Harvey Mudd College

CS 121
Software Development
Spring 2001

Professor Bob Keller
keller@cs.hmc.edu
621-8483

Office Hours (1242 Olin):

- Note: 1242 is inside of 1240, near the SE entry
- Monday 3-4 p.m.
- Tuesday 4-6 p.m.
- Wednesday 3-4 p.m.
- Any other time you can find me, which is almost any time, including evenings, except Friday.
- Test whether I'm here using email or phone: keller@cs.hmc.edu, x 18483

CS 121 Grader/Tutor

- Eric Huang
- ehuang@cs.hmc.edu

Text

- None required.
- See web page for a list of optional ones.
- Web reading will be assigned.

CS 121 Topics

- Development processes
- Requirements analysis
- Design (primarily object-oriented)
- Design patterns
- Project organization and management
- Software specification, formal & informal
- Verification and testing
- Cost estimation
- Standards (UML, CORBA, XML, etc.)

Course Work

- Large portion will be in teams of 4 students each.
- Practice in requirements analysis, specification, design, coding, documentation, walkthroughs, etc.
- Significant final project
- Exams or quizzes as needed

Grading Breakdown

- Attendance & participation: 15%
- Team project: 40%
- Weekly assignments: 30%
- Late mid-term exam: 15%

Some Motivation for Systematic Study of Software Development

- Big, serious, business, in addition to being academically enjoyable.
- The results may have *global* impact, even unexpectedly.
- The challenges are many (reliability, cost reduction, ergonomics, …)

Towards "Software Architecture" (1)

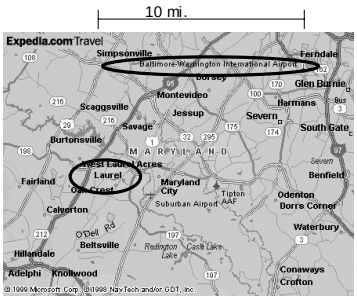
- Building-architecture has achieved its stature because it deals with large and expensive systems that affect the lives of many people.
- It has developed methodologies and standards for design.

Towards "Software Architecture" (2)

- Software now often falls into this category of being large, expensive, and affecting the lives of many people.
- The methodologies and standards for software architecture are in their infancy compared to those of building architecture.

Designing and Implementing Good Software is Hard

Humorous Software Flaws:
A Route Advising System

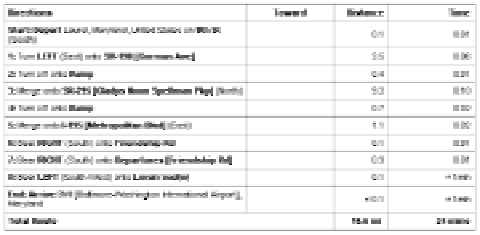


Example reported in THE RISKS DIGEST 20.62, Oct. 1, 1999

Excerpts from **Expedia Maps** directions:
From: **Laurel, Maryland**
To: **Baltimore-Washington International Airport, Maryland**
Driving Distance: 5865.1 miles
Time: 9 day(s) 3 hour(s) 22 minute(s)

Time (**hour**:minute)	Instruction
0:00	Depart Laurel, Maryland
1:01	Entering Delaware
1:17	Entering New Jersey
3:24	Entering New York
3:51	Entering Connecticut
5:51	Entering Massachusetts
7:29	Entering New Hampshire
7:44	Entering Maine
12:20	Entering New Brunswick
20:20	Take the North Sydney-Argentia Ferry
34:32	Entering Newfoundland
36:35	Turn left onto Local road(s) (4543.1 mi)
219:22	Arrive Baltimore-Washington International Airport, Maryland

Revised Result on the web today



Serious Software Flaws

Therac-25 Accelerator Treatment Facility
(see IEEE Computer, Vol. 26, No. 7, July 1993, pp. 18-41.)

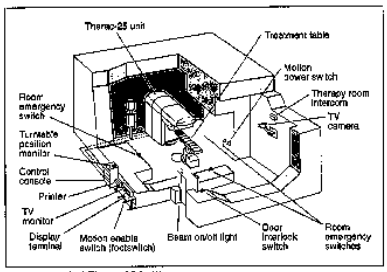


Figure 1. Typical Therac-25 facility.

Therac-25: Overdoses caused by "incorrect software"

- Six massive-overdose accidents between 1985 and 1987.
- Incidents (dose of > 1000 rads can be fatal):
  - 3 June 1985: Marietta, Georgia, patient receives overdose (est. dose: 15,000-20,000 rads).
  - 26 July 1985: Hamilton, Canada, patient severely burned, later dies November 3, 1985 (est. dose: 13,000-17,000 rads)
  - 21 March 1986: Tyler, Texas, patient receives overdose, patient dies later (est. dose: 16,500 - 25,00 rads).
  - 11 April 1986: Tyler, Texas, another patient receives overdose, patient dies in 3 weeks (est. dose: 4,000 rads).
  - 17 January 1987: Yakima, Washington, patient receives overdose, patient dies 3 1/2 months later (est. dose: 8,000 - 10,000 rads).
- Recalled in 1987 for extensive design changes, including hardware to safeguard against software errors.

Therac-25 Software Errors
(source: http://kachina.kennesaw.edu/~mking/courses/is8070/lectures/ethics4.html)

- The user interface was weak: error messages were cryptic. Operator's manual for the Therac-25 did not contain a reference to error messages. Some errors would pause the machine waiting for the operator to press a key to resume. It was assumed these errors were trivial. Other errors forced a restart of the system: these errors were assumed to be serious. This intuitive assessment of the errors was not valid. The overdoses occurred when the system paused.
- One specific error was the incrementation of an integer counter in a one-byte address. When the system was reset for the 256th time, the counter indicated 0, signifying a valid status for the component being tested.
- Another specific error occurred when the system failed to properly check for input from the keyboard. If the operator wanted to make changes to the set up, these could be done while parts of the system were initializing. Parts of the system received the modified set-up instructions, and other parts did not. Experience users were more prone to invoke this error.

(Scene from a play "Therac 25")

Therac 25 is the story of Alan and Moira, two twentysomethings who meet in the basement of the old Princess Margaret treatment centre. Over three weeks of radiation and chemo therapies, they become involved. (source: http://www.eye.net/eye/issue/issue_08.07.97/theatre/theatre.html)

Federal Aviation Agency Advanced Automation System

- Announced in 1981, to modernize air-traffic control.
- IBM awarded contract in 1989 after 4 year bid process. Estimated 1.5 million lines of code, **$2.5 billion**, to be deployed by 1991.
- Estimate increased to **$4.3 billion** in 1987, deployment slipped to 1995.
- Determined in 1994 that the **project would never be completed,** and the project was cancelled.
- Scaled-down version awarded to Loral (which bought IBM FSD) at estimated cost of $1.5 billion, to be deployed by 1997. The project was completed two weeks early.

New Denver Airport (1)

BAE Automated Systems, Inc.

New Denver Airport (2)

- Contract of $193 million in June 1992 to begin work on the baggage-handling system.
- Involves 100 computers, 56 laser scanners, 400 radio systems.
- Airport lost $1 million per day upon opening.
- Baggage system failures:
  - Continued to unload bags despite jam on conveyor belt.
  - Loaded bags onto full carts, causing bags to fall onto tracks.
  - Bags wedged under carts due to timing problems.
  - Lost track of carts themselves, due to above types of incidents.

USS Yorktown dead in water after divide by zero (http://www.csl.sri.com/neumann/risks-new.html)

- Navy's Smart Ship technology is considered a success, because it has resulted in reductions in manpower, workloads, maintenance and costs for sailors aboard the Aegis missile cruiser USS Yorktown.
- However, in September 1997, the Yorktown suffered a systems failure during maneuvers off the coast of Cape Charles, VA., apparently as a result of the failure to prevent a divide by zero in a Windows NT application.
- The zero seems to have been an erroneous data item that was entered manually. Atlantic Fleet officials said the ship was dead in the water for about 2 hours and 45 minutes.

Transition

- The preceding examples are a few of many.
- What can we do about it?
- "Get our software act together."
- Software development processes strive for means to manage quality development.

Components of Software Development

- Requirements elicitation
- Requirements analysis
- Requirements specification

"Requirements"

- Modeling and design

"Design"

- Implementation (coding)
- Validation, verification, testing
- Maintenance and upgrade
- Configuration management

"Implementation"

- Assessment

"Assessment"

Requirements Analysis

Before software is developed, it is important to **specify** clearly the requirements for the ultimate system, in order to:

- estimate the costs involved
- serve as a starting point for design
- provide a reference point for the verification of results

Specification

In order to carry out analysis, design, and evaluation in rigorous terms, it is important to have a clear specification of the system, using, for example:

- structured forms of English
- specification languages
- clearly-stated assumptions

Models and Design

For larger systems, with many facets, it is important to have models, design methods, and tools that

- fit well with the software specification techniques
- provide a framework in which development proceeds
- permit tracing from implementation back to initial requirements

Implementation

- Implementation concerns the development of code modules that constitute a system.
- *Ab initio* implementation is increasingly cost-prohibitive.
- *COTS* (commercial, off-the-shelf) software is not a panacea; often not sufficiently customizable.
- *Standardized reusable modules*, especially ones that have been formally specified and certified, are more economical in the long run.

Validation

Validation refers to ascertaining that software systems, once developed, meet the requirements initially specified. This topic covers:

- Mathematical verification methods
- Formal testing methods
- Management techniques for paths from requirements to testing and verification

Examples of Emerging Architectural Techniques

- Specialized architectural (as opposed to programming) languages, such as **UML**
- Software tools for
  - Requirements management
  - Configuration management
  - Design tools
- Standards
  - Layered distributed architectures (DCOM)
  - Object repositories and brokers (CORBA)
  - Parallel processing software architectures (MPI)

Four P's of Software Development

- Product
  - What is being developed
- People
  - Who does the developing
- Process
  - How are products developed, what do the people do
- Patterns
  - Repeatable approaches that work

Software Development Process

(not to be confused with "processes" in the sense of concurrent operating-system tasks)

Process

- A process focuses on *how* something is done, rather than what is being done.
- The next few slides list *possible components* of a software development process, but not necessarily in the order in which those components are executed.

Components of a Software Development Process (1 of 6)

- Program Construction
  - Writing the program, debugging
  - *All* projects have this component.

Components of a Software Development Process (2 of 6)

- Program Validation
  - Establishing, as thoroughly as is feasible, that the program performs as desired by the customer
  - All *worthwhile* projects have this component.

Components of a Software Development Process (3 of 6)

- System Design
  - Determining structural aspects of the program or system *prior* to programming
  - *Most* successful and within-budget projects of significant size have this component.

Components of a Software Development Process (4 of 6)

- Requirements Specification
  - Specifying how system is to behave
  - Occurs prior to design or programming
  - Most *funded* projects will have this component.

Components of a Software Development Process (5 of 6)

- Requirements *Elicitation*
  - Getting the client's view of what the requirements are, through dialog
  - Typically less "technical" than specification

Components of a Software Development Process (6 of 6)

- Requirements *Analysis*
  - Translating the customer dialog into a specification

Ordered Summary of a Software Development Process

- Requirements
  - Elicitation
  - Analysis
  - Specification
- System Design
- Program Construction
- Validation

How a typical developer might spend his/her time

- __% interacting with customer, management, other developers
- __% writing requirements, specification, design
- __% writing code
- __% testing

How a typical developer might spend his/her time

- 20% interacting with customer, management, other developers
- 20% writing requirements, specification, design
- 40% writing code
- 20% testing

(your mileage may vary)

Requirements

(Elictation, analysis, specification, documentation, etc.)

Terminology

- SRS: "Software Requirements Specification"
- The SRS should contain all and only information that *defines* the software product.
- The SRS should not contain ancillary information about how the work is to be conducted, although this might be part of a contract that *refers* to the SRS.

Typical Elements of a Software Requirements Specification (SRS)

- **Background information**
  - Type of product and its purpose
  - Intended users of product
  - Glossary of terms, both domain-specific and product-specific
- **"Functional" requirements**
  - Behavioral descriptions of software use, including how exceptional circumstances are to be handled.

Elements of an SRS (cont'd)

- **"Non-functional" requirements**
  - Performance requirements (speed, memory use, disk space)
  - Constraints, including security requirements
  - Collateral requirements (other software)
  - Hardware platforms supported
  - User documentation to be provided
  - Maximum-size requirements (for input data, etc.)

Not in an SRS

- Acceptance tests to be used
- Cost estimate of doing the project
- Delivery schedule
- Design process to be used
- Development plan, milestones
- Management structure
- Market analysis
- Stakeholders in the project

SRS Example: Meeting Scheduler

- See handout or
  http://www.cs.hmc.edu/courses/2001/spring/cs121/MeetingSchedulerSRS.html

Requirements != Product Design

- Requirements are the "what", not the "how".
- They dictate the **problem**, not the solution.
- Requirements typically **don't** specify the internal structure of the product.
- They *might* specify the programming language to be used.
- They *might* specify that a specific design notation, such as UML, must be available as a by-product.

Ways to "capture" requirements

- Customer writes fully (rare).
- Interview customer (elicitation) (common).
- You write, customer approves.
- Iterative combination of the above.

Potential Mismatch

- The customer's language might not be your preferred language.
- It will typically be non-computer-ese.
- The customer's and users' needs, rather than the designer's or implementor's favorite approaches, should be what drives the project.

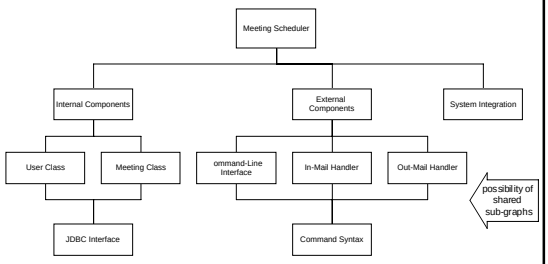
First Assignment Due next class

- Two problems: For both problems, submit **typed**, not hand-written, solutions.
- Problem 1: Read the Meeting Scheduler SRS distributed. Then provide a critique:
  - Which parts, if any, don't belong in an SRS?
  - Which parts are sufficiently clear to enable a design to proceed; which are too vague for this purpose?
  - Identify any parts that are mutually contradictory.
  - Identify any items that seem to be needed but are missing.

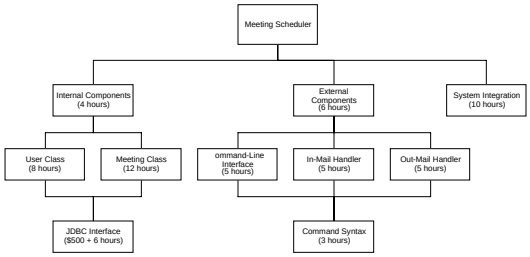
First Assignment Due next class

- Problem 2:
  - The instructor is the customer and has a product he would like to see developed.
  - As a typical customer, he has a vague idea about what it is.
  - Interview him in class today to find out the requirements.
  - Write-up the requirements as an SRS in a readable form, for presentation at the next class.
  - Your write-up should survive the critique points of part 1.

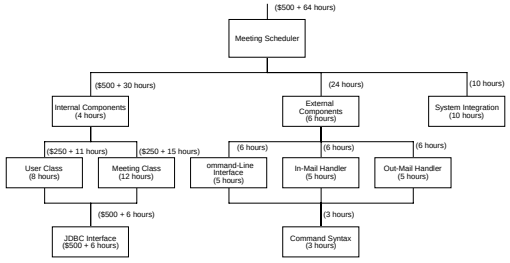
Work Breakdown Structure (WBS)



WBS with Cost Annotations



WBS with Cost Roll-Up



Additional Requirements for Cost Rollup Not Illustrated

- Provide 2 Sharing Models
  - Even split among parents
  - Pass exclusively through a selected parent
- Provide dollar cost summarization based on labor rate.

Resist temptation

In constructing your SRS, please resist the temptation to introduce elements of internal structure and design into the requirements.