CSC 402
Requirements Engineering

Fall Quarter, 2005

Clark S. Turner
Administration

• Instructor
  – Clark S. Turner

• Required Books
  – Wiegers, Software Requirements
  – Jackson, Software Requirements and Specifications
  – Yourdon, Death March

• Other References
  – Gause and Weinberg, Software Requirements
  – Weinberg, The Psychology of Computer Programming

• Office: 14-211
  – phone (805) 756 6133
  – Hours (tentative):
    € Monday 1:10 pm - 3 pm
    € Friday 12:10 pm- 3 pm

• Prerequisites
  – permission of instructor
    € 205, 206, 305 (recommended!)

• Course website at:
  – www.csc.calpoly.edu/~csturner
  – course details and lecture slides
    € updates mayl be weekly
Basic Overview of the Course

• We’re going to elicit requirements
  – from a “rea”l customer: Trimble
    € anyone in here experienced with GPS?
    € the project involves customer interface with proprietary boards
      • some closed source
      • IP issues: expect an agreement and NDA’s
        – we will have to be security conscious
        – we are at the edge of a real project - “proof of concept” prototype at the least
        – teams each need to initiate an agreement between members

• We’ll form 5 teams (of 5 or 6 people)
  – with a manager, and other job titles
    € all responsible for the work products (evaluations of personal effort)

• The course is about process and product:
  – our final deliverable is a Requirements Specification (to give to 405)
    € we also plan a basic architecture
Basics (continued)

• This course requires personal responsibility
• This course requires teamwork, interpersonal skills
• This course requires clear, concise, precise writing
• There will be a steady workload
  – your process will determine the amount of pain :-)  
    € DO read Yourdon “Death March” cover to cover (early!)
• It is a real project with real customers
  – and all that entails: the customers do NOT know all the answers
    € neither will I
  – we’re all in this together, ideally for the coming 3 terms
• We expect to be flexible but to create some working product
Basics (continued)

• Evaluation will be wholistic, based on a large picture
  – quality of deliverables
  – presentations and reviews
  – team performance
    € self evaluation and team evaluation of each member’s performance
    € team dynamics are very, very important
  – homework
  – final exam
  – final team interview with instructor

• I expect to give “A’s” but it will take serious commitment
How to Use a Textbook

• Look at front and back covers
• Read Preface, Intro
• Review TOC, and look for glossary and index
• Ask questions
  – what is the pedigree of the author?
  – why did the author write this book?
  – why did the instructor choose this book?
  – what can I actually expect to get from this book?
My Background

• Mathematics - pure theory
• Law - contracts
• Requirements analysis at UC Irvine
  – worked on TCAS for FAA
  – worked on Therac-25 case with FDA
  – dissertation says that you can’t objectively tell the difference between design and implementation for code
  € continuing work in the area of software code defects involved in personal injury:
    • failure to satisfy specifications
    • specifications that take unreasonable risks with human lives
The Basic Definition of our Work

- Software Engineering is...
  - the study of software process, software development principles, methods and tools
    - requirements elicitation and analysis
    - requirements and design notations
    - implementation strategies
    - testing methods
    - maintenance techniques
    - management strategies
  - the production of *quality software*, delivered *on-time*, within *budget*, and *satisfying users’ needs*

Find other definitions of “software engineering”
What is a “Program” (only one of the objects of Software Engineering…)

• A static description of a dynamic process to be instantiated in the future (Turner)
  – how strange is that?
Why This Course Though?

• IMPORTANT PRINCIPLE: you can’t solve a problem unless you know what the problem is…
  – When stating solutions, be clear about the problem that is solved

• Why CSC 402? (why software engineering? … why anything …
  – what is the problem that needs a solution?
  – how do we attempt to solve the problem?
  – what are the benefits in concrete terms?
  – what are the limitations of the approach?
The problem and the response...

• Software is typically
  – late
  – over budget
  – faulty
  – hence... the “software crisis”
    € go see the “Chaos Report” referenced on my website

• Software Engineering
  – software production should use established engineering principles
  – history: coined in 1967 and endorsed by a NATO conference in 1968
What type of software?

• Small single-developer projects can typically get by without Software Engineering
  – typically no deadlines, small budget (freeware), not safety-critical

• Software Engineering is required for
  – large projects (100,000 lines of code and up)
  – multiple subsystems
  – teams of developers (often geographically dispersed)
  – safety-critical systems (software that can kill people...)

CSC 402, Requirements Engineering
Software Engineering is still young

- Traditional engineering disciplines have been around for hundreds, if not thousands, of years
- Software Engineering still needs
  - standard definitions that make sense (check the IEEE definition of “requirement” - I might fail you for writing that!)
  - standard specification and design techniques
  - formal analysis tools
  - established processes
- Currently experimenting in
  - techniques, notations, metrics, processes, architecture, etc.
  - some success has been reported
    - € and occasionally overreported (See Watts Humphrey’s work?)
  - a foundation is being formed...
What is Engineering?

• Engineering is
  – sequence of well-defined, precisely-stated, sound steps, which follow method or apply technique based on some combination of
    € theoretical results derived from a formal model
    € empirical adjustments for un-modeled phenomenon
    € rules of thumb based on experience

• This definition is independent of purpose ...
  – “engineering” can be applied to many disciplines
    € however, does software have the formal models, rules of thumb...?

• Are software “engineers” employees or professionals?
  – are we independent in our employ?
    € do we have obligations to society?
      • go look at the Software Engineering Code of Ethics (ref on my website)
Software Engineers require ...

• a broad range of skills
  – Mathematics
  – Computer Science
  – Economics
  – Management
  – Psychology

• applied to all phases of software production
Software economics...

• Software Production = development + maintenance
  – maintenance accounts for approximately 67% of the overall costs during the lifecycle of a software product (Boehm)
    € faster development is not always a good thing
      • may result in software that is difficult to maintain
      • resulting in higher long-term costs
    € any of you familiar with Xtreme programming or JIT methods?
Lifecycle Costs (Schach data from Boehm)

- Maintenance: 67%
- Integration: 6%
- Module testing: 7%
- Module coding: 5%
- Design: 6%
- Planning: 6%
- Requirements: 4%
- Problem Def: 3%
- Requirements: 4%
- Planning: 2%
- Maintenance: 67%
What was that?

• Can you interpret the pie chart and explain it?
  – what *should* the chart look like?
    € what do we know about software projects in general?

• One researcher claims we’ll avoid maintenance costs by buying new software more frequently
  – we’ll avoid the “rare errors” in the short run
    € he’s in the safety-critical domain!

• What is “maintenance” anyway? Is this part of the problem we’re looking at?
  – was it a requirements failure or a change due to a new understanding of the problem…..
Maintenance Data

- All products undergo maintenance to account for change ...
- Three major types of maintenance
  - Perfective (60.5%)
    - Changes to improve the software product
      - an interesting figure!
        - why is this so high???
  - Adaptive (18 %)
    - Responding to changes in a product’s environment
  - Corrective (17.5 %)
    - Fixing bugs...

“Real world” is constantly changing
Maintenance is a necessity
Requirements and Design Aspects

• User needs and perceptions are difficult (impossible?) to assess
  – functionality isn’t enough
• Requirements specification is a contract with the customer
• Requirements must provide a definitive basis for testing
• Requirements is about the problem domain (Jackson)
• Design suggests a solution in the software domain

Requirements addresses the problem domain only
Design addresses the programming solution
Verification and Validation Aspects

- The longer a “fault” exists in software
  - the more costly it is to detect and correct
  - the less likely it is to be fixed correctly
    - e.g. fixing it “breaks” something else!
    - BUT, think about this one! See Beizer, “Software IS Different” QW 1996
- 60-70 % of all faults detected in large-scale software products are introduced in its specification and design
  - data regarding “requirements” defects shows LOTS of problems start there.
- Thus...faults should be found early (or prevented!)
  - requires specification and design V&V
  - validate first description and verify each phase with respect to previous
  - evaluate testability and develop test plans at each phase

Verification and validation must permeate the software lifecycle
Relative cost of fixing a fault (Boehm data)
Team Programming Aspects

• Reduced hardware costs afford hardware that can run large and complex software systems – products too complex for an individual to develop
• Most software is produced by a team of software engineers, not an individual
  – Team programming leads to interface problem between components and communications problems between members
  – Team programming requires good team organization to avoid excessive communication (a nontrivial problem)
  – Teams may be distributed geographically and temporally (even in this class)

Team programming introduces real communication overhead
Software Engineering Principles

• Deal with both *process and product* (big issues here!)
• Applicable throughout the lifecycle
• Need abstract descriptions of desirable properties
• Same principles as other engineering disciplines (witness Leveson)

€ is this true?
Rigor and Formality

• Rigor is a necessary complement to creativity
  – enhances understandability, improves reliability, facilitates assessment, controls cost
• Formality is the highest degree of rigor
• Engineering = sequence of well-defined, precisely-stated, sound steps, which follow method or apply technique based on some combination of
  – theoretical results derived from formal model
  – empirical adjustments for un-modeled phenomenon
  – rules of thumb based on experience
Separation of Concerns

- Enables mastering of inherent complexity
- Allows concentration on individual aspects
  - product features: functions, reliability, efficiency, environment, user interface, etc.
  - process features: development environment, team organization, scheduling, methods,
  - economics and management
- Concerns may be separated by
  - time (process sequence)
  - qualities (e.g., correctness vs. performance)
  - views to be analyzed separately (data vs. control)
  - components
- Leads to separation of responsibility
- Sometimes an intuitive exercise to separate concerns
Modularity and Decomposition

• Complex system divided into modules
• Modular decomposition allows separation of concerns in two phases
  
  ![Diagram showing bottom-up and top-down approaches with labels: aspects of modules in isolation and overall characteristics of integrated system]

• Modularity manages complexity, fosters reusability, and enhances understandability
  – compositibility vs. decomposibility
  – high cohesion and low coupling quality metrics
    – € for great discussion see Perrow, “Normal Accidents”
Abstraction

• Identify important aspects and ignore details
• Abstraction depends on the purpose or view
• Models are abstractions of reality
  – what does this really mean?
• Abstraction permeates software development
  – from requirements to code
  – from natural language descriptions to mathematical models
  – from products to process
• One specification but many realizations
Anticipation of Change

• Constant change is inevitable in large software systems
  – software repair & error elimination
  – evolution of the application (users get a new view via the app)
  – evolution of the social order (business and legal requirements)

• Identify likely changes and plan for change
  – software requirements usually not entirely understood
  – users and environments change
  – also affects management of software process

• Maintenance is process of error correction and modification to reflect changing requirements
  – regression testing with maintenance
  – configuration management of versions

• Is this one of the distinctions from other standard Engineering disciplines?
Generality

- Focus on discovering more general problem than the one at hand
  - fosters potential reuse
  - facilitates identification of OTS solution
- Trade-offs between initial costs vs. reuse savings
- General-purpose, OTS products are general trend in application domains
  - standard solutions to common problems
  - how far can this be taken?
Incrementality

- Step-wise process with successively closer approximations to desired goal
- Identify and “deliver” early subsets to gain early feedback
  - fosters controlled evolution
- Incremental concentration on required qualities
- Intermediate deliverables may be prototypes
- Requires careful configuration management and documentation
Sample Software Qualities

- Correctness
- Reliability
- Robustness
- Performance
- Usability
- Testability

*What the heck do these terms mean?*
- what are the relationships between these qualities?
- what about safety? Is this a property of software itself?
  - € Is it subsumed under “reliability”???
    - See Leveson, Safeware
Uniqueness of Software

• What are we dealing with?
  – The stuff doesn’t “wear out” (but does exhibit a bathtub curve …)
  – The stuff has no “tolerance” - it is binary
  – The stuff weighs nothing, and you can’t really “see” it.
  – It is very plastic, we can always “change” it in place
    € try that with your automobile!

• Why don’t other engineering principles apply?
  – For example, statistical reliability methods?
    € No mean value theorem applies
    € No accurate user profile or operational distribution
  – So, when we test, what do we find out about software?
    € Can’t tell for sure if our software is good or not.
Get Your Own Definitions

- Requirement
- Engineering
  - including the purpose for it!
- Process
  - See Osterweil’s “Software Processes are Software Too”
- Tools
- Methods
- Design
- Function
  - distinguish “feature”
Readings

• Wiegers, Part 1 (Ch 1 - 4 inclusive)
• Read Jackson on “Machines” and “Descriptions”
• Look over Yourdon, “Death March”
Written Homework

• Create your resume for this course today in lab:
  – experience, relevant classes (gpa?), other relevant facts, email
  € you’ll be “hired” on the basis of this resume. Make it 1 page please
  – management candidates: I will choose managers
  € we’ll need 5 or maybe 6 managers for as many teams
Journal Creation

• Begin your Journal in good quality loose-leaf notebook so that you can use dividers
  – Keep space (by divider or a separate journal) for your team notes, copies of assignments, documents, sketches, and other things relevant to the project.

• I recommend that you begin with working definitions, one per page, with room to refine as the project progresses:
  € Software Engineering
  € Engineering (find one that emphasizes the social aspects!)
  € Requirement
  € Design (to distinguish the two!)
  € Tools
    • analytical, software
  € Process
    – (go find Osterweil’s “Software Processes are Software Tool” article and look it over at some point.)
  € Abstraction
  € Function (versus “feature”)
Journal (cont’d)

- Constraint
- Attribute
- Preference
- Expectation
- Geek

• Note that the journal should be brought to each class and lab.
  - purpose - record your engineering experience
  - document your work and progress
  - record references for use later
  - prove to instructor that you’re not a slacker
  - “play with” ideas (even bad ones…)
• Most every document, note and idea for the project must appear in the journal
  - please organize it well
    - I need to be able to see how good it is in order to give you the grade you deserve!
Teams (we’ll form this or next class)

• Plan a social activity over the weekend
• Make a report, oral and summary in writing, for next week: Monday during lab
• Produce a document due on Monday in class:
  – Cover sheet for my folder containing your team documents and notes
    € what do I need to know?
      • your team structure, member names, contact information
      • team name on front, motto, other relevant information
      • done professionally, make it “useful” to me as a manager of teams