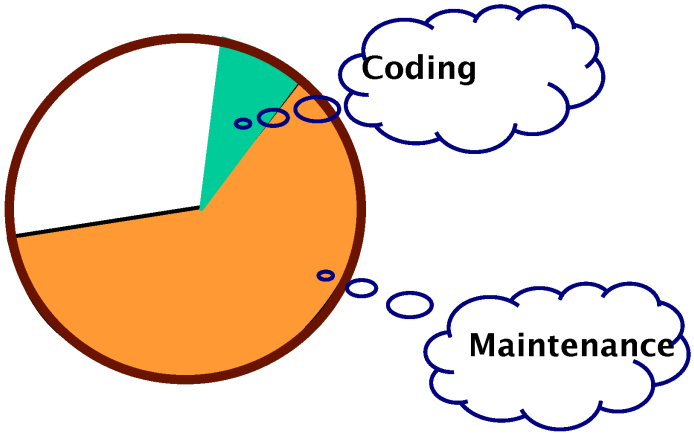
Course Overview

Outline

- Overview of main course topics
  - Software crisis
  - Processes
  - Software Quality
  - Object Oriented Programming
  - UML
  - Design Patterns
  - Refactoring
  - XP
- Presentation of programming assignment 1 (PA1)

The Software Crisis

- 1967 NATO study group coins the term Software Engineering
- 1968 NATO declare software crisis; quality of software is abysmal and deadlines and cost limits not met
- The cost of software development



1.1 The Nature of Software...

Software is intangible

- Hard to understand development effort

Software is easy to reproduce

- Cost is in its *development*
  - —in other engineering products, manufacturing is the costly stage

The industry is labor-intensive

- Hard to automate

The Nature of Software ...

Untrained people can hack something together

- Quality problems are hard to notice

Software is easy to modify

- People make changes without fully understanding it

Software does not 'wear out'

- It *deteriorates* by having its design changed:
  - —erroneously, or
  - —in ways that were not anticipated, thus making it complex

The Nature of Software

Conclusions

- Much software has poor design and is getting worse
- Demand for software is high and rising
- We are in a perpetual 'software crisis'
- We have to learn to 'engineer' software

1.2 What is Software Engineering?...

The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints

Other definitions:

- IEEE: (1) the application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).
- The Canadian Standards Association: The systematic activities involved in the design, implementation and testing of software to optimize its production and support.

What is Software Engineering?…

Solving customers' problems

- This is the *goal* of software engineering
- Sometimes the solution is to *buy, not build*
- Adding unnecessary features does not help solve the problem
- Software engineers must *communicate effectively* to identify and understand the problem

Basic Activities of Software Engineering

Object Oriented Software Engineering

Basic Activities of Software Engineering

- **defining the software development process to be used**

January 02, 2009 17
Adapted from Software Engineering: An Object-Oriented Perspective by Eric J. Braude (Wiley 2001), with permission.

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project

Object Oriented Software Engineering

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product

January 02, 2009 17
Adapted from Software Engineering: An Object-Oriented Perspective by Eric J. Braude (Wiley 2001), with permission.

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product

Object Oriented Software Engineering

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product
- implementing the product

January 02, 2009 17
Adapted from Software Engineering: An Object-Oriented Perspective by Eric J. Braude (Wiley 2001), with permission.

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product
- implementing the product
- testing the parts of the product

Object Oriented Software Engineering

Basic Activities of Software Engineering

- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product
- implementing the product
- testing the parts of the product
- integrating the parts and testing them as a whole

January 02, 2009 17
Adapted from Software Engineering: An Object-Oriented Perspective by Eric J. Braude (Wiley 2001), with permission.

Basic Activities of Software Engineering

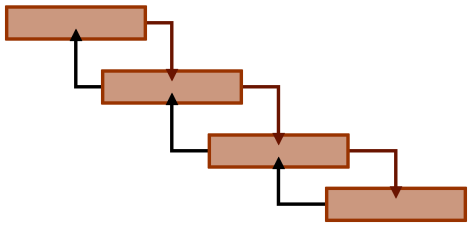
- defining the software development process to be used
- managing the development project
- describing the intended software product
- designing the product
- implementing the product
- testing the parts of the product
- integrating the parts and testing them as a whole
- maintaining the product

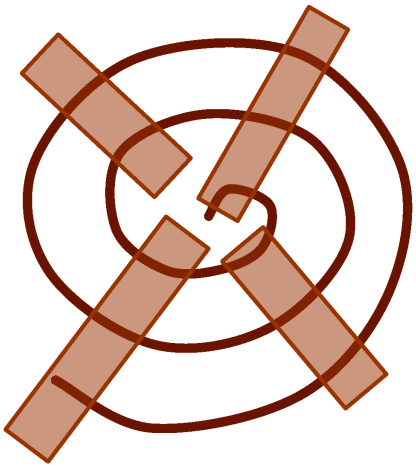
The Software Process

1. Requirements Phase
   - Requirements Phase Testing
2. Specification Phase
   - Specification Phase Testing
3. Design Phase
   - Design Phase Testing
4. Implementation Phase
   - Implementation Phase Testing
5. Integration Phase
   - Integration Phase Testing
6. Maintenance Phase
   - Maintenance Phase Testing
7. Retirement

Software Life-Cycle Models

- Of waterfalls and spirals…





What is software quality?

- Correctness is the ability of software products to perform their exact tasks, as defined by their specifications
- Robustness is the ability of software systems to react appropriately to abnormal conditions
- Extendibility is the ease of adapting software products to changes of specification
- Reusability is the ability of software elements to serve for the construction of many different applications
- Compatibility is the ease of combining software elements with others
- Efficiency is the ability of a software system to place as few demands as possible on hardware and software environments
- Ease of use is the ease with which people of various backgrounds and qualifications can learn to use software products

Bertrand Meyer, Object-Oriented Software Construction, PH

How to achieve software quality

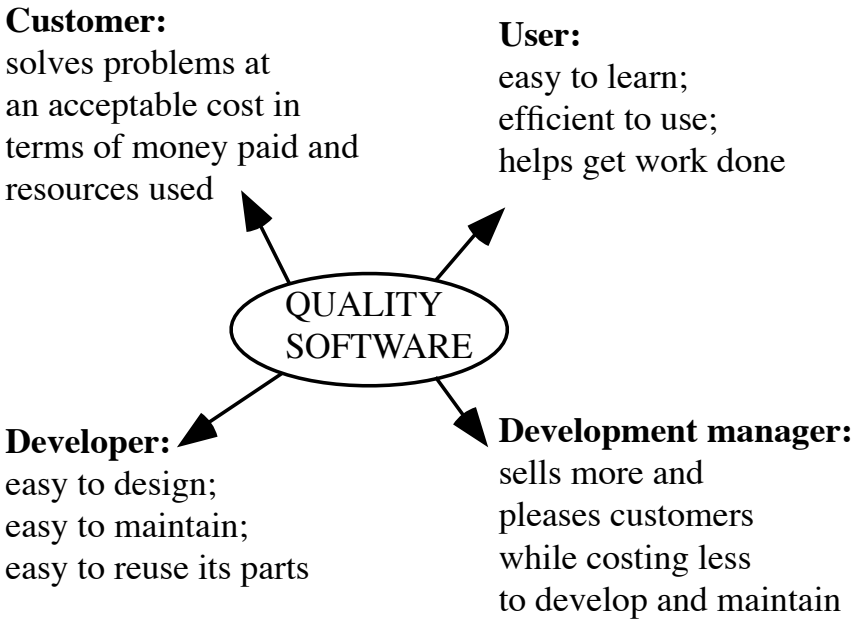
- by Design
  - pre and post conditions, class invariants
  - disciplined exceptions
  - design patterns, refactoring
- by Testing
  - unit tests, system tests
  - repeatable regression tests
  - do it, do it right, do it fast
    - Aim for simplicity and clarity, not performance
    - Fine-tune performance only when there is a demonstrated need!

Software Quality (ctd)

Other Methods to attain quality level:

- Inspection
  - team-oriented process for ensuring quality applied to all stages of the process
- Formal methods
  - Mathematical techniques to convince ourselves and peers that our programs do what they are meant to do applied selectively
- Project control techniques
  - predict costs and schedule
  - control artifacts (versions, scope etc.)

Software Quality and the Stakeholders



Software Quality: Conflicts and Objectives

The different qualities can conflict

- Increasing efficiency can reduce maintainability or reusability
- Increasing usability can reduce efficiency

Setting objectives for quality is a key engineering activity

- You then design to meet the objectives
- Avoids ‘over-engineering’ which wastes money

Optimizing is also sometimes necessary

- E.g. obtain the highest possible reliability using a fixed budget

Short Term Vs. Long Term Quality

Short term:

- Does the software *meet the customer's immediate needs*?
- Is it sufficiently efficient for the volume of data we have *today*?

Long term:

- Maintainability
- Customer's future needs
- Scalability: Can the software handle larger volumes of data?

Why object-oriented programming

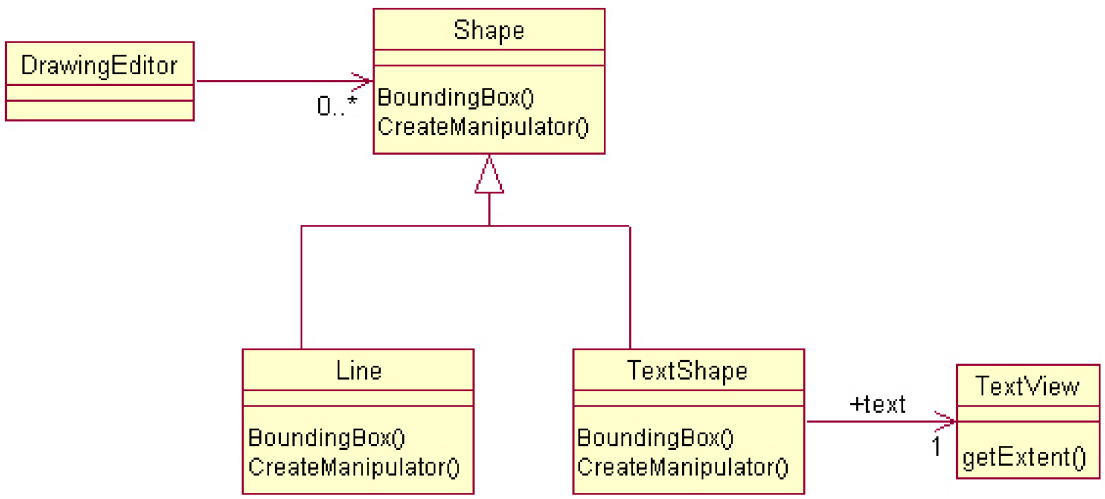
- Modeling
  - Complex systems can be naturally decomposed into software objects
- Data abstraction
  - Clients are protected from variations in implementation
- Polymorphism
  - Clients can uniformly manipulate plug-compatible objects
- Component reuse
  - Client/supplier contracts can be made explicit, simplifying reuse
- Evolution
  - Classes and inheritance limit the impact of changes

UML

- Unified Modeling Language a standardized set of techniques for describing object oriented designs.
  - Class models
  - Use case models
  - Interaction diagrams
  - State and activity diagrams
  - ...

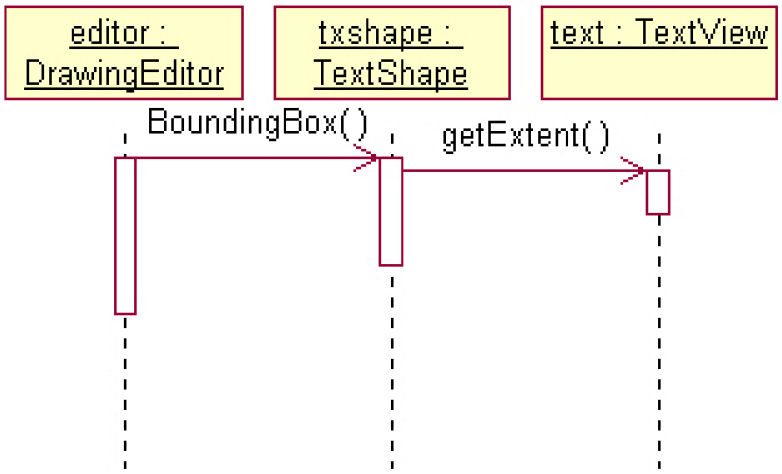
UML

- Drawing editor class diagram



UML

- Interaction diagram



Refactoring

- Refactor you design whenever the code starts to smell
  - Methods that are too long or hard to read
    - Decompose and delegate responsibilities
  - Duplicated code
    - Factor out the common parts
  - Violation of encapsulation
  - Too much communication between objects
  - Big case statements
    - Introduce subclass responsibilities
  - Hard to adapt to different context
    - Separate mechanism from policy

XP

eXtreme Programming — Kent Beck

- a light-weight methodology for small to medium sized teams developing software in the face of vague and rapidly changing requirements

- Tenets of XP:
  - If code reviews are good, then lets review all the time (pair programming)
  - If testing is good, everybody will test all the time (unit & functional testing)
  - If design is good, make it part of the daily routine (refactoring)
  - If simplicity is good, always choose the simplest design (KISS)
  - If integration testing is important, integrate and test several time a day
  - If short iterations are good, make the iteration really short — minutes and hours, not weeks and month

Why Java ?

Talking about Software Engineering without understanding programming is like performing surgery with your eyes closed … risky and painful

- C++ - - complexity
- Clean integration of feature
- A language academics do not have to be ashamed of
- Large standard library
- Simple object model
  - Almost everything is an object
  - No pointers
  - Garbage collection
  - Single inheritance
  - Multiple subtyping
  - Static and dynamic type checking

Programming Assignment 1

PA1

- The goal of PA1 is to implement one or more bit vector class(es).
- A bit vector is a map from integer values to booleans.

```
simple ops:
   bs.set(1);          // bs.test(1) == true
   bs.set(2);          // bs.test(2) == true
   if (bs.test(2)) bs.negate(1);
   boolean v = bs.set(1);  // v == true
binary ops:
   BitVector bv = bs.copy(); // bs.equals(bv)
   bs.set(12);               // bs.test(12) == true
       bv.or(bs);                    // bv.test(12) == true
   bv.unset(12);             // bv.test(12) == true
   bs.and(bv);               // bs.test(12) == false
```

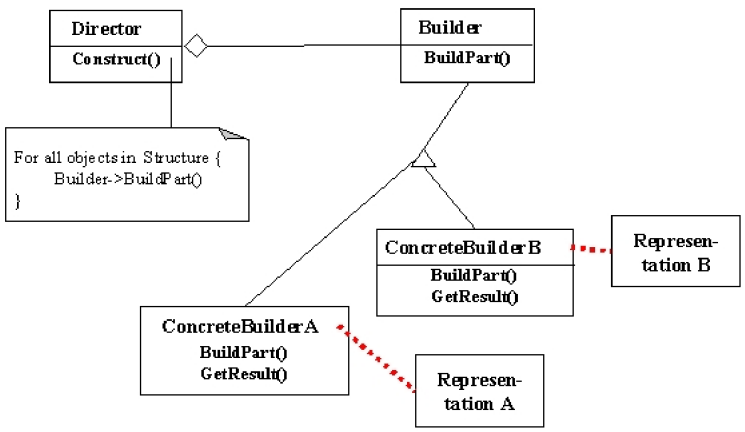
PA1

```
iteration:
   Iterator iter = bs.iterator;
   while (iter.hasNext())
        boolean val = iter.getNext();
        …

   for (iter.skipToSet(); iter.hasNext(); iter.skiptToSet()) {
        int pos = iter.position();
        iter.getNext();
        …
   }
```

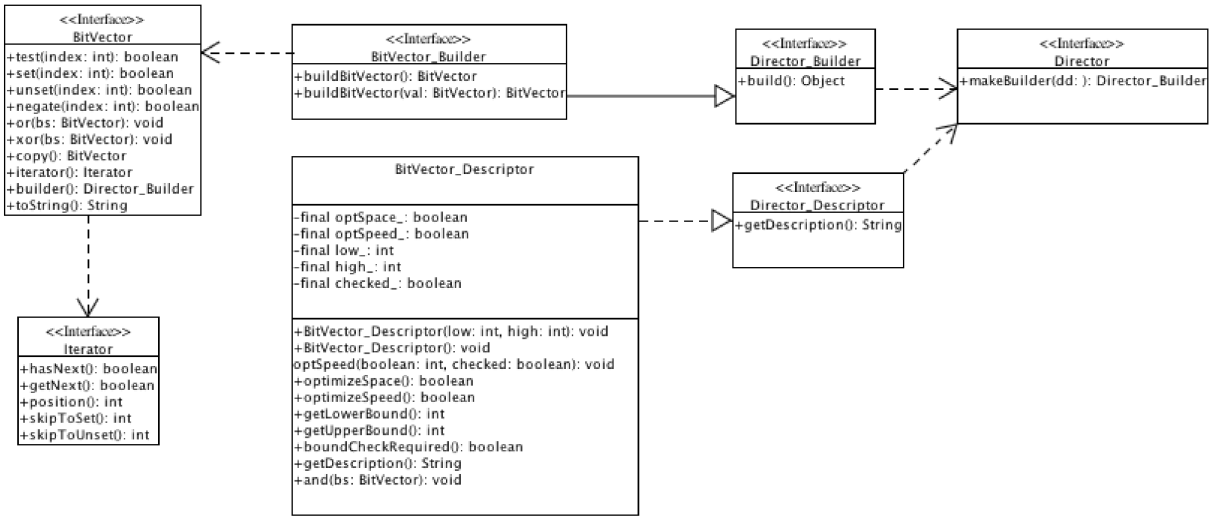

PA1

- **The Builder Pattern** [GoF].
  The Builder pattern allows a client object to construct a complex object by specifying only its type and content. The client is shielded from the details of the object's construction.



PA1

- Putting it all together



Object Oriented Software Engineering

PA1

- You will have to implement:
  - BitVector_xy implement BitVector
  - Iterator_xy implements BitVector.Iterator
  - Builder_xy implements BitVector.Builder
  - Director_xy implements Director

PA1

```
// Create one instance of a director.
static final Director director = new Director_jv();

// Create a descriptor for small and compact
// bit vectors with bound checks enabled.
static final BitVector.Descriptor small
   = new BitVector.Descriptor(0, 31, false, true, true);

// Create a builder
static final BitVector.Builder builder
         = (BitVector.Builder) director.make(small);

 BitVector bs = builder.build();
```


PA1

Use cases

- **Small-sets** This scenario is concerned with small bits vectors for which the test, or and and methods are speed critical. Furthermore, the size of data structures has to minimized. In the small-set scenario, expect large numbers of vectors (thousands) with ranges from 0 to 127.
- **Big-sets** This scenario has a number of large, sparse, bit vectors being used mostly for their set and test methods. In this scenario a space efficient representation of large vectors and speed of tests are critical. All other operations are used rather infrequently.