Lecture 2: SE Review

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 2000

Today's Lecture

- Review Software Engineering definitions
- Discuss the Nature of Software
  - Present Software Qualities
- Examine Software Engineering principles

Software Engineering

- Software
  - Computer programs and their related artifacts
    - e.g. requirements documents, design documents, test cases, specifications, protocol documents, UI guidelines, usability tests, ...
- Engineering
  - The application of scientific principles in the context of practical constraints

What is Engineering?

- Engineering is
  - a sequence of well-defined, precisely-stated, sound steps, which follow a method or apply a technique based on some combination of
    - theoretical results derived from a formal model
    - empirical adjustments for unmodeled phenomenon
    - rules of thumb based on experience
- This definition is independent of purpose...
  - i.e. engineering can be applied to many disciplines

Software Engineering (Daniel M. Berry)

- Software engineering is that form of engineering that applies:
  - a systematic, disciplined, quantifiable approach,
  - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines,
- to creating, developing, operating, and maintaining cost-effective, reliably correct, high-quality solutions to software problems.

Software Engineering

- the study of software process, requirements and design notations, implementation strategies, and testing techniques
- the production of quality software, delivered on-time, within budget, and satisfying its users' needs
- halfway between a discipline and an art form(!)

Sub-fields of SE

- Theory of Programs and Programming
- Formal & Heuristic Methods
- Configuration Management
- Testing
- Requirements & Design
- Metrics/Experimental SE
- Software Architecture, etc.

Software is Malleable

- Webster's definition
  - susceptible of being fashioned into a different form or shape
- Why is this bad?
  - Too easy to change software without going back to change requirements, design, etc.
    - This would never be done in other engineering disciplines!

Design vs. Manufacturing

- The creation of software is human-intensive
  - In other engineering disciplines, the majority of the costs associated with a product are located in manufacturing
  - In SE, software is more design intensive
    - Manufacturing is a trivial step (low relative cost)
    - Software maintenance is more costly
      - 67% of a software system's costs occur in this phase!

Software Qualities

- Correctness
- Reliability
- Robustness
- Performance
- User Friendliness
- Verifiability
- Maintainability
- Reusability
- Portability
- Understandability
- Interoperability
- Productivity
- Timeliness
- Visibility

Classifications of Qualities

- External vs. Internal
  - external - visible to a system's end-user
  - internal - visible only to a system's developers
  - internal qualities help developers achieve external qualities
  - boundary is blurry
- Product vs. Process
  - qualities of a process can impact the qualities of a product
  - Note: product can take on different meanings for different stakeholders
    - developers, marketing, customers

Correctness

- A system is functionally correct
  - if it behaves according to its functional requirements specifications
- Correctness asserts an equivalence between
  - the software and its specifications
- Assessment
  - Testing and Verification (program proofs)

Reliability

- Can a user depend on software?
- A system can be reliable but not correct
  - e.g. the fault is not serious in nature and the user can continue to get work done in its presence
- Contrast with other engineering disciplines
  - Engineering products are expected to be reliable; with software, users expect bugs!

Robustness

- How well does a system behave in situations not specified by its requirements?
  - Examples
    - incorrect input, hardware failure, loss of power
- Related to correctness
  - response specified
    - implementation must handle to be correct
  - response not specified => robustness involved

Software Qualities, continued

- Performance
  - In SE, performance is equated with efficiency
    - How quickly does it perform its operations?
    - Does it make efficient use of resources?
    - Is it scalable?
- User Friendliness
  - Better term: Human-Computer Interaction
    - Related: Human Factors, Cognitive Science

Software Qualities, continued

- Verifiability
  - Can properties of the system be verified?
  - Typically an internal quality
    - Security and safety critical domains are exceptions
- Maintainability
  - Corrective, Adaptive, and Perfective
  - Related: Repairability and Evolvability

Software Qualities, continued

- Reusability
  - software components, people, requirements
  - SE needs to make reuse standard practice
    - Why? It's standard practice in all engineering disciplines!
- Portability
  - The ability to run the same system in multiple contexts (typically hardware/OS combinations)

Software Qualities, continued

- Understandability
  - How well do developers understand a system they have produced?
    - supports evolvability and understandability
- Interoperability
  - Can a system coexist and cooperate with other systems?
  - Again, present in other engineering disciplines

Software Qualities, continued

- Productivity
  - The efficiency of a development process
    - An efficient process can produce a product faster and with higher quality
    - Can parts of it be automated?
    - Standard processes?
      - Software Life Cycles
      - Capability Maturity Model
        - » Measure everything!
        - » Use the results to improve the process the next time

Software Qualities, continued

- Visibility
  - A process is visible if all of its results and current status are documented clearly to internal and *external* viewers
- Timeliness
  - The ability to deliver a system on-time
    - requires careful scheduling, accurate estimates and visible milestones

Software Engineering Principles

- Rigor and Formality
- Separation of Concerns
- Modularity
- Abstraction
- Anticipation of Change
- Generality
- Incrementality

Rigor and Formality

- Webster definition for Rigor
  - strict precision
  - Is this at odds with creativity?
    - No, you can still be creative but apply rigorous standards in assessing the product of creativity
- The highest level of rigor is formality
  - Mathematically-based techniques
  - The trick is knowing when you need it!

Separation of Concerns

- Identify different aspects of a problem
  - so that they can each be addressed separately
  - the idea is to reduce complexity
- Separation by Time
  - Software life cycles
- Separation by Qualities
  - Correctness vs. Performance, for example

Modularity

- Systems can be divided into modules
  - Modules help address separation of concerns
    - bottom-up design: modules in isolation
    - top-down design: global module relationships
      - Cohesion and Coupling are major concerns
- Modularity is important in other engineering disciplines
  - factories produce products from components

Abstraction

- Identify the important aspect of some phenomenon and ignore the details
- Allows the user of an abstraction to be independent of the hidden details
  - This allows the details to change without a user knowing about it (or caring)
- Abstraction supports the design of layered systems or virtual machines

Anticipation of Change

- We know that software will change
  - Bug fixes, environmental changes, new features
- So how do we plan for it?
  - Modularization and Abstraction
  - Configuration Management Systems
- Need to anticipate personnel turnover

Generality

- Attempt to find general (broad) solutions to (software) problems
  - A general solution is more likely to be reusable
- Trade-off
  - The general solution may not be efficient
    - its hard to optimize something that must work across many different contexts

Incrementality

- Characterizes a process which proceeds in a stepwise fashion
  - The desired goal is reached by creating successively closer approximations to it
- Examples
  - Software life cycles
    - Especially those with prototypes and user feedback
  - "Don't write the whole program before you compile!"