Lecture 2: Software Engineering (a review)

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2003

Credit where Credit is Due

- Some material presented in this lecture is taken from chapter 1 of Maciaszek's "Requirements Analysis and System Design".
- © Addison Wesley, 2000

Goals for this Lecture

- Introduce definition of software engineering
- Begin review of topics covered in Maciaszek, Chapter 1

What is "Software Engineering"

- Software
  - Computer programs and their related artifacts
- Engineering
  - The application of scientific principles in the context of practical constraints

A Definition (Daniel M. Berry)

- Software engineering is that form of engineering that applies:
  - a systematic, disciplined, quantifiable approach,
  - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines,
- to creating, developing, operating, and maintaining cost-effective, reliably correct, high-quality solutions to software problems.

Engineers Build Machines

- Software is intangible
  - Descriptions of a desired machine, written according to specific languages and notations
- Computer is tangible
  - General-purpose description executor
  - Behaves as if it were the desired machine
- Software engineers "build" descriptions
  - Organizing, structuring, and making complex assemblages of descriptions
  - Raw materials: languages and notations

Basic Software Engineering Activities

- Create
  - Modeling
- Record
  - Specification
- Analyze
  - Verification & Validation
- Configure
  - Selection, Translation, & Deployment

Topics from Chapter 1

- The Nature of Software Development
- System Planning (skipped)
- Software Lifecycle Phases
- Software Development Approaches

The nature of software (Brooks)

- Maciaszek starts by discussing the problems that arise because software is an intangible medium
  - He cites the classic source on this topic, Fred Brooks' "No Silver Bullet" essay that appeared in 1986

No Silver Bullet

- "There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity."

  -- Fred Brooks, 1986

- i.e. There is no magical cure for the "software crisis"

Why? Essence and Accidents

- Brooks divides the problems facing software engineering into two categories

  essence

  - difficulties inherent in the nature of software

  accidents

  - difficulties related to the production of software

- Brooks argues that most techniques attack the accidents of software engineering

An Order of Magnitude

- In order to improve the development process by a factor of 10
  - the accidents of software engineering would have to account for 9/10ths of the overall effort
  - tools would have to reduce accidents to zero
- Brooks
  - doesn't believe the former is true and
  - the latter is highly unlikely, even if it was true

The nature of software (Brooks)

- The software essence
  - Complexity
  - Conformity
  - Changeability
  - Invisibility
- The software accidents
  - Stakeholders, Process, Tools

Software Development Invariant

- Software production is an art
  - Software is developed, not manufactured

… certainly SE advances have increased our ability to construct software but the success of a software system cannot be guaranteed

Software Invariant continued

- Advances such as…
  - object technology, commercial-off-the-shelf (COTS) software and Enterprise Resource Planning (ERP) systems
- have helped to transform the development process from one of "building from scratch" to "customizing component"
- … but what about core business processes?
  - Typically these processes lead to requirements that have to be analyzed and designed from scratch, even if software reuse is applied during implementation

Software Invariant continued

- One technology that aids in this process is components
  - A component is an executable unit of software with well-defined functionality (services) and communication protocols (interfaces) that can be assembled with other components into a software system
- Examples: CORBA, .Net, EJB
- This technology does not attack the essence but it eliminates some accidental difficulties (while introducing some of its own) which allows humans more time to attack the essence on their own

Maciaszek's Three

- Maciaszek identifies three contributors to accidental difficulties
  - Stakeholders
  - Process
  - Modeling language and tools

  (Brooks focuses mainly on tools but would not disagree with Maciaszek's points)
- developers must be aware of the role these entities play, along with the benefits and limitations of each

Stakeholders

- Two groups
  - Customers
  - Developers
    - Analysts, Designers, Programmers, etc.
- Main causes of software failures
  - because customers often do not know what they want and developers are sometimes not up to the task
  - See pages 3 and 4
- "Great designs come from great designers"

Process (Software Lifecycle)

- Process for:
  - Order of activities
  - Product delivery (what, when)
  - Task Assignment to developers
  - Monitoring → measuring → planning
- Cannot be codified or standardized
  - Each organization has a different process
- Process is dependent on project size
  - In the Mythical Man-Month, Brooks talks about
  - a project (OS/360) that required 1000 developers!
- Iterative and incremental
  - These are good characteristics but only if well managed; otherwise it is too easy to slip into "ad hoc hacking"

Modeling Language and Tools

- Stakeholders make use of a process to develop "modeling artifacts," e.g. the requirements, specifications, and design of a software system
  - As such, they require a language to build these artifacts; in turn, this allows them to share, discuss, and evolve these artifacts over the lifetime of a software development project
- In addition, they need tools that facilitate the construction of models using the language
  - CASE (computer-aided software engineering) tools provide benefits such as a persistent repository, consistency checking, versioning, code generation, and reverse engineering

Language Characteristics

- Support for Abstraction
  - Need to specify requirements and characteristics of a system at different levels of detail
  - Need to be able to say "this diagram provides more detail about this box in that diagram"
- Visual Component
  - Humans have powerful visual systems; our models should try to exploit this power as much as possible
- Declarative Semantics
  - It should allow capturing "procedural" meaning in "declarative" sentences
    - "what" not "how"

UML

- Unified Modeling Language
  - Developed by Booch, Jacobson, and Rumbaugh
    - "the three amigos"
  - Each had their own (competing) object-oriented design methods in the early 90's
- Rational Software hired each of them to combine their methods into one unified method
  - The result is the Rational Unified Process and the notation (or language) used by this (and other) methods is called the Unified Modeling Language

UML, continued

- It is important to realize that the UML is not an OO method
  - it is just a notation, a language that can be used to create modeling artifacts
- The UML provides three types of models
  - State models (that describe static data structures)
  - Behavior models (that describe object collaborations)
  - State change models (that describe the allowed states for a system over time)
- In addition, UML provides architectural constructs (such as packages) for grouping models into collections that can be developed iteratively and incrementally

Software development approaches

- The past
  - Procedural programs
  - Deterministic execution; batch processing
  - Program in control; little interaction with user
- The present
  - Interactive program with user in control
  - Event-driven execution
  - Objects respond to events and perform tasks
- Structured vs. Object-Oriented

Structured approach

- Modeling techniques
  - Data Flow Diagrams
    - Process Modeling
  - Entity Relationship Diagrams
    - Data Modeling
- Problems
  - Sequential and transformational
    - Difficult to respond to changing user requirements
  - Inflexible solutions
  - No reuse

Object-Oriented approach

- Data-centric (evolves around class models)
- Event-driven (supports modern applications)
- Applies benefits of object technology
  - abstraction, encapsulation, reuse, inheritance, etc.
- Follows iterative and incremental process
  - A single model is elaborated through analysis, design, and implementation phases with multiple iterations
  - A single language is used to capture all parts of the model

Object-Oriented Approach

- Problems
  - Semantic gap in case of relational database implementation
    - Because relational model of tables and rows do not nicely map into object classes and instances
  - Project management
    - Iterative/Incremental approach harder to manage than sequential waterfall
  - Solution complexity
    - Object systems can sometimes be more complex (think lots of objects) than structured approaches which has implications for maintainability and scalability

Short-Term Roadmap

- Next Lecture
  - Introduction to Software Life Cycles and Object-Oriented Design Methods
- Then
  - Introduction to Object-Oriented Concepts as they relate to analysis (we will also look at structured requirements techniques to use as a comparision)