Software Engineering

CSC440/640

Prof. Nicholas Schweitzer

Contact Information

Nicholas Schweitzer

- Email: nschweit@carrollu.edu
- Mailbox: Lowry Hall – 105 (I think)
- Office Hours
  - After Class (NH 023) 9:00PM – 10:00PM on Wednesdays
  - Other Times By Appointment

Course Materials

- *Object-Oriented and Classical Software Engineering*, 8th Edition, Stephen R. Schach
- Additional Recommended Reading
  - The Mythical Man-Month: Essays on Software Engineering, Frederick P. Brooks Jr.
  - Software Estimation: Demystifying the Black Art, Steve McConnell

Grading

- Grades will be determined based on the following:
  - Quizzes (35%)
    - First 15 Minutes of Each Class
    - Each Quiz Worth 10 Points
  - Project including Presentation (50%)
  - Reflection Paper (15%)
- Grade Scale
  - Standard Carroll University Grading Scale
  - A – 92-100
  - AB – 88-92
  - B – 82-88
  - BC – 78-82
  - C – 70-78
  - D – 60-70
  - F – 0-60

Introductions

- Name
- What's your background?
- What do you do?
- What do you currently use:
  - Programming Languages (Java/C#/Python/etc.)
  - Software Frameworks (jQuery/Spring/Hibernate/etc.)
  - Development Software (IntelliJ IDEA/Visual Studio/etc.)
- What in software are you currently excited about?
- *What do you want to get out of this class?*
  - Besides a good grade…

Science vs. Engineering

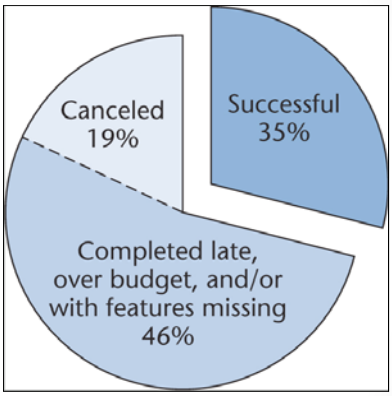
- What is the difference between a *Scientist* and an *Engineer*?
- Scientists:
  - Develop theories and concepts for how things work
  - Don't create things for consumers
  - Aren't necessarily concerned with reliability
- Engineers:
  - Take scientific theories and make products from them
  - Have to make something that other people will use
  - Are concerned with reliability and safety
  - Interested in Process and Repeatability

The Software Crisis

- Hardware keeps getting cheaper and faster. Software gets later and more expensive to develop.
- Why are Projects Late or Canceled?
  - Inaccurate Estimates
  - Poor Quality
  - Inaccurate or Incomplete Requirements
  - Lack of Resources
  - Business Reasons

Standish Group Data

- Data on projects completed in 2006
- Just over one in three projects were successful



Cutter Consortium Data

- 2002 survey of information technology organizations
  - 78% have been involved in disputes ending in litigation
- For the organizations that entered into litigation:
  - In 67% of the disputes, the functionality of the information system as delivered did not meet up to the claims of the developers
  - In 56% of the disputes, the promised delivery date slipped several times
  - In 45% of the disputes, the defects were so severe that the information system was unusable

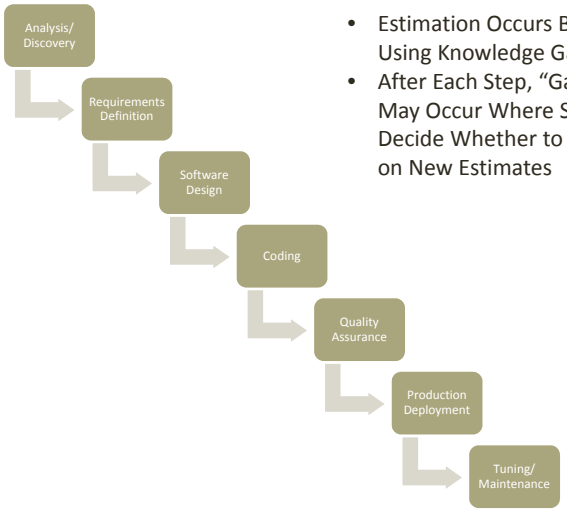
Economic Aspects

- Coding method CM_{new} is 10% faster than currently used method CM_{old}. Should it be used?
- Common sense answer
  - Of course!
- Software Engineering answer
  - Consider the cost of training
  - Consider the impact of introducing a new technology
  - Consider the effect of CM_{new} on maintenance

Maintenance Aspects

- Life-cycle model
  - The steps (*phases*) to follow when building software
  - A theoretical description of what should be done
- Life cycle
  - The actual steps performed on a specific product

Waterfall Model



- Estimation Occurs Before Each Step Using Knowledge Gained in Last Step
- After Each Step, "Gating" Process May Occur Where Stakeholders Decide Whether to Continue Based on New Estimates

Analysis/Discovery

- Objective is to learn about the problem domain
- Who are the users? What are their daily tasks?
- What is the production environment?
- Are there existing systems you must integrate with?
- Is this replacing an existing system? Are you modifying an existing system?
- What are the *pain points* that are being experienced?
- What are the measurable items in the project?
  - System Performance
  - User Load
  - Measurable Project Items – i.e. “Increase the number of applications submitted to the system by 20%”

Analysis Methods

- User Documentation
- Shadowing
- Source Code Analysis
- Prototyping
  - Rapid Prototyping
  - Paper Prototyping
- Focus Groups
- Affinity Diagramming

What Makes a Good Requirement?

- User Stories
  - What User(s) is this requirement for?
  - Short Narrative in Business Terms Describing the Story
- Specific verbiage describing the functionality asked for
  - Complicated requirements should be sub-divided
  - Multiple User Stories can be Organized Into "Epics"
- Associated Screen Mock-Ups
- User Acceptance Criteria
  - Can be used to write user acceptance tests later
- Exception Paths
  - What can go wrong, and what should happen if it does
- Validation Conditions
- Associated Non-Functional Requirements
  - Performance Requirement (i.e. – Form submission shall take less than 3 seconds)

Software Design

- Designing software components to meet Requirements
- Various Methods and Artifacts are Created
- Methods
  - Domain Driven Design (DDD)
  - Test Driven Development (TDD)
  - Behavior Driven Development (BDD)
- Common Design Artifacts
  - Using Unified Modeling Language (UML)
  - Class Diagrams
  - Sequence Diagrams
  - Database Design Diagrams (Object-Relational Mapping)

Coding

- How do you know when you've completed a coding task?
- Coding Includes Unit Testing
  - Unit Test Code Coverage
  - Is 100% Code Coverage Necessary?
- Have you satisfied all of the acceptance criteria laid out in the original requirements definition?

Quality Assurance

- Test Plans Written Based on Requirement Acceptance Criteria
- Regression Tests Run to Ensure Previously Tested Functionality Still Works After Changes Made
  - Automated Tools Make Regression Testing Especially Powerful
  - Selenium
- Performance and Load Testing
- Exploratory Testing
  - Giving QA Engineers Freedom to Try to Break Things
  - What are the values you'd never think of using? Your users *will think to use them!*
- **Remember**: *Unit Tests are Part of Coding – Not Quality Assurance!*

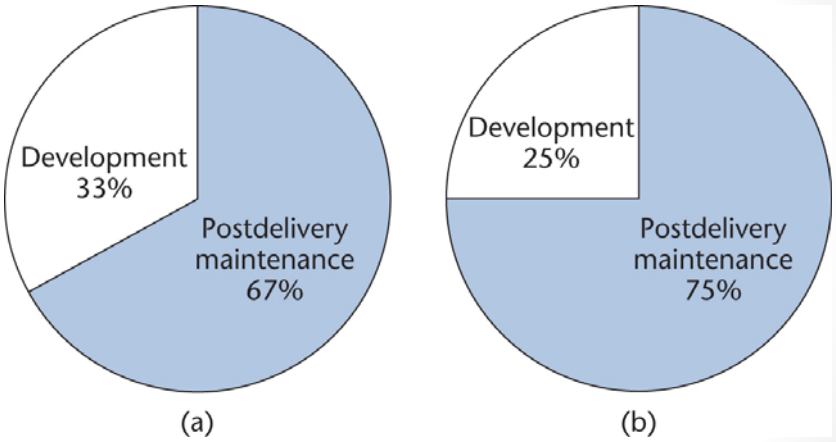
Production Deployment

- Requires Careful Planning
  - How large is your Window for Deployment if Downtime is Required?
  - Do developers perform the deployment?
  - Do you test your deployment in a Production-Like environment?
- Is a Migration Required?
  - User or Other Data Needs to be Moved from an Old System
- Do you have a contingency plan for a failed deployment?
  - Backing Up the Old System Prior to Deployment
  - What needs to be rolled back to get to the original state?
- Production Testing
  - Testing plan needs to determine that all required components were deployed and configured correctly
  - Subset of User Acceptance Tests

Tuning/Maintenance

- Post Production Monitoring
- Analyzing Log Files
- Database Indexes and Tuning
- Bug Fixes
- How Handle Feature Requests?
  - Is Maintenance an ongoing minor task, or should new feature requests be gathered together to turn into a new project?

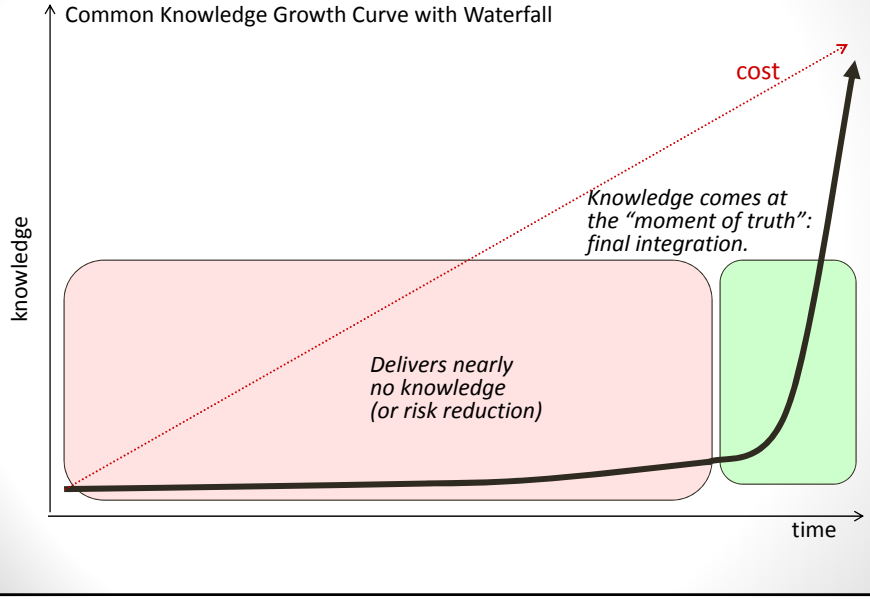
Time (= Cost) of Post-Delivery Maintenance



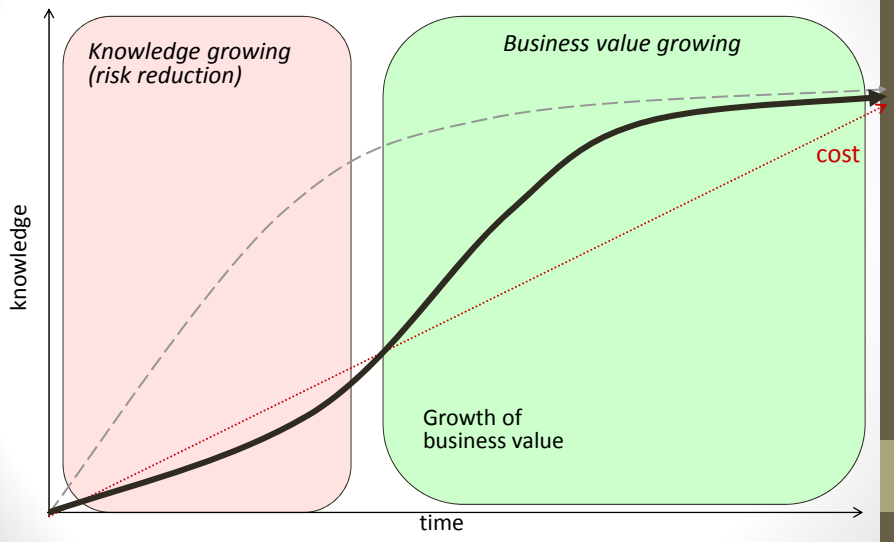
(a) Between 1976 and 1981

(b) Between 1992 and 1998

Late Learning Strategies



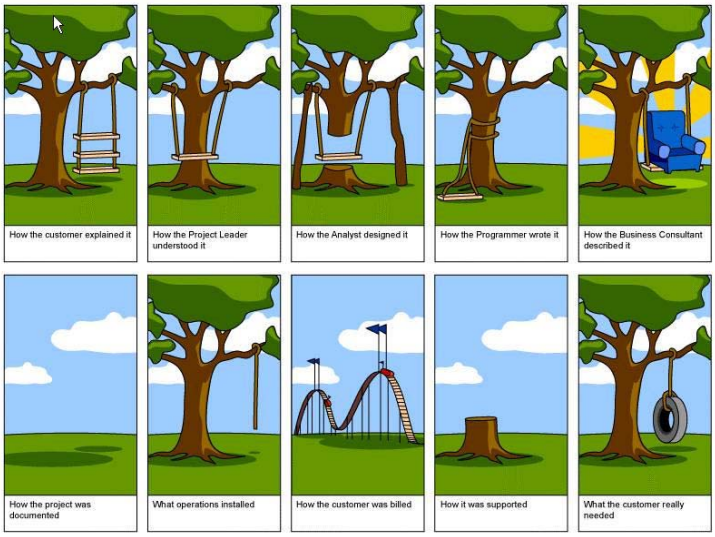
Early Learning Strategies



Team Programming Aspects

- Hardware is cheap
  - We can build products that are too large to be written by one person in the available time
- Software is built by teams
  - Interfacing problems between modules
  - Communication problems among team members

The Communication Problem



Missing Phases

- Why There is No Planning Phase
  - We cannot plan at the beginning of the project —we do not yet know exactly what is to be built
  - Planning is done all the time
- Why There is No Testing Phase
  - It is too late to test after development and before delivery
- Why There is No Documentation Phase
  - Documentation must be kept up to date all the time as changes are made
  - Documentation required for development, testing, and maintenance

Programming Paradigms

- Instructional
  - Assembly Language
- Procedural
  - COBOL, C
- Object-Oriented
  - C++, Java, .NET
- Declarative
  - SQL
- Functional
  - Subset of Declarative
  - Haskell, Lisp, F#

Imperative

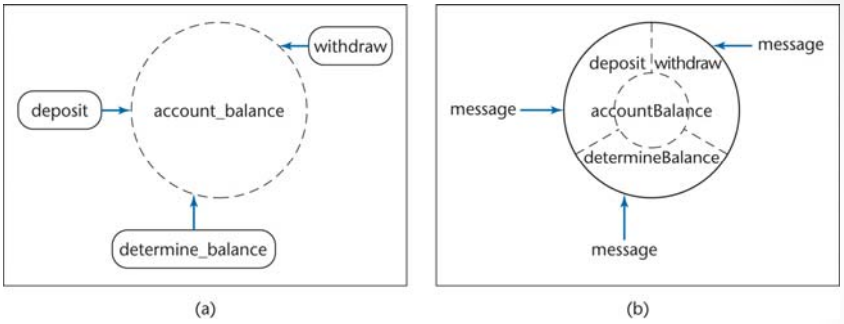
The Object-Oriented Paradigm

- The structured paradigm was successful initially
  - It started to fail with larger products (> 50,000 LOC)
- Post-Delivery maintenance problems (today, 70 to 80% of total effort)
- Reason: Structured methods are
  - Action oriented (e.g., finite state machines, data flow diagrams); or
  - Data oriented (e.g., entity-relationship diagrams, Jackson's method);
  - But not both

The Object-Oriented Paradigm

- Both data and actions are of equal importance
- Object:
  - A software component that incorporates both data and the actions that are performed on that data
- Example:
  - Bank account
    - Data: account balance
    - Actions: deposit, withdraw, determine balance

Structured vs Object-Oriented



- Information hiding
- Responsibility-driven design
- Impact on maintenance, development

Information Hiding

- In the object-oriented version
  - The solid line around `accountBalance` denotes that outside the object there is no knowledge of how `accountBalance` is implemented
- In the classical version
  - All the modules have details of the implementation of `account_balance`

Strengths of the Object-Oriented Paradigm

- With information hiding, post-delivery maintenance is safer
  - The chances of a regression fault are reduced
- Development is easier
  - Objects generally have physical counterparts
  - This simplifies modeling (a key aspect of the object-oriented paradigm)
- Well-designed objects are independent units
  - Everything that relates to the real-world item being modeled is in the corresponding object — *encapsulation*
  - Communication is by sending *messages*
  - This independence is enhanced by *responsibility-driven design* (see later)

Responsibility-Driven Design

- Also called *design by contract*
- Send flowers to your mother in Chicago
  - Call `1-800-flowers`
  - Where is `1-800-flowers`?
  - Which Chicago florist does the delivery?
  - Information hiding
  - Send a message to a method [action] of an object without knowing the internal structure of the object

Analysis/Design "Hump"

- Structured paradigm:
  - There is a jolt between analysis (what) and design (how)
- Object-oriented paradigm:
  - Objects enter from the very beginning
- In the classical paradigm
  - Classical analysis
    - Determine what has to be done
  - Design
    - Determine how to do it
    - Architectural design — determine the modules
    - Detailed design — design each module

Removing the "Hump"

- In the object-oriented paradigm
  - Object-oriented analysis
    - Determine what has to be done
    - Determine the objects
  - Object-oriented design
    - Determine how to do it
    - Design the objects

In More Detail

Classical Paradigm	Object-Oriented Paradigm
2. Analysis (specification) phase • Determine what the product is to do	2′. Object-oriented analysis workflow • Determine what the product is to do • Extract the classes
3. Design phase • Architectural design (extract the modules) • Detailed design	3′. Object-oriented design workflow • Detailed design
4. Implementation phase • Code the modules in an appropriate programming language • Integrate	4′. Object-oriented implementation workflow • Code the classes in an appropriate object-oriented programming language • Integrate

- Objects enter here

Object-Oriented Paradigm

- Modules (objects) are introduced as early as the object-oriented analysis workflow
  - This ensures a smooth transition from the analysis workflow to the design workflow
- The objects are then coded during the implementation workflow
  - Again, the transition is smooth

Object-Oriented Terminology

- Data component of an object
  - State variable
  - Instance variable (Java)
  - Field (C++)
  - Attribute (generic)
- Action component of an object
  - Member function (C++)
  - Method (generic)

Object-Oriented Terminology

- C++: A member is either an
  - Attribute ("field"), or a
  - Method ("member function")
- Java: A field is either an
  - Attribute ("instance variable"), or a
  - Method

Project Management Myths

Myth: *If a task takes 4 hours for one person to complete, than it should take 2 hours for two people to complete.*

Reality: *9 Women Cannot Make a Baby in 1 Month.*

Project Management Myths (2)

Myth: *If a project is late, I can add resources to get it back on schedule.*

Reality: *Adding more resources to a late project makes it later*

Next Week

- Read Chapters 2 & 3
  - Recommended to Do Exercises
- First Quiz
  - 1st 15 Minutes of Class
- Team Selection
  - Ensure that Class Roster is Complete
- Project Definition