Advanced Software Engineering

Dr. Cheng

Overview of Software Engineering and Development Processes

CSE870

FYI

- Professor in CSE
- Here at MSU for > 20 years
  - Software Engineering and Network Systems (SENS) Lab
  - Digital Evolution (DEVOLab)
  - BEACON: NSF Science and Technology Center (“Evolution in Action”)
- Research and Instruction areas:
  - High-assurance systems
  - Model-driven engineering
  - Autonomic (self-adaptive) systems
  - Recently, also working in following areas:
    - Search-based SE (e.g., evolutionary computing, MOO, ML, etc.)
    - Cybersecurity for Automotive and onboard Systems
  - Work extensively with industrial collaborators (e.g., Ford, GM, Continental Automotive, Motorola, BAE Systems, Siemens, ZF, Aerospace Corp.)
What is Software Engineering?

- Systematic approach for developing software
- Methods and techniques to develop and maintain quality software to solve problems. (Software Engineering: Methods and Management, Pfleeger, 1990)
- Study of the *principles* and *methodologies* for developing and maintaining software systems. (``Perspectives on Software Engineering,'' Zelkowitz, 1978)

What is Software Engineering?

- *Practical* application of scientific knowledge in the design and construction of computer programs and the associated *documentation* required to develop, operate, and maintain them. (``Software Engineering,'' Boehm, 1976)
- Deals with establishment of *sound engineering principles and methods* in order to *economically* obtain software that is *reliable* and works on real machines. (``Software Engineering,'' Bauer, 1972)
Questions addressed by Software Engineering

• How do we ensure the quality of the software that we produce?
• How do we meet growing demand and still maintain budget control?
• How do we avoid disastrous time delays?

Why apply Software Engineering to Systems?

• Provide an understandable process for system development.
• Develop systems and software that are maintainable and easily changed.
• Develop robust software and system.
• Allow the process of creating computing-based systems to be repeatable and manageable.
Objectives of Course

- Provide exposure to leading-edge topics
  - Emphasize model-driven engineering
  - Emphasize requirements and design
  - Emphasize assurance of computing-based systems
- Provide hands-on experience to reinforce concepts
  - Homework assignments
  - Modeling and specification assignments
- Synthesize several topics into mini-projects
  - Programming/design Project with written component
  - Prepare presentation materials for lay audience.

- Overarching application theme: assurance for onboard automotive systems

Tentative Topics

- Requirements Engineering
- Model-driven engineering (UML)
- Architectural Styles
- Design Patterns
- Security
- Testing
- (Search-based Software Engineering)
- (Interplay between SE and ML)
Administrative Work

- Background Survey
- Initial Assessment
- Tentative Evaluation Mechanisms:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exams (2)</td>
<td>40 %</td>
</tr>
<tr>
<td>In-class participation;</td>
<td>20 %</td>
</tr>
<tr>
<td>Homework/Design Exercises</td>
<td></td>
</tr>
<tr>
<td>Mini-Project(s)</td>
<td>40 %</td>
</tr>
</tbody>
</table>

PAUSE
Historical Perspective

- **1940s**: computers invented
- **1950s**: assembly language, Fortran
- **1960s**: COBOL, ALGOL, PL/1, operating systems
  
  *1969: First conference on Software Eng*
- **1970s**: multi-user systems, databases, structured programming

Historical Perspective (cont.)

- **1980s**: networking, personal computing, embedded systems, parallel architectures
- **1990s**: information superhighway, distributed systems, OO in widespread use.
- **2000s**: virtual reality, voice recognition, video conferencing, global computing, pervasive computing...
- **2010s**: EMRs, autonomous vehicles, new security awareness, ...
Why is software so expensive?

- Hardware has made great advances
- But, software has made great advances ...
- We do the least understood tasks in software.
  - When task is simple & understood, encode it in hardware
  - Why?
- Demand more and more of software
  - Consider your cell phone
Size of programs continues to grow

- **Trivial**: 1 month, 1 programmer, 500 LOC,
  - Intro programming assignments
- **Very small**: 4 months, 1 programmer, 2000 LOC
  - Course project
- **Small**: 2 years, 3 programmers, 50K LOC
  - Nuclear power plant, pace maker
- **Medium**: 3 years, 10s of programmers, 100K LOC
  - Optimizing compiler

Size of programs continues to grow

- **Large**: 5 years, 100s of programmers, 1M LOC
  - MS Word, Excel
- **Very large**: 10 years, 1000s of programmers, 10M LOC
  - Air traffic control,
  - Telecommunications, space shuttle
- **Very, Very Large**: 15+ years, 1000s programmers, 35M LOC
  - W2K
- **Ultra-Large Scale**: ? years, ? developers distributed,
  - 1000s of sensors, decision units,
  - heterogeneous platforms, decentralized control
  - Intelligent transportation systems; healthcare systems
New Scale
Ultra-Large Scale SW-Intensive Systems (IOT)

Healthcare Infrastructure

New Scale
Intelligent Transportation and Vehicle Systems
The ULS Ecosystem

- **Key elements:**
  - Computing devices
  - Business and organizational policies
  - Environment (including people)

- **Forces:**
  - Competition for resources
  - Unexpected environmental changes
  - Decentralized control
  - Demand for assurance

---

Context: “Sufficient” System Health

- **High-level Objective:**
  - How to design a safe adaptive system with incomplete information and evolving environmental conditions

- **Execution environment**
  - How to model environment
  - How to effectively monitor changing conditions
  - Adaptive monitoring

- **Decision-making for dynamic adaptation**
  - Decentralized control
  - Assurance guarantees (functional and non-functional constraints)

- **Adaptation mechanisms:**
  - Application level
  - Middleware level
What’s the problem?

• Software cannot be built fast enough to keep up with
  – H/W advances
  – Rising expectations
  – Feature explosion
• Increasing need for high reliability software

What’s the problem?

• Software is difficult to maintain
  “aging software”
• Difficult to estimate software costs and schedules
• Too many projects fail
  – Arianne Missile
  – Denver Airport Baggage System
  – Therac
Why is software engineering needed?

- To predict time, effort, and cost
- To improve software quality
- To improve maintainability
- To meet increasing demands
- To lower software costs
- To successfully build large, complex software systems
- To facilitate group effort in developing software

Software Engineering Phases

- Definition: What?
- Development: How?
- Maintenance: Managing change
- Umbrella Activities: Throughout lifecycle
Definition

- Requirements definition and analysis
  - Developer must understand
    - Application domain
    - Required functionality
    - Required performance
    - User interface

Definition (cont.)

- Project planning
  - Allocate resources
  - Estimate costs
  - Define work tasks
  - Define schedule

- System analysis
  - Allocate system resources to
    - Hardware
    - Software
    - Users
Development

- Software design
  - User interface design
  - High-level design
    - Define modular components
    - Define major data structures
  - Detailed design
    - Define algorithms and procedural detail

Development (cont.)

- Coding
  - Develop code for each module
  - Unit testing
- Integration
  - Combine modules
  - System testing
Maintenance

- Correction - Fix software defects
- Adaptation - Accommodate changes
  - New hardware
  - New company policies
- Enhancement - Add functionality
- Prevention - make more maintainable

Umbrella Activities

- Reviews - assure quality
- Documentation - improve maintainability
- Version control - track changes
- Configuration management - integrity of collection of components
Development Process

- Step-by-step procedure to develop software
- Typically involves the major phases:
  - analysis
  - design
  - coding
  - testing

Waterfall Process Model

- Requirements
- Design
- Coding
- Testing
- Maintenance
2.2 Software Process Models

V Model (continued)

- REQUIREMENTS ANALYSIS
  - Validate requirements

- SYSTEM DESIGN
  - Verify design

- PROGRAM DESIGN
  - UNIT & INTEGRATION TESTING
  - SYSTEM TESTING

- OPERATION & MAINTENANCE

Prototyping Process Model

- Requirements
- Quick Design
- Prototype
- Evaluate
- Design
When to use prototyping?

- Help the customer pin down the requirements
  - Concrete model to “test out”
  - Often done via the user interface
- Explore alternative solutions to a troublesome component
  - e.g., determine if an approach gives acceptable performance
- Improve morale
  - Partially running system provides visibility into a project

2.2 Software Process Models
Spiral Model (continued)

Figure 2.10 the spiral model.
2.2 Software Process Models

Agile Methods

• Emphasis on flexibility in producing software quickly and capably

• Agile manifesto
  – Value individuals and interactions over process and tools
  – Prefer to invest time in producing working software rather than in producing comprehensive documentation
  – Focus on customer collaboration rather than contract negotiation
  – Concentrate on responding to change rather than on creating a plan and then following it

Agile Methods: Examples of Agile Process

• Extreme programming (XP)
• Crystal: a collection of approaches based on the notion that every project needs a unique set of policies and conventions
• Scrum: 30-day iterations; multiple self-organizing teams; daily “scrum” coordination
• Adaptive software development (ASD)
2.2 Software Process Models

Agile Methods: Extreme Programming

- Emphasis on four characteristics of agility
  - **Communication**: continual interchange between customers and developers
  - **Simplicity**: select the simplest design or implementation
  - **Courage**: commitment to delivering functionality early and often
  - **Feedback**: loops built into the various activities during the development process

---

2.2 Software Process Models

Agile Methods: Twelve Facets of XP

- The planning game (customer defines value)
- Small release
- Metaphor (common vision, common names)
- Simple design
- Writing tests first
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration (small increments)
- Sustainable pace (40 hours/week)
- On-site customer
- Coding standard
2.2 Software Process Models
Sidebar 2.2 When Extreme is Too Extreme?

- Extreme programming's practices are interdependent
  - A vulnerability if one of them is modified
- Requirements expressed as a set of test cases must be passed by the software
  - System passes the tests but is not what the customer is paying for
- Refactoring issue
  - Difficult to rework a system without degrading its architecture

Process Models

- Idealized views of the process
- Different models are often used for different subprocesses
  - may use spiral model for overall development
    - prototyping for a particularly complex component
    - waterfall model for other components
**Capability Maturity Model**

- **Level 1: Initial**
  - ad hoc
  - success depends on people

- **Level 2: Repeatable**
  - track cost, schedule, functionality

- **Level 3: Defined**
  - use standardized processes

- **Level 4: Managed**
  - collect detailed metrics

- **Level 5: Optimizing**
  - continuous process improvement
  - “built-in” process improvement

---

**Why is software development so difficult?**

- **Communication**
  - Between customer and developer
    - Poor problem definition is largest cause of failed software projects
  - Within development team
    - More people = more communication
    - New programmers need training

- **Project characteristics**
  - Novelty
  - Changing requirements
    - 5 x cost during development
    - up to 100 x cost during maintenance
  - Hardware/software configuration
  - Security requirements
  - Real time requirements
  - Reliability requirements

---

Software Engineering Institute: [http://www.sei.cmu.edu/cmm/](http://www.sei.cmu.edu/cmm/)
Why is software development difficult? (cont.)

- Personnel characteristics
  - Ability
  - Prior experience
  - Communication skills
  - Team cooperation
  - Training

- Facilities and resources
  - Identification
  - Acquisition

- Management issues
  - Realistic goals
  - Cost estimation
  - Scheduling
  - Resource allocation
  - Quality assurance
  - Version control
  - Contracts

Summary

- Software lifecycle consists of
  - Definition (what)
  - Development (how)
  - Maintenance (change)

- Different process models concentrate on different aspects
  - Waterfall model: maintainability
  - Prototype model: clarifying requirements
  - Spiral model: identifying risk

- Maintenance costs much more than development
## Bottom Line

- U.S. software is a major part of our societal infrastructure
  - Costs upwards of $200 billion/year
- Need to
  - Improve software quality
  - Reduce software costs/risks