CISC 323
Introduction to Software Engineering

Winter 2004

Administrativa

Section A:
instructor: Margaret Lamb
malamb@cs.queensu.ca
DUP 217
slot 3: Mon 10:30
Wed 9:30
Fri 8:30

Section B:
instructor: Juergen Dingel
dingel@cs.queensu.ca
DUP 215
slot 1: Mon 8:30
Tues 10:30
Thurs 9:30

Course web page:
http://www.cs.queensu.ca/home/cisc323
(is up, minor changes may still be made)

WebCT area: forum and grades

Assignments, Exams and Marking Scheme

5 assignments: 30%
midterm: 20%
final exam: 50%

Midterm: two-hour evening exam
during Week 8, date to be announced soon

Final Exam: date and location to be announced
Please don't make plans to leave Kingston before
end of the exam period!

All are closed book with 1 8.5”x11” data sheet

Tutorials

- Monday 1:30 - 2:20, WLH205
  or
  Wednesday 12:30 - 1:20, WLH210
- You may attend either tutorial, regardless of your
  lecture section.
- Purpose of tutorials:
  ▶ review/reinforce material from lecture & readings
  ▶ ask questions
  ▶ discuss assignments
- NO TUTORIALS DURING WEEK 1!
Assignments

- There will be 5 assignments during the course.
- Posted on web site.
- You may work in pairs.
- No groups larger than 2.
- Academic Dishonesty will not be tolerated!
- No late assignments accepted.

Text Material

- Required texts:
  2. custom courseware Both available in bookstore

- If your Java is rusty:
  - texts from old Java courses or any other good Java book
  - Another option:

Programming Resources

- Project implemented in Java.
- You may use any Java tool which is based on a Sun Java JDK version 1.2 or later:
  - BlueJ
  - Ready To Program
  - JBuilder 3+ (some labs have JBuilder 7)
  - Forte
  - straight Sun jdk/sdk
- do not use :
  - Visual J++ (based on Java variant)
  - JBuilder 2 in labs (old version)

What is Software Engineering?

Any thoughts?
Software Engineering: A Definition

**engineering:**
The application of scientific and mathematical principles to practical ends such as the design, manufacture, and operation of efficient and economical structures, machines, processes, and systems

American Heritage Dictionary

**software engineering:***
The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering to software

IEEE Standard 610.12

What Is Software?

• The programs, routines, and symbolic languages that control the functioning of the hardware and direct its operation.

American Heritage Dictionary

• Crucial to functioning of modern society
  ▶ transportation
  ▶ business and finance
  ▶ health care
  ▶ communication
  ▶ entertainment
  ▶ education
  ▶ …

What Is Software? (Cont’d)

• Crucial for western economies
  ▶ Value of US prepackaged sw
    • in 1996: US$109 billion
    • in 2002 (est): US$222 billion (2.2% of US$9.9 trillion of US GDP)
  ▶ Expected growth of global sw market: 6.1%
  ▶ Average IT spending of US companies as % of gross revenue in 2000: 7.5%

<table>
<thead>
<tr>
<th>Company</th>
<th>#employees</th>
<th>Revenue (US$ in 10^9)</th>
<th>Net income (US$ in 10^9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft</td>
<td>47,600</td>
<td>25.3</td>
<td>7.3</td>
</tr>
<tr>
<td>IBM</td>
<td>N/A</td>
<td>12.6</td>
<td>10.3</td>
</tr>
<tr>
<td>Oracle</td>
<td>42,927</td>
<td>10.8</td>
<td>2.5</td>
</tr>
</tbody>
</table>


What Is Software? (Cont’d)

• Not created equal. At least 4 different types:
  ▶ shrinkwrap
    • e.g., Win xyz
    • high volume, little service, needs to be very usable, lots of maintenance
    • multi-platform, multi-OS, multi-human-language, multi-version
    • developers likely to be geographically dispersed
  ▶ customized/internal
    • e.g., SAP business software
    • low-volume, high service, lots of maintenance
  ▶ embedded
    • e.g., device drivers, control software
    • medium volume, strong hardware restrictions, requires high performance, high reliability (safety-critical), little maintenance
  ▶ games
    • lots of fancy graphics, high performance
What Is Software? (Cont’d)

- One of most complex man-made artifacts:
  - Windows 95
    - > 11 million lines of code
    - more than 200 programmers & testers
  - Windows NT: > 16 million lines of code
  - Windows XP: 45 million lines of code
  - Pacemakers: > 100,000 lines of code
  - "I believe the [spreadsheet product] I’m working on now is far more complex than a 747 (jumbo jet airliner)"
    -- Chris Peters (Microsoft, 1992)

- Rapidly evolving:
  - Changing languages:
    - Paradigms: imperative, functional, logical, object-oriented
  - Changing technologies:
    - computer architectures, means of communication (networks, internet, web)
  - Changing demands:
    - GUIs, Web applications, eCommerce, security

What Is Software? (Cont’d)

- Highly constrained: Must distinguish at least 3 different “stakeholders”:
  - Customers:
    - Which expectations do customers have of software they buy or use?
  - Developers:
    - Which expectations do developers have of software they
      - write from scratch,
      - test,
      - maintain?
  - Managers:
    - Which expectations do managers have of software whose
devvelopment they oversee?

-- Bill Gates, 1992

Lientz & Swanson, 1980
Consequences for Process of Software Development

- Given all these properties of and constraints on software, the development process of (large, commercial) software must be
  - divided into several phases (e.g., requirements analysis, design, implementation, testing)
  - able to measure and control the quality of the produced software
  - predictable
  - efficient (through, e.g., lots of reuse, use of modern languages, doing things in parallel when possible, minimizing number of developers and testers, facilitating communication between people)
  - able to adapt to changing requirements

Software Process

- A software (development) process is a method for developing software that organizes the effort in a number of separate steps such that large software can be developed by many people in an organized, manageable and trackable way

- Numerous software processes (also called models) exist
  - Waterfall process
  - Prototyping process
  - Evolutionary development process
  - Sync and Save process (Microsoft)
  - Object-oriented development process (e.g., Uniform Approach (UA), …)
  - …

Consequences for Software Itself

- Given all these properties of and constraints on software, the software must
  - be modular
    - divide the code into appropriate number of appropriate units (classes, modules, components) with
    - small and well-defined interfaces and dependencies
  - use information hiding
    - units are only accessible through their interfaces
    - the internals of a unit are invisible
  - be implemented with the user’s requirements in mind
  - be checked for satisfaction of these requirements
  - use existing code as much as possible
  - minimize typos, bugs, security vulnerabilities
  - be well documented and use good coding style consistently
  - use efficient data structures and algorithms

Consequences for Software Itself (Cont’d)

- Object-oriented languages like Java support
  - modularity through
  - information hiding through
  - reuse through
  - correctness through
  - documentation through
Consequences for Software Itself (Cont’d)

- Object-oriented languages like Java support
  - modularity through
    - packages, classes, interfaces
  - information hiding through
    - access modifiers: public, protected, private
  - reuse through
    - inheritance
    - rich API
  - correctness through
    - strong typing (programs are checked for type correctness)
    - automatic memory management
    - run-time assertions (JDK1.4)
  - documentation through
    - documentation facilities like Javadoc
    - run-time assertions

Example 1

Browser War between Microsoft and Netscape:
- From 1995 to 1997 NS concentrated on features at the expense of good design
- MS hurried to get IE going, but took time to restructure IE3.0 (NT built from scratch, shared components in Office)
- By 1997, NS C4.0 had 130 developers, 3M loc
- Two months not enough to rearchitect NS C4.0
- NS decides to start from scratch with C6.0
- C6.0 never finished, developers reassigned to C4.0
- C5.0 open source, but nobody wants to work on it
- MS wins Browser War, AOL buys NS
- NS C4.0 still contains 1.2M loc

From: Competing on Internet Time. M.A. Cusumano, D.B. Yoffie

Example 2

Mars Climate Orbiter (2000):
- Some programs worked in English (imperial) units, some metric units
- Conversion from English to metric forgotten
- Instead of 65 miles probe attempted to orbit 65 km (40 miles) above Mars
- $130M lost

Costs of Failed Software Development and Failed Software

- There are numerous examples of
  - failed software development projects (e.g., budget overrun, cancelled).
  - According to Standish Group in US in 2000:
    - ¼ of all software projects cancelled at total cost of US$67 billion
    - total cost of budget overruns: US$21 billion
  - buggy software making people lose time, money, their lives
- Annual cost of errors in software in US in 2001
  - US National Institute of Standards and Technology: US$60 billion
  - Sustainable Computing Consortium: > US$200 billion
- Watts Humphrey (CMU SEI):
  - Professional programmers make 100 to 150 errors per 100 lines of code
  - Win NT4 (16 million lines of code) written with 2 million errors
- The Risk Digest collects risks to the public in computers
  - moderated by Peter Neumann at SRI
  - http://catless.ncl.ac.uk/Risks/
Example 3

Ariane 5 Explosion:
- June 4, 1996
- Controlled destruction 40 seconds after lift-off
- due to rounding error, code re-use
- 15 years of development
- $7B investment
- $500M uninsured cargo lost (4 satellites)
- Failure to meet correctness requirements

Example 4

Therac 25:
- The most infamous medical software failure
- Computer based electron-accelerator radiation therapy system
- Accidents from 1985 to 1987 involved massive radiation exposure
- Results: 3 deaths, several injuries
- tests after accidents showed no problems
- three subtle interacting flaws in system
- Failure to meet correctness requirements
- Another example of safety critical system

Example 5

Bank of America MasterNet:
- Abandoned existing trust accounting and reporting system for a new computer-based system called MasterNet
- Initial budget $23M for 5-year development
- Additional $60M spent trying to get MasterNet working after 5 years
- Failure to meet correctness, performance requirements
- Eventually abandoned MasterNet
- Lost $Billions in customer business

Example 6

Blue Cross and Blue Shield (Wisconsin):
- 1983 hired EDS to build $200M system
- Delivered on time but did not work correctly:
  - System issued $60M in overpayments and duplicate cheques
- Failure to meet correctness requirements
- By 1987 company had lost 35,000 policyholders
Example 7 & 8

First Shuttle Launch (1981):
- launch on hold 2 days
- timing error kept main computer & backups from synchronizing

Challenger Explosion (1986):
- main problems: booster rocket design, cold weather
- better software might have saved the astronauts
- design decision to save money: remove sensors on booster rockets
- sensors & software might have detected booster failure earlier
- some speculate detection would have permitted separation before explosion


Example 9

Bruce Nuclear Generation Plant:
- Fuel Handling Software Failure (1990)
- Brakes on a fueling machine were released while clamped to a fuel channel fitting
- Machine dropped 40cm damaging the fitting causing a leak of radioactive heavy water (1400 kg/h)
- reactor down for 6 weeks
- Software error had been introduced 4 years earlier
- A unique combination of events lead to the problem


Topics in This Course

1. Introduction (2 lectures)
2. OO review and UML (4)
3. Debugging (1)
4. GUIs in Java (6)
5. Software processes (2)
   - waterfall, incremental, prototyping, evolutionary, sync & save, object-oriented (Uniform Approach)
6. Object-Oriented Analysis (3)
   - ways to determine and capture what system is supposed to do
7. Object-Oriented Design (3)
   - determine how system is supposed to be implemented
8. Design Patterns (3)
   - high-level way to solving certain coding problems
9. Software Architecture (3)
   - high-level ways of structuring code
10. Software Quality (4)
    - ways to ensure that system behaves as desired
11. Open Source (1)
Top 10 List:

Why This Course Is Relevant To You

Top 10

10. Watch the “growing pains” of an “emerging discipline”
   ▶ Construction and analysis of models crucial for the theory and practise of all engineering disciplines
   ▶ In SWE this is not true --- yet!

9. Job availability and personal interests change
   ▶ Do you really know what you’ll be doing after graduation?
   ▶ Knowing more has never hurt anybody

8. Some SWE concepts may be useful/applicable in other contexts
   ▶ It’s all about controlling complexity

7 to 1. SW is everywhere
   ▶ may have to interact with SW people
   ▶ may have to manage SW projects
   ▶ may have to write code

“Software is pervasive. EEs that pursue a technical career will at some point likely have to develop, manage, or specify software.”

-- Geoffrey Chan, ECE UG Chair
July 11, 2003

Extended Example: Obtaining Marks in Course

► A common problem in courses is that the grade is computed from many components
   ▶ assignments, labs, quizzes, final exam

► Students typically see only the final result
   ▶ Therefore hard for students to verify their grade has been computed, recorded accurately
Solution: Program to Obtain Marks in Course

- Allow students to run a program to view their grades for each of their courses
  - Student must be logged in to CASLab
  - CASLab user id used to identify student
  - Student selects course for which he/she wishes to see grades
  - Grades displayed on students screen

Architecture of Marks Program

- Client-server architecture
- **Server** contains
  - Marks data for all students, courses
  - Service allowing marks data to be obtained
- **Client** contains
  - Program allowing student to query and display his/her marks
- **Communication**
  - Network-based

Components of Marks Program
Components of Marks Program

- Marks Server: Serves requests for marks, using CASLab user id and course id as key.
- Marks Repository: Contains each student’s set of marks for each course they are taking.
- Marks Client: Makes requests for marks, interacts with user interface.
- User Interface: Allowing students to request and view marks.
- Network link

Marks Repository

- Solution 1: Use a text file
- Solution 2: Use a database

Solution 1: Marks Repository as a Text File

- File contains lines of form
  
  CASLabId CourseNumber Mark1 Mark2 ... Markn

- For example:
  
  cs1234 CISC323 42 55 93 ...

- When students look up grades, they get back the list of marks

Marks Repository as a Text File: Consequences

How usable would this solution be?

- Professors must all share this large, complex text file to enter grades
- Format is cryptic
- Easy to make typos putting data into this format (therefore, marks data may not accurately reflect master list)
Marks Repository as a Text File: Consequences (Cont’d)

How **scalable** would this solution be?

- As number of users becomes large, accessing text file may become too slow
  - Files only support **sequential access** (such as audio tapes), not random access (such as CDs)
  - Each lookup requires sequential scan of file

How **tolerant to faults** would this solution be?

- If text file deleted or marks server machine goes down, system fails
- However, text file easy to copy and back up

Solution 2: Marks Repository as a Database

- Database contains tables representing marks information
- Marks server connects to database via standard protocol
  - E.g., ODBC (Microsoft), JDBC (Sun), ...
- Resolves problems of **scalability**
  - Database access far faster than sequential file access when large number of entries
- Does not help with **fault-tolerance**
  - Failure of database still renders system inoperative
  - Database should have built-in support for backups. However, backups may still be difficult/costly

Marks Repository as a Database: Consequences

**Usability**

- Entry into a database requires professor to use a special Marks Entry Client
  - E.g., forms-based interface
- Easier to use and less error-prone than text editor
- More work to develop and maintain
Some System Attributes

- Usability
- Fault tolerance
- Scalability
- Others that we have not considered:
  - **Security**
    - Marks data sent over network
    - If using standard Ethernet, could be “snooped” by third party
  - **Portability**
    - What platforms does Marks Client User Interface run on?
    - What are required?
  - **Maintainability, reusability, cost, ...**

Lessons from this Example

- There’re typically several different ways of implementing a system
  - Design decisions must be made
  - Each decision may have different impact on quality attributes of resulting system
  - Need to which attributes we need to care about
  - Need to do **requirements analysis** before we can design (if you’re using an object-oriented software development process, the requirements analysis is called **object-oriented analysis**)

Lessons from this Example (Cont’d)

- Design is typically **top-down**
  - Start by breaking system up into a set of communicating components
    - How do we split the system into components?
    - How do we deploy the components on a set of computers?
    - How do the components communicate with each other?
    - How does the architecture influence the quality attributes?
  - Then, think about design of components in terms of processes, packages, tools, classes, etc
  - **Ideally:** Keep on breaking each problem down until you have an existing solution for it, or it is so simple that you can see how to implement it

Engineering as Art and Science

"The conception of a design for a new structure can involve as much a leap of the imagination and as much a synthesis of experience and knowledge as any artist is required to bring to his canvas or paper. And once that design is articulated by the engineer as artist, it must be analyzed by the engineer as a scientist in as rigorous an application of the scientific method as any scientist must make."

--- Henry Petroski

Author of “To Engineer is Human: The Role of Failure in Successful Design”