Software Development

Harvey Mudd College

CS 121

Spring Semester 2008

Prof. Bob Keller

Data

- Bob Keller
- Office: 1253 Olin
- Phone: x 1-8483
- Office Hours: M, T, W: 4:15-5:30, or by appointment, or by drop-in
- Unavailable: MTWTh: 2:45-4, TTh: 1:15-2:30, TTh 11-12, M 9-10:30, Fri AM

"Software Development"

- Called "Software Engineering" (SE) most places.
- At HMC, the "E" word is owned by another interest group.
- Prior to 1992, was "Large-Scale Software Development" (LSD).

Software Development vs. Programming

- **Requirements**
- **Planning**
- **Specification**
- **Architecture**
- **Design**
- *Programming*
- **Validation**
- **Assessment**
- **Maintenance**
- **Management**

A Crude Analogy

- Programming : Software Development
- Plumbing : Building Construction

Importance of Software Development

- Society increasingly relies on software to accomplish its tasks, for leisure activities, and for communication in general.
- Business
- Transportation
- Defense

Overview of the Field

- SWEBOK =

  "Software Engineering Body of Knowledge"
- http://www.swebok.org/
- Download 200 page PDF

Career Considerations

- Many people educated in engineering or the sciences

  end up doing software development in some form.

- Would they be better off having devoting a greater share of their education to the topic?

Scientific Considerations

- Is there a **science** to software development?
- SD is becoming increasingly more scientific, with new research and tool development. The field is young.
- There are still elements of **art**, as well as science.

Richard Feynman in Lectures on Computation

- "Computer science also differs from physics in that it is not actually a science. It does not study natural objects. Neither is it, as you might think, mathematics; although it does use mathematical reasoning pretty extensively. Rather, computer science is like engineering - it is all about getting something to do something, rather than just dealing with abstractions as in pre-Smith geology."

- [Does "science" need to study "natural objects"?]
- [What is "pre-Smith" geology?]
- [Who was Richard Feynman? Did he ever work for a computer-related company?]

For the Theorist:

- Software development is often the "proving point" of computer science.

- Having a result published in a paper is nothing like producing a working product based on the result.

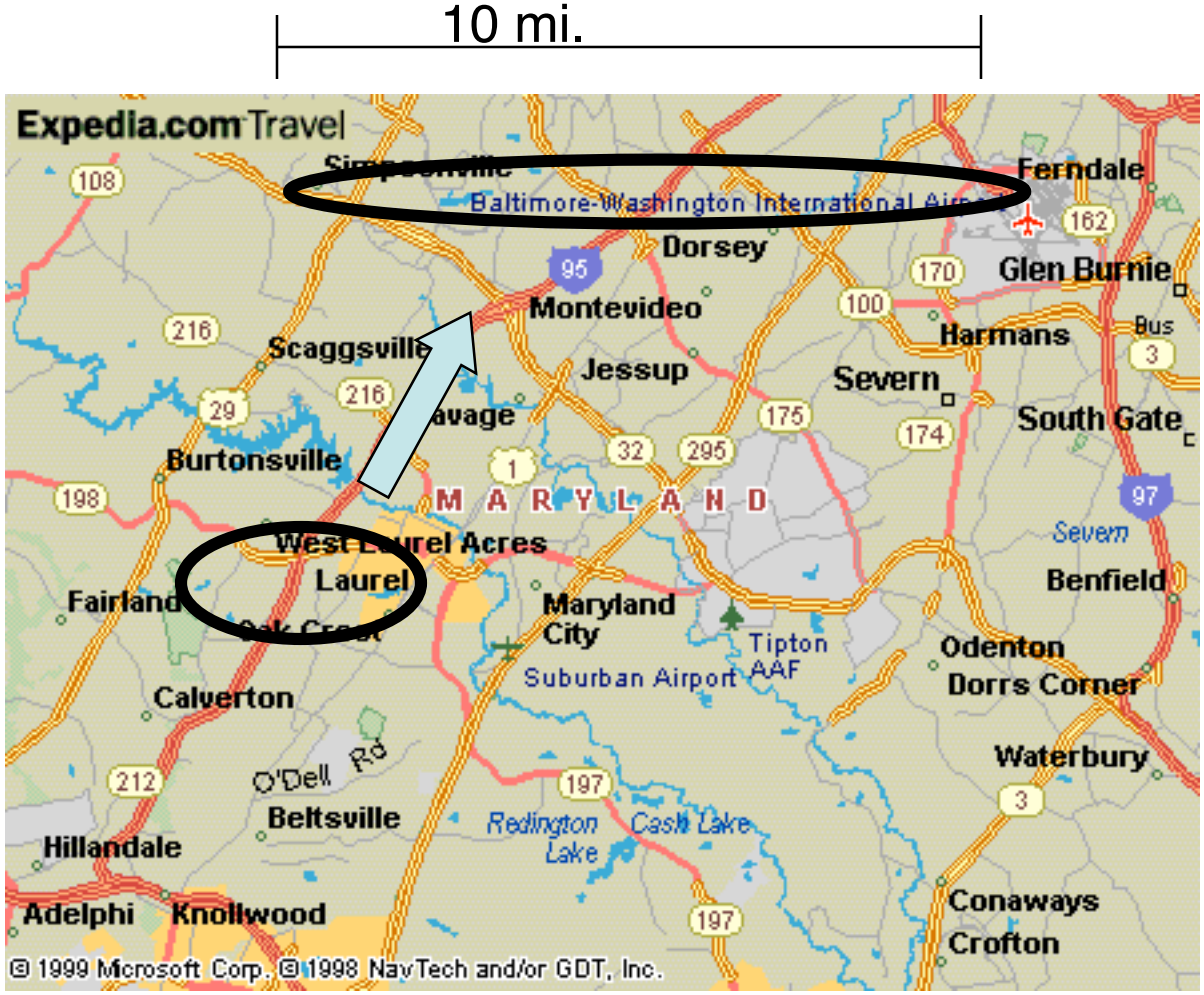
Do the techniques of SD apply to other fields?

A Common Issue

- The **management of complexity** is common to several areas.
- Some information systems probably qualify as the most complex systems ever built.
- *Un-needed* complexity is not a feature; SE attempts to prevent it.

Global Impact of a Single Software Development Project

Example:
An Early Route Advising System



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

Excerpts from **Expedia Maps (Microsoft)** 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

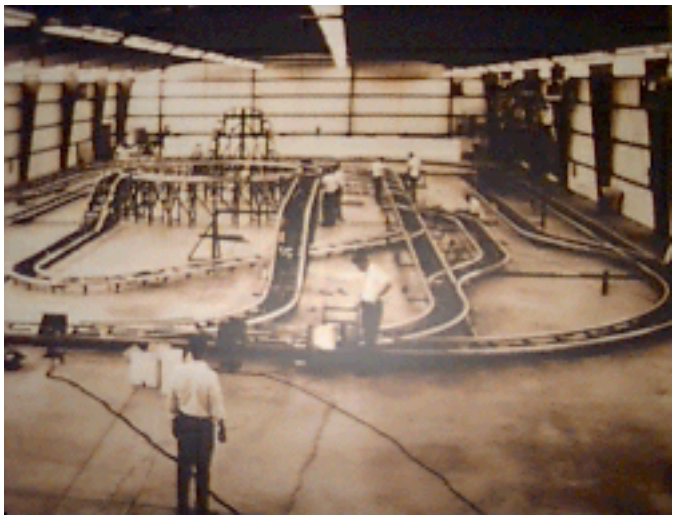
Current Version

Directions	Distance	Time
Start: Depart Start on SR-198 [Gorman Ave] (East)	3.5	0:06
1: Take Ramp (RIGHT) onto SR-295 [Baltimore Washington Pkwy] (MD-295 / Balt./Wash. Pkwy. / Baltimore)	9.5	0:10
2: Take Ramp (RIGHT) onto I-195 (I-195 / Bwi Airport)	1.8	0:03
3: Road name changes to Local road(s)	0.1	0:01
4: Bear RIGHT (South) onto Local road(s) (Arriving Flights / Hourly Parking)	0.4	0:02
5: Turn RIGHT (South) onto Local road(s)	0.1	< 1min
End: Arrive End	< 0.1	< 1min
Total Route	**15.4 mi**	**22 mins**

Example:
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:
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.

Therac-25 Accelerator Treatment Facility

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

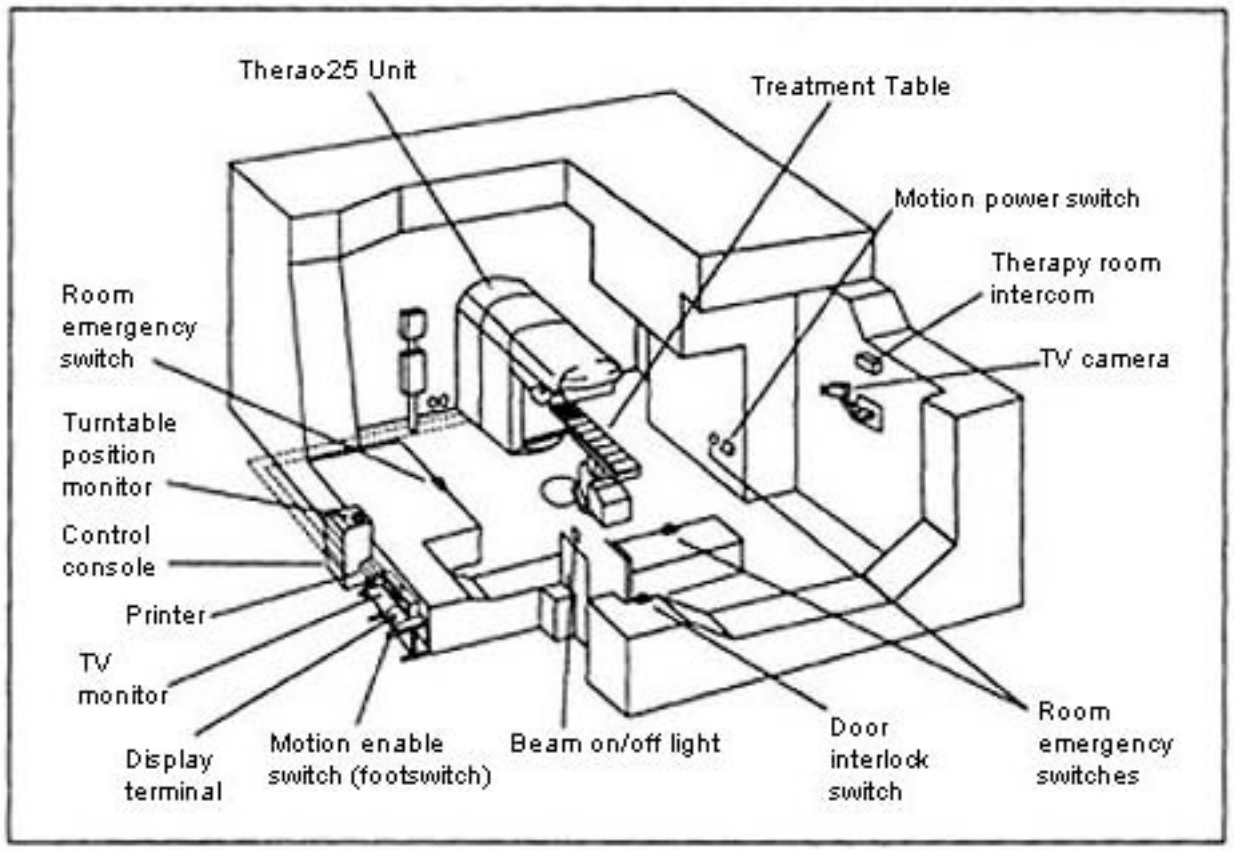


Figure 1. Typical Therac-25 facility

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.

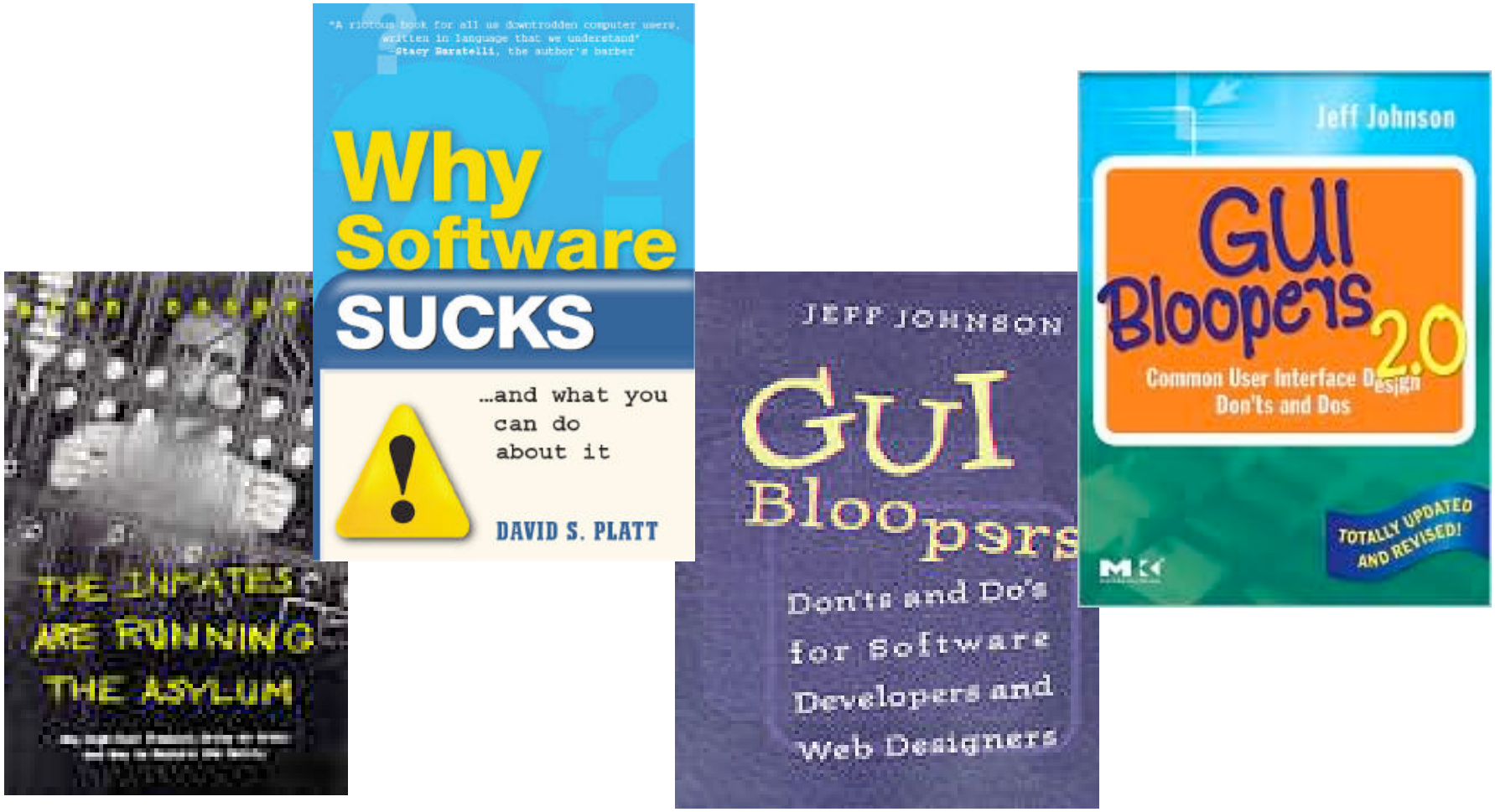
- Another specific error occurred when the **system failed to properly check for input from the keyboard**. If the operator wanted to make changes to the set up, these could be done while parts of the system were initializing. Parts of the system received the modified set-up instructions, and other parts did not. Experienced users were more prone to invoke this error.

Risks Digest

- Consult the Risks digest for many other examples.

- http://catless.ncl.ac.uk/Risks/

Usability Concerns



Transition

- The preceding examples of flaws are a few of many.
- What can we do about it?
- Software development **processes** must promote management of quality software development.

Class Activity (5 mins)

- Brainstorm a list of the **three most critical** problems that you think face a software development organization.
  - First make your own list.
  - Second, convince a partner that the items on your list are the most critical.
  - Form a combined list, again restricting to what you think are the three most critical problems.
  - Transcribe your list of three to a viewgraph slide.
  - Put your and your partner's name at the top of the slide.

Facets of Software Development

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

more loosely

"Requirements"

"Design"

"Implementation"

"Assessment"

"Maintenance"

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 validation 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
- "Use Cases", scenarios
- 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

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
  - Object repositories and brokers (CORBA)
  - Layered distributed architectures (DCOM)
  - Parallel processing software architectures (MPI)

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; it often is 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:

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

Software Development Process

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

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, debugging, unit testing
  - *All* projects have this component.

Components of a Software Development **Process**

- 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**

- 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 might a typical SE spend his/her time?

__% interacting with customer, management, other developers

__% writing requirements, specification, design

__% writing code, debugging

__% testing

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 statement of work (SOW) 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.

Fred Brooks' "No Silver Bullet" paper

- "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work cripples the resulting system if done wrong. No other part is more difficult to rectify later."

Economic Importance of Requirements

- Conventional wisdom: Cost of addressing incorrect or absent requirement "skyrockets" as project approaches deployment.

Project phase	**Relative cost** to correct requirement defect.
Requirements	1%
Design	3%
Code	8%
Test	18%
Operation	110%

Software Requirements Specification (SRS)
Typical Elements

- **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

An SRS Example: CHIMMP

- See

  http://www.cs.hmc.edu/courses/2002/spring/cs121/chimmp/web/CHIMMP_SRS.html

Desirable Properties of an SRS

- Consistency: No conflicting requirements.
- Completeness: All requirements present.
- Clarity: Unambigous statements.
- Coherent presentation
- Organized Hierarchically: Group related topics.
- No Gold-Plating: (no unnecessary frills).

Desirable Properties of an SRS, continued

- Feasible (e.g. does not try to solve the halting problem).
- Written so that requirements are ultimately traceable to and testable in the final product.
- Prioritized by
  - Degree of necessity
  - Desired completion order
- Evolvable

FURPS+ mnemonic
software product properties ("ilities")

- **F**unctionality - Feature set, Capabilities, Generality, Security
- **U**sability - Human factors, Aesthetics, Consistency, Documentation
- **R**eliability - Frequency/severity of failure, Recoverability, Predictability, Accuracy, Mean time to failure
- **P**erformance - Speed, Efficiency, Resource consumption, Throughput, Response time
- **S**upportability - Testability, Extensibility, Adaptability, Maintainability, Compatibility, Configurability, Serviceability, Installability, Localizability, Portability
- The model, developed at Hewlett-Packard, was first publicly elaborated by Grady and Caswell. The + was later added to the model after various campaigns at HP to extend the acronym to emphasize various attributes. "FURPS+ is now widely used in the software industry."

source: wikipedia

Ways to "capture" requirements

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

Requirements Elicitation Exercise

- Instructor plays the role of client.
- Class members play the role of requirements elicitor.
- Write-up an SRS for the next class meeting.

Requirements Engineering

- The requirements engineer is a specialist that provides interfaces between both stakeholders and developers.
- This is elaborated on the following 3 slides.
- Source:

  http://cs.wwc.edu/~aabyan/435/Requirements.html

Client and Other Stakeholders

- may not be able to accurately describe what they do (knowledge may be tacit),
- may not know what is technically feasible,
- may change their minds once they see the possibilities more clearly, and
- often do not appreciate the complexity inherent in software engineering, and the impact of changed requirements

Developers desire specifications that are:

- clear,unambiguous, and, where appropriate, quantitative.
- nonprescriptive
- correct, complete, concise
- precise and verifiable
- consistent
- traceable
- modifiable

Requirements Engineer has

- technical skills,
- ability to acquire an understanding of the application domain, and
- interpersonal skills to help build consensus between heterogeneous groups of stakeholders.

Potential Mismatch

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

Roger Pressman's RE Checklist

- Is there a written request for the work to be done?
- Does the written request outline the problem to be solved, the output to be produced, and the input that will be required?
- Does the written request have a single named author?
- Does the named author have authority to make decisions about the request?
- Is the person doing requirement elicitation familiar with the business area under consideration?
- Is the person doing requirement elicitation familiar with local jargon, business issues, personnel and politics?
- Has a formal meeting been requested with the customer?
- Has an agenda for the meeting been established; have ground rules for the meeting been defined; have all people who need to attend been invited?
- Has a "definition mechanism" been defined?
- Is the customer willing to attend the meeting(s)?
- Have context-free questions (SEPA, 5/e, p. 275) been asked and answered?
- Have written usage scenarios been developed as part of the elicitation activity?
- Has the customer walked through the usage scenarios? Have inaccuracies and modification been specified?
- Have basic output objects been fully defined?
- Have input objects been defined?
- Have the functions to be implemented been defined, decomposed and tied to usage scenarios?
- Has the behavior of the system been fully defined (from the customer's viewpoint) under various operating conditions?
- Have user interface characteristics been discussed and defined?
- Has earned value been determined for customer requirements?
- Have constraints and limitation been identified?
- Have requirements been ranked by importance to the customer?
- Have review(s) been conducted for all information collected as part of requirements elicitation?