Harvey Mudd College

CS 121
Software Development
Spring 2005

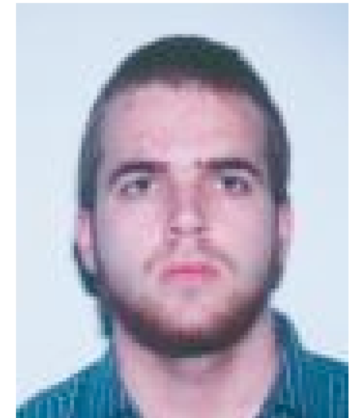
Professor Bob Keller
keller@cs.hmc.edu
1249 Olin
621-8483

Office Hours (1249 Olin):

- Note: 1249 is in the southwest corner of Olin
- MW 4-5, TuTh 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 Graders/Tutor

- Jonathan Dodge '06
  x72055, 406 West Hall, jdodge@cs.hmc.edu



- Mjumbe Poe '05
  x18827, 176 East Hall, mpoe@cs.hmc.edu

Text

- Required:
  - John P. Flynt,
    Software Engineering for Game Developers,
    Thomson Course Technology, 2005,
    ISBN 1-59200-155-6

What is Software Development (aka "Software Engineering")?

- Software development is an end-to-end process that includes:
  - **Specification** of a software product
  - **Design** of a system to meet the specification
  - **Implementation** of the design
  - **Quality Assurance** of the implementation
  - **Maintenance** of the implementation

Why study Software Development?

- Society has become increasingly dependent on software systems.
- Failures in software systems can be dangerous and costly.
- Software development is a complex problem.

CS 121 Topics

- Development processes
- Requirements analysis and specification
- Design (primarily object-oriented)
- Modeling (using UML: Unified Modeling Language)
- Design patterns
- Project organization and management
- Software specification, formal & informal
- Verification and testing
- Cost estimation

Will we get to develop games?

- We will follow recent departmental practice in this course of using games as a vehicle for software development.
- It must be kept in mind that this is not a course on game development, but rather a course on software development in general.
- Therefore, there will be some activities that do not relate to games specifically.
- If you object to using games as the vehicle, you should tell me up front.

Course Work

- Projects will be in teams of 3 students each.
- Teams will remain intact for the entire semester.
- There will be practice in requirements analysis, specification, design, coding, documentation, code inspections, etc.
- Classroom participation is required.
- There will be an oral mid-term exam on software development concepts about 2/3 of the way through.

Grading Breakdown (tentative)

- 10% First project
- 20% Second project
- 30% Third project
- 15% Other homework
- 15% Oral mid-term exam
- 10% Attendance & participation

Note

- Do not expect every topic discussed to be directly relevant to your particular project.
- We wish to educate on things that will be useful after this course, not just within it.
- We are interested in *education* more than *training*.
- We are not interested in only games.

Some Motivation for Systematic Study of Software Development

- Big, serious, business, in addition to being professionally and academically enjoyable.
- The results may have *global* impact, even unexpectedly.
- The challenges are many:
  - reliability,
  - economics,
  - ergonomics, ...

Designing and Implementing Good Software is Hard

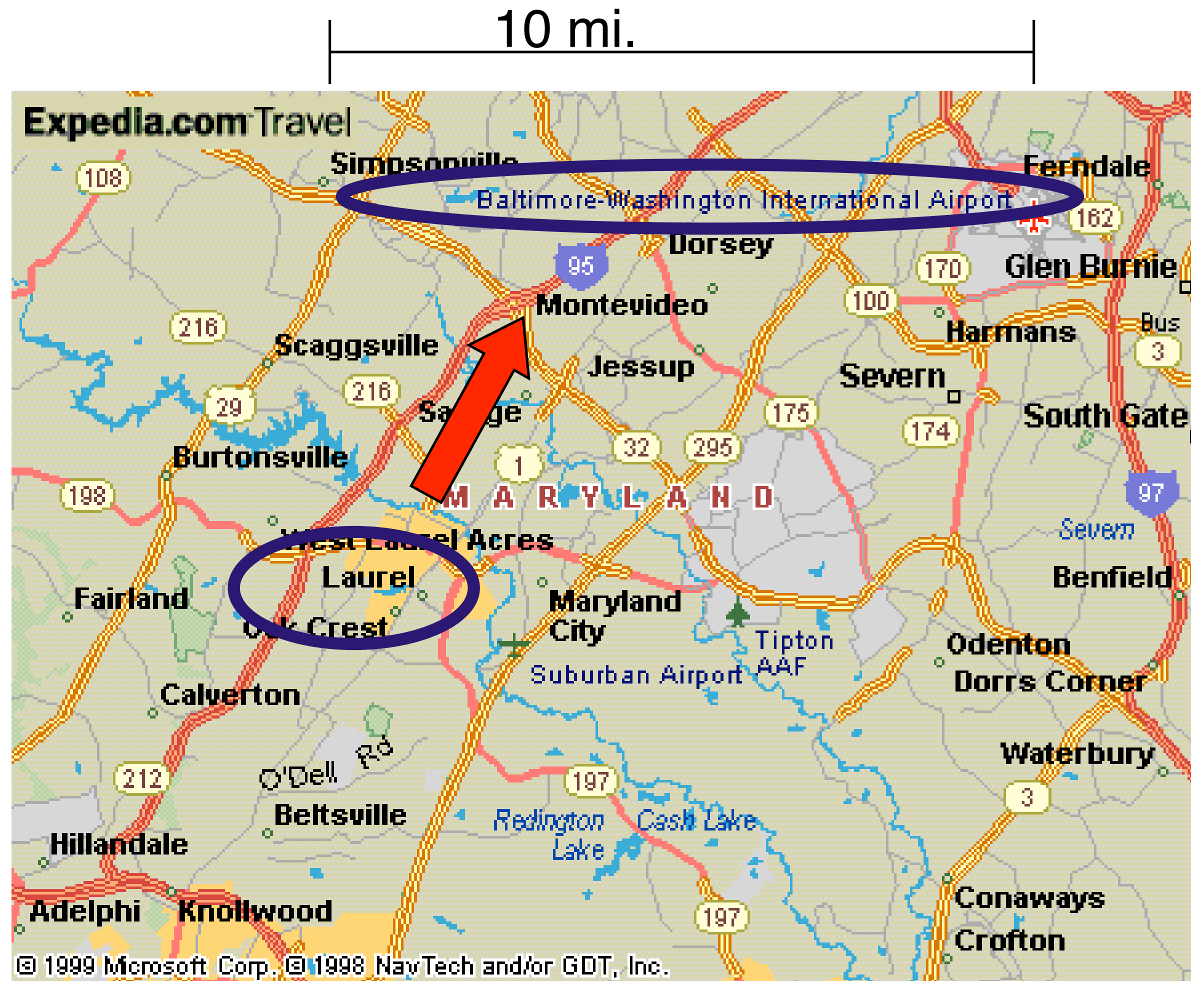
as evidenced by
many failures
to do it right

Humorous Flaws

in

Commercial Software

Example 1:
Early Expedia Route Advising System



Output 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

Repaired version on the web today

Directions	Toward	Distance	Time
Start: Depart Laurel, Maryland, United States on **9th St** (South)		0.1	0:01
1: Turn **LEFT** (East) onto **SR-198 [Gorman Ave]**		3.5	0:06
2: Turn off onto **Ramp**		0.4	0:01
3: Merge onto **SR-295 [Gladys Noon Spellman Pky]** (North)		9.2	0:10
4: Turn off onto **Ramp**		0.7	0:02
5: Merge onto **I-195 [Metropolitan Blvd]** (East)		1.1	0:02
6: Bear **RIGHT** (South) onto **Friendship Rd**		0.1	0:01
7: Bear **RIGHT** (South) onto **Departures [Friendship Rd]**		0.3	0:01
8: Bear **LEFT** (South-West) onto **Local road(s)**		0.1	< 1min
End: Arrive BWI [Baltimore-Washington International Airport], Maryland		< 0.1	< 1min
Total Route		**15.6 mi**	**24 mins**

Example 2:
Code Reuse can be Harmful?

The Australian military had a combat simulator, essentially a video game for helicopter training. It included roving packs of kangaroos, because pilots need to consider that disturbing wildlife can betray their position, according to the Defense Science and Technology Organization. To get the kangaroos into the simulator, programmers had to model the animals' reactions to the helicopters.

Code was reused to save time. The programmers took code from another simulator that had modeled the movement of infantry troops and essentially dropped the kangaroo images on top of it.

And it worked like a charm. When the program was demonstrated, the virtual helicopter comes buzzing by, and the kangaroos stop grazing, and they go hopping, over the hill and out of sight...

... only to reappear seconds later, firing projectiles at a very surprised helicopter pilot.

modified from: http://www.govtech.net/magazine/gt/2000/mar/lastbytesfolder/lastbytes.php

Example 3:
8 character limit on file names

- Bill Gates' idea of a joke on the world?

Not-So Humorous Software Flaws

Example 1:

Microsoft's Passport authentication system

Software flaws in the security of Microsoft's Passport authentication system left consumers' financial data wide open, causing the software giant to remove a key service from the Internet to protect people from having their data stolen, a company representative acknowledged on Friday.

The admission came after an open-source programmer demonstrated serious security flaws in Wallet – the Passport service that keeps track of data used by e-commerce sites. Microsoft shut down the service Thursday, casting a pall on the company's recent efforts to convince consumers that it is serious about security. The incident also undermined the software giant's claims that its Passport system can keep customers' financial data safe.

source: `http://news.zdnet.co.uk/story/0,,t269-s2098563,00.html`
5 November, 2001.

Example 2:

Therac-25 Accelerator Treatment Facility, 1983

(see IEEE Computer, Vol. 26, No. 7, July 1993, pp. 18-41.)

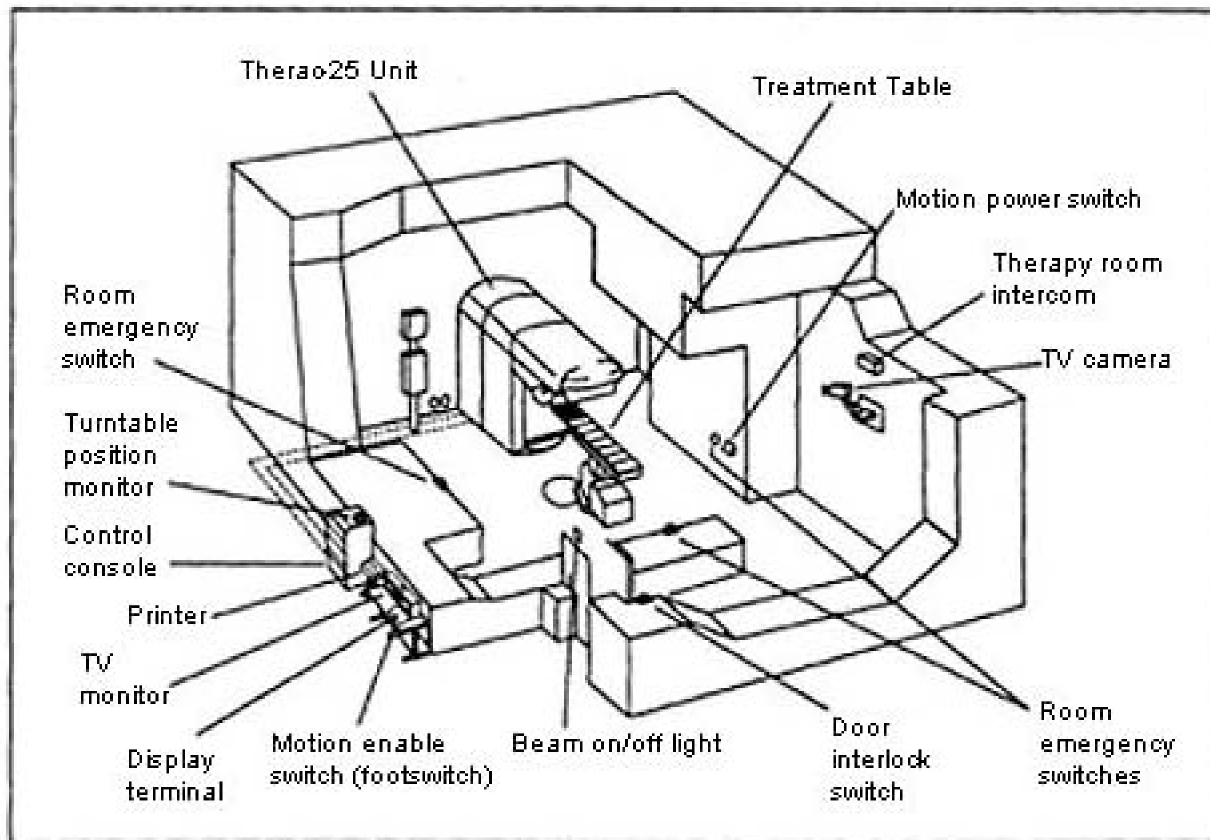


Figure 1. Typical Therac-25 facility

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, dies later (est. dose: 16,500 - 25,00 rads).
  - 11 April 1986: Tyler, Texas, another patient receives overdose, dies in 3 weeks (est. dose: 4,000 rads).
  - 17 January 1987: Yakima, Washington, patient receives overdose, 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.

Analysis of Therac-25 Software Errors

(source: http://kachina.kennesaw.edu/~mking/courses/is8070/lectures/ethics4.html)

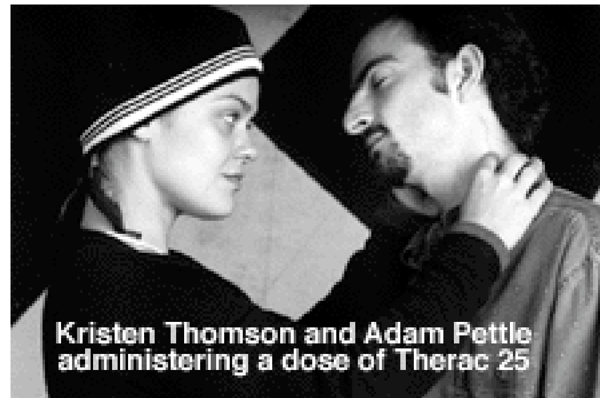
- The user interface was weak: error messages were cryptic. Operator's manual for the Therac-25 did not contain a reference to error messages. Some errors would pause the machine waiting for the operator to press a key to resume. It was assumed these errors were trivial. Other errors forced a restart 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. Experienced users were more prone to invoke this error.

Artistic Spin-Off:

Scene from a play inspired by (?) "Therac 25"

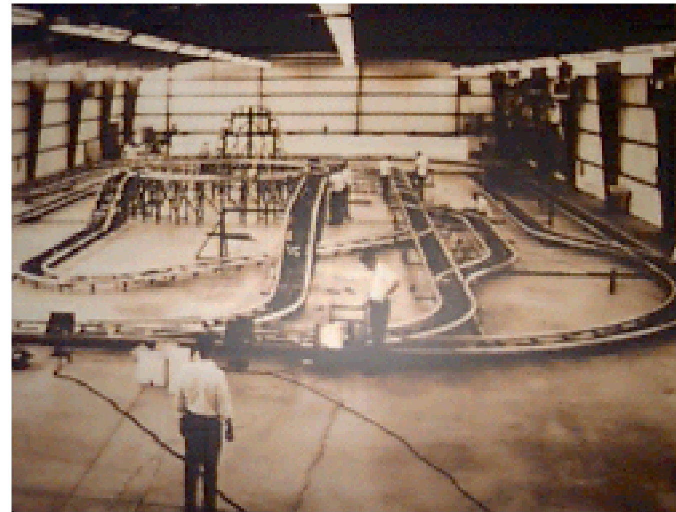

Kristen Thomson and Adam Pettle administering a dose of Therac 25

Therac 25 is the story of Alan and Moira, two twenty-somethings 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)

Example 3:
FAA 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 revised project was completed two weeks early.

Example 4:
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.
- Involved 100 computers, 56 laser scanners, 400 radio systems.
- 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.
- Airport lost $1 million per day upon opening.

Example 5:
USS Yorktown dead in water after divide by zero
(http://www.csl.sri.com/neumann/risks-new.html)

- Navy's Smart Ship technology was considered a success, because it had 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 of flaws are a few of many.
- What can we do about it?
- Software development processes must strive for management of quality development.

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.

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
  - Data representation standards (XML)
  - Object repositories and brokers (CORBA, SOAP, …)
  - Layered distributed architectures (DCOM)
  - Parallel processing software architectures (MPI)

Facets of Software Development

- Requirements elicitation
- Requirements analysis
- Requirements specification
- Modeling and design
- Implementation (coding)
- Validation, verification, testing
- Maintenance and upgrade
- Configuration management
- Assessment

"Requirements"

"Design"

"Implementation"

"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* ("components") especially ones that have been formally specified and certified, may be more economical in the long run.

Validation

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

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

Sample Components of a Software Development Process

Process

- 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

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

Components of a Software Development Process

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

Components of a Software Development Process

- 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

- 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

- 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

- Requirements *Analysis*
  - Translating the understanding derived from requirements elicitation 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

- 30% interacting with customer, management, other developers
- 20% writing requirements, specification, design
- 30% writing code
- 20% testing

(will vary, depending on environment)

Requirements

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

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 product is to be constructed or developed, although this might be part of a contract or plan that *refers* to the SRS.
- Adding the second type of information as a requirement might overly constrain the product construction, preventing the best techniques from being used.

Requirements ≠ 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 that a certain programming language be used (because source is a deliverable).
- They ***might*** specify that a specific design notation, such as UML, must be available as a by-product.

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.)
  - What language?
  - What artifacts?

Not in an SRS (why?)

- 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 Examples

- Walris game: See handout
- Google on: "software requirements specification"
- Make notes on variety and range of specifications
- Note that some hits will be templates rather than actual specifications

Ways to “capture” requirements

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

Potential Mismatch

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

First Assignment
Due next class meeting

- Problem 1 of 1:
  - The instructor is the client and has a software product he would like to see developed.
  - As a typical customer, he has an idea that may be specific in some areas, vague in others.
  - 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.
- In constructing your SRS, please resist the temptation to introduce elements of internal structure and design into the requirements.