Software Engineering: Principles

CSE 403 - Spring 1997

Outline

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

Why Principles?

- If we are to apply principles to software development and maintenance, we need to know what they are.
- And we need to know what their implications are on the development and maintenance of software.
- By applying principles, we are trying to increase certain qualities of software and, in turn, its development and maintenance process.

Rigor and Formality

- Software development is a creative activity.
- But we need to be rigorous, if we are to increase our confidence in the quality of the software.
  - reliable, cost control, etc.
- It may also enhance creativity.
  - Strive to produce a creative solution.
  - But analyze and evaluate rigorously.

Formalism: Bringing Rigor

- Use mathematical formalism to be rigorous.
  - Use formality to eliminate errors in reasoning and understanding.
  - What formalisms do we have in our arsenal and what are their benefits?
- One can still be rigorous in informal setting.
  - One needs to always be aware of the degree of rigor to be achieved.
  - Is being formal justified?
- How do rigor and formality contribute in reducing misunderstandings among the parties involved?
  - What are the challenges in bringing rigor and formality into communication?

Anticipation of Change

- Software always changes.
  - Only useful software goes through changes.
  - Prompted by needs external and internal to software.
    - Fix the bugs!
    - Make it better., e.g., run faster.
    - Customer needs and external environment do change.
- Software evolvability needs to be engineered in.
  - Other engineered products do not have this problem to the same degree as software does.
    - When designing a jet-liner, an engineer may anticipate stretching its fuselage and attendant increased load, or increase in engine thrust. But definitely not the number of engines or where they are mounted.
    - How about retrofitting a bank account balance system to give balances on your brokerage account?
    - How about fashioning a windowed UI for a text editor?

Anticipation of Change

- We need to organize the process so that changes are handled in a controlled manner.
  - Both in software and development and maintenance process.
- Gain leverage from domain experts.
  - Domain experts may shed some lights on what changes are reasonably expected in their application area.
  - One source of domain expertise: **customers!**

Separation of Concerns

- Deal with different aspects of a problem separately.
  - Aspects of a software product:
    - reliability, space and time efficiency, interactions with environment, UI, etc.
  - Aspects of software development process:
    - development environment, team organization and structure, scheduling, control procedures, etc.
- Some ways to separate concerns:
  - Time:
    - E.g., Mornings: Teaching, Afternoons: Research.
    - Avoid the overhead of switching activities in an ad hoc manner.
  - Quality:
    - E.g., efficiency vs. correctness.
    - Apply appropriate techniques.
  - Views: e.g., analysis of data flow vs. control flow.
  - Parts $\rightarrow$ modularity.

Separation of Concerns

- Sometimes, issues are so intertwined that they can not be separated.
  - Must account for the effects of decisions made for one issue have on others.
  - E.g., Atomic vs. non-atomic transaction and performance and correctness issue.
- Separation of concerns may lead to separation of responsibilities.
  - Utilization of different skill sets for better performance.

Modularity

- An application of the principle of separation of concerns to system architecture design.
  - Divide and Conquer.
  - One may deal with a module in isolation, without regards to other modules...
    - ... that is if they are well designed.
  - Then, deal with the system architecture design as a specification of how these modules interact.
  - Bottom-up vs. top-down design.
- Benefits:
  - Divides the problem into a comprehensible chunk.
  - Work assignments and independent development.
  - Protection against a propagation of changes... sometimes.
  - Others?

Modularity

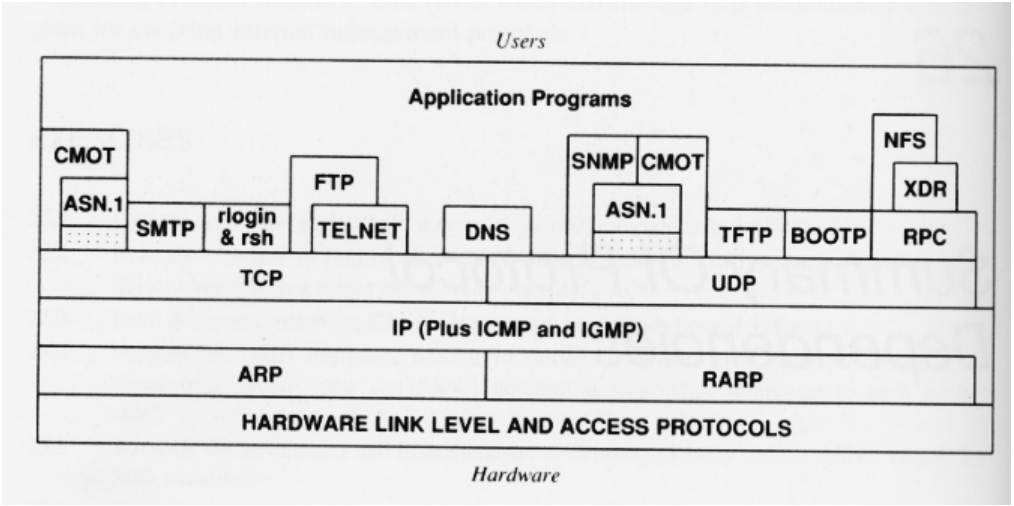
- Goals:
  - Decomposability
  - Composability
  - Understandability
- High cohesion and low coupling.
  - Cohesion - an internal property of a module.
    - Measure a degree of cooperation and interactions within a module.
  - Coupling - a degree of interdependence among the modules.
    - If two modules depend on each other heavily, they are highly coupled.

Abstraction

- Expose important aspects of a phenomenon or an entity and **hide the details**.
- What vs. How
  - A clock: Displays the current time. How it keeps the display current is hidden from its user.
  - A programming language: It presents an abstract machine capable of executing a program written in it.
- Benefits:
  - Helping comprehension of the system by hiding away the details not necessary for its role in a system.
  - Allows changes to hidden parts without affecting its client.
  - Others?
- How does abstraction affect coupling and cohesion?

TCP/IP Stack

- Abstraction in action.
- How did this architecture contribute to the success of TCP/IP?



Generality

- More general problem hidden behind a specific one on hand.
  - A solution to more general problem may be reusable.
  - Somebody else may have solved it.
    - Database servers.
    - Web browsers.
    - MFC.
  - Watch out for the "not invented here" syndrome.
- But is it economical?
  - execution speed, memory, amount of effort required, etc.

Incrementality

- Big bang vs. evolutionary approach.
  - Big bang: release one final product.
  - Evolutionary:
    - Deliver useful subset of a system.
    - Get **early** feedback from the customer.
    - May need a correction on the direction.
    - Expand on the subset.
- Prototyping
  - Rapid prototyping CASE tools.
  - Does not have to be a high-tech.
    - Story boarding.
    - Concreteness and rigor.

Models

- A model abstracts a certain aspects of phenomenon.
  - E.g., work-flow model for a loan application.
  - Allows reasoning about and communications on the modeled aspects.
  - Intellectual leverage is gained by focusing on the modeled aspects.
  - Other benefits?
- Often, multiple models are needed.