Software Tools & Methods
In4mtx 111, CSE 121

Lecturer: Greg Bolcer, greg@bolcer.org

Summer Session 2009

ELH 110 9am-11:50am

Course home page

Http://www.ics.uci.edu/~gbolcer/inf111/

- Linked from homepage and EEE
- Syllabus, slides, homework assignments, lab assignments all posted there
- Lecturer: Greg Bolcer, greg@bolcer.org (please call me “Greg”)
- Teaching Assistant: Hye Jung Choi, hchoi7@uci.edu

Course mechanics

- Lectures – ELH 110
  - June 23rd through July 23rd
  - 10 weeks of classes into 10 classes
  - T,Th 9:00am – 11:50am, two 10 minute breaks, 9:50-10, 10:50-11
- Discussions – ELH 110
  - Tuesday 1:00pm – 1:50pm
  - Best place to do homework
  - okay to turn in early, but won't get graded until deadline
- Labs – ICS 192
  - Thursday 1:00pm – 1:50pm
  - Hands on experience with software tools
  - Assignments due during lab session, signed off by TA, "just follow the instructions"
  - Don't forget the take home portion of the assignments, always due following Tuesday

Objectives

- Learn about software tools with hands on, interactive experience
  - Eclipse, Subversion, Junit, Rational Developer Workbench, Scrumworks, ArgoUML
  - Familiarity with tools used in real world, aka “Job” and good for your resume
- Experience with methods
  - Ideas more important than particular tool
  - When and when not to use a tool

Grading

- Everyone gets an “A”……*if* you
  - Show up to class
  - Show up to all discussions
  - Show up to all labs
  - Read the assigned chapters
- Midterm and Final exams will be a combination of multiple choice, short answer, long answer taken from class, homework, labs, and text
- Homework and Labs 60%, 6 homeworks, 1 extra credit
- Midterm test 15%
- Final exam 25%

Assignments & Labs

- Using software tools
  - Get accustomed to tools early (try them out before going to the lab)
  - All tools are installed in the CS lab
  - Freeware and open source tools can be installed at home, but make sure TA can read output files and/or screen shots
  - Specific tools will be required, i.e. Rational Software Architect, not MS Visio
- Assignments submitted electronically
- Quality more important than length
- Express yourself clearly

Important Dates

- Midterm – July 9th 9:00am
- Final – in class July 23rd 9:00am

Topic List

B = Brooks, L = Larman, V = van Vliet. Schedule is subject to change.

	Topic	Readings	Evaluations
June 23	Course Overview - Introduction - Software Technology - Orders of Ignorance - Nature of Software Development - "No Silver Bullet"	B16 (or "No Silver Bullet" or this more readable version that is available from campus IP addresses) V15 (Chapter 19 in Second edition)	
June 25	Software Process - Process models - Plan-based models - Iterative models Programming Practices - Coding conventions - Code Reading - Reverse Engineering	V3, 14.3 Programming Best Practices	HW1 due HW2 Lab
June 30	Programming Practices (continued) Configuration Management -Tools: Subversion 	V4, 14, 15.3.2	HW2 due
July 2	Configuration Management (continued) Unified Modeling Language (UML) - Modeling - Perspectives in Modeling - Domain Models - Class Diagrams - Tools: Rational Software Developers Workbench	L1, 9, 13, 14, 16	HW3 due HW3 Lab
July 7	Design Patterns - Singleton - Observer - Façade - Factory - Strategy - Composite UML (continued)	L26	
July 9	Midterm	L3	HW4 due HW4 Lab
July 14	Iterative Software Development - Agile - Unified Process (UP) Iterative Software Development (continued) Use Cases	L2, 6	HW6 due
July 16	More UML - Sequence diagrams - State chart diagram - Activity diagram	L10, 15, 28, 29	HW6 Lab
Julyl 21	Testing - Types of testing - Acceptance testing - Unit testing  Tools: JUnit	V13	HW7 due
July 23	Review Software Industry		HW8 due HW7 Lab

 Guest lectures – real world use of tools and methods

Be involved

- Come to class
- Ask questions
- Best place to do homework is in discussion and labs
- If you attend lectures, do the assignments and read the text, you will do well on the exams

Cheating

- Students caught cheating will
  - Get a letter in your UCI file
  - Have the potential to fail the class

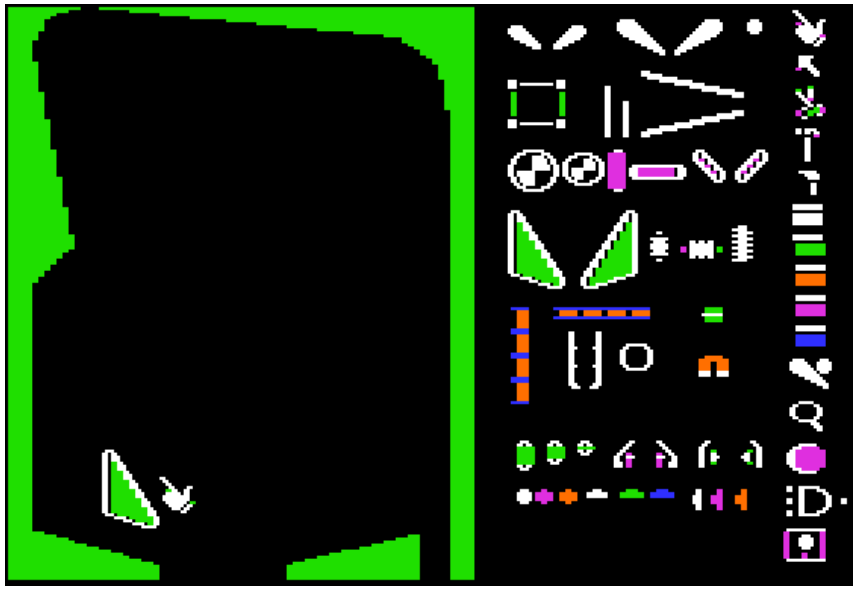
- Discussing an assignment is okay, copying the solution is not
- Plagiarism – all text copied from books, articles, or even Google must be cited (just say where you got it)

Whiteboard Exercise (tools)

- What is the most interesting piece of software you have used?
- What made it interesting?
- What is the most interesting piece of software you have worked on?
- What would you do differently if you could do it again?

Whiteboard Exercise (Greg)

- Pinball Construction Set (1983)
- What made it interesting?
  - First Rapid Application Development tool and dynamic Runtime config
- What is the most interesting piece of software you have worked on?
  - Magi – micro-Apache Generic Interface, peer to peer Web tool
- What would you do differently if you could do it again?
  - Build a Myspace Web front end on top of it to simplify the task for non-techie users
  - Would have been Napster and Myspace all rolled up into one application



Whiteboard Exercise (method)

A chicken and a pig are walking down the road trying to figure out what they are going to do with their lives.

Chicken: I got it! We'll open a restaurant!

Pig: Great, what will be call it?

Chicken: We'll call it Ham & Eggs

What is the difference between the Pig and the Chicken with respect to their project?

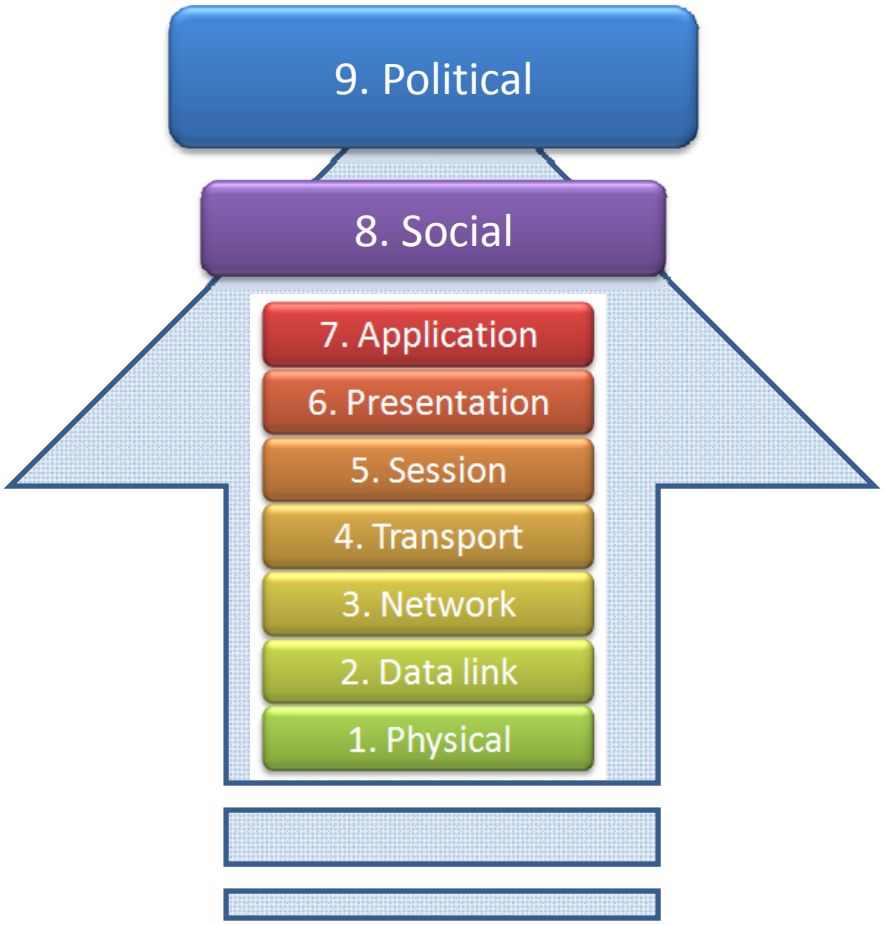
Whiteboard Exercise (method)

- The chicken only has to contribute to make the project successful, but the Pig's neck is on the line
- Used to describe roles in "SCRUM" development process
- An iterative, incremental framework for managing complex work
- What other animals are metaphors?

Overview

- Introduction to software technology
  - Building software
  - Why do we need software tools and methods?
  - What are software tools and methods?
- Information theory
  - Acquisition of knowledge and orders of ignorance
- No Silver Bullet
- Software Process

Building Software is Hard



- Building anything more than a toy/hobby program is an inherently social activity
- “Layer-8” and “Layer-9” problems are issues with developers and users
- Tools and Methods are used to limit risk, reduce fallout, and guide users towards a successful completion of their project
  - Building the right thing and completing it under budget and on time

Orders of Ignorance

INFORMATION THEORY

Øth Order Ignorance (ØOI)

- I have Zeroth Order Ignorance (ØOI) when I know something and can demonstrate my lack of ignorance in some tangible form
- ØOI is provable and proven knowledge that can be deemed correct by some qualified agency
- In software, this means the knowledge is invariably factored into some usable form
- In all forms of knowledge, there must be some external proof element that qualifies the knowledge as being correct
- Examples
  - Trivia
  - Building a system that satisfied the user
  - Ability to sail

2nd Order Ignorance (2OI)

- Lack of awareness
  - I have 2OI when I am not aware that I don't know something
  - Not only am I ignorant of some knowledge (1OI), I am unaware of what it is I am ignorant about
  - I don't have enough information to know what it is that I don't know
- I can't give a good 2OI example, of course
  - How do I calculate my tax liability for my will? Ø

3rd Order Ignorance (3OI)

- Lack of process, i.e. you don't know how to go about discovering the knowledge
- No suitably efficient (reasonably time limited) help is at hand to help guide your "journey of discovery" (there's always Google, Wiki right?)
  - Some "build it and see what happens" is good for exploratory development
  - Not very suitable in the real world when your job is on the line

4th Order Ignorance (4OI)

- 4OI is meta-ignorance of "orders of ignorance" model
  - **0th Order Ignorance (0OI)** – Lack of Ignorance: I know something
  - **1st Order Ignorance (1OI)** – Lack of Knowledge: I don't know something
  - **2nd Order Ignorance (2OI)** – Lack of Awareness: I don't know that I don't know something
  - **3rd Order Ignorance (3OI)** – Lack of Process: I don't know of a suitably efficient or systematic way to find out that I don't know that I don't know something
  - **4th Order Ignorance (4OI)** – Meta Ignorance: I don't know about the Five Orders of Ignorance
- Knowledge is inherently recursive, you must know about other things that define what you know

Importance of Ignorance in Software

http://www.ics.uci.edu/~gbolcer/inf111/ignoramus.pdf

- A failing project, ignorance hiding as a way to get to the root of the requirements
- Problems with the startup that prevented the engineers to come together on the requirements document
  - All using the same vocabulary in slightly different ways
  - None were aware of others tacit assumptions
  - Each “wallowing” in their own pit

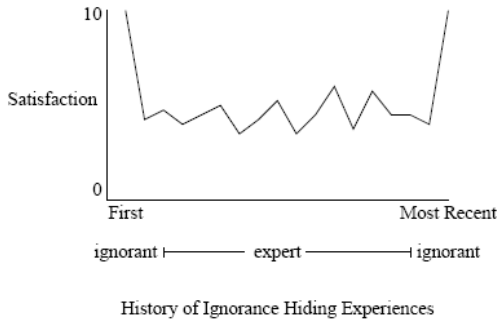
Ignorance Hiding in Requirements

- Iterate the requirements discovery process until all inconsistencies are resolves and the following two principles are true for each requirements

1. The requirements for the system must be modified and refined until the system can be developed by software designers and coders who are familiar with the fundamental algorithms of the app domain, but who are ignorant about the requirements engineering process. The eventual requirements will be so complete an unambiguous that the design will be clear and that the coding can be done easily by implementing standard, well-known algorithms
2. The requirements for the system must be modified and refined until the system requirements can be understood by requirements engiineers who are knowledgeable about requirements engineering, but who know nothing about the application domain. The eventual requirements will be so complete and unambiguous that the requirements engineer will understand them completely, even without knowledge of the application domain.

Ignorance is the Key

- By being ignorant of the application area, Dan was able to
  - Ferret out and make explicit assumptions
  - To regard the ever-so-slight differences in terminologies
- Conclusion: Every requirements team requires at least one person who is ignorant in the application domain
- Conclusion: We still need experts



History of Ignorance Hiding Experiences

Essence and Accident in Software Engineering

NO SILVER BULLET

Whiteboard Exercise (silver bullet)

- What is a silver bullet?
- How does it relate to software?
- What does it mean to have no silver bullet?
- Who coined the term?
- When might we expect a silver bullet in software?

Whiteboard Exercise (silver bullet)

- What is a silver bullet?
  - Any straightforward solution perceived to have extreme effectiveness.
- How does it relate to software?
  - Is there some great technological discovery yet to be invented that will take away all the inherent problems with building software?
- What does it mean to have no silver bullet?
  - "there is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude [tenfold] improvement within a decade in productivity, in reliability, in simplicity."
  - "We can't expect to see twofold gain in two years."

Whiteboard Exercise (silver bullet)

- Who coined the term?
  - Fred Brooks, 1986; author of the Mythical Man Month; No Silver Bullet Refired, 1995
- When might we expect a silver bullet in software?
  - An order of magnitude over 40 years might be achievable with several techniques in tandem
- Summary: There is no magic 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 Essence

- Brooks divides the essence into four subcategories
  - complexity
  - conformity
  - changeability
  - Invisibility
- Lets consider each in turn

Complexity

- Software entities are amazingly complex
  - No two parts (above statements) are alike
    - Contrast with materials in other domains
  - They have a huge number of states
    - Brooks claims they have an order of magnitude more states than computers (e.g. hardware) do
  - As the size of the system increases, its parts increase exponentially

Complexity continued

- Problem
  - You can't abstract away the complexity
    - Physics models work because they abstract away complex details that are not concerned with the essence of the domain; with software the complexity is part of the essence!
  - The complexity comes from the tight interrelationships between heterogeneous artifacts: specs, docs, code, test cases, etc.

Complexity continued

Problems resulting from complexity

- difficult team communication
- product flaws
- cost overruns
- schedule delays
- personnel turnover (loss of knowledge)
- Un-enumerated states(lots of them)
- lack of extensibility(complexity of structure)
- unanticipated states (security loopholes)
- project overview is difficult (impedes conceptual integrity)

Conformity

- A significant portion of the complexity facing software engineers is arbitrary
  - Consider a system designed to support a particular business process
  - New VP arrives and changes the process
  - System must now conform to the (from our perspective) arbitrary changes imposed by the VP

Conformity continued

- Other instances of conformity
  - Non-standard module or user interfaces
    - Arbitrary since each created by different people not because a domain demanded a particular interface
  - Adapting to a pre-existing environment
    - May be difficult to change the environment
    - however if the environment changes, the software system is expected to adapt!
- It is difficult to plan for arbitrary change!

Changeability

- Software is constantly asked to change
  - Other things are too, however
    - manufactured things are rarely changed
      - the changes appear in later models
      - automobiles are recalled infrequently
      - buildings are expensive to remodel
- With software, the pressures are greater
  - software = functionality (plus its malleable)
    - functionality is what often needs to be changed!

Invisibility

- Software is invisible and un-visualizable
  - In contrast to things like blueprints
    - here geometry helps to identify problems and optimizations of space
  - Its hard to diagram software
    - We find that one diagram may consist of many overlapping graphs rather than just one
      - flow of control, flow of data, patterns of dependency, etc.
- This lack of visualization deprives the engineer from using the brain's powerful visual skills

What about X?

- Brooks argues that past breakthroughs solve accidental difficulties
  - High-level languages
  - Time-Sharing
  - Programming Environments
- New hopefuls
  - Ada, OO Programming, AI, expert systems, “automatic” programming, etc.

Promising Attacks on Essence

- Buy vs. Build
  - Don’t develop software at all!
- Rapid Prototyping
  - Brooks buys in
- Incremental Development
  - grow, not build, software
- Great designers

No Silver Bullet Refired

- Brooks reflects on the “No Silver Bullet” paper, ten years later
  - Lots of people have argued that there methodology is the silver bullet
    - If so, they didn’t meet the deadline of 10 years!
  - Other people misunderstood what Brooks calls “obscure writing”
    - For instance, when he said “accidental”, he did not mean “occurring by chance”

The Size of “accidental” effort

- Some people misunderstood his point with the “9/10ths” figure
  - Brooks doesn’t actually think that accidental effort is 9/10th of the job its much smaller than that
  - As a result, reducing it to zero (which is probably impossible) will not give you an order of magnitude improvement

Obtaining the Increase

- Some people interpreted Brooks as saying that the essence could never be attacked
  - That's not his point however; he said that no *single technique could produce an order of* magnitude increase by itself
- He argued that several techniques in tandem could achieve that goal but that requires industry-wide enforcement and discipline

The Bottom Line

- Brooks states
  - “We will surely make substantial progress over the next 40 years; an order of magnitude over 40 years is hardly magical…”
- Essence of software development
  - Complexity
  - Conformity
  - Changeability
  - Invisibility

Whiteboard Exercise (cloud computing)

- Hosting services in the cloud so don't have to build up and operate infrastructure and computing
- Addresses Accidental or Essential?
- A type of rapid prototyping for certain types of applications?
- What other techniques can it be combined with?