Software Engineering Principles
SE Principles

• Principles are statements describing desirable properties of the product and process.
SE Principles

- Principles are statements describing desirable properties of the product and process.
- Focus on both process and product are needed to deliver software systems.
  - Cannot focus on just one.
  - We control process in an effort to control the quality of the product.
SE Principles

- Principles are statements describing desirable properties of the product and process.
- Focus on both process and product are needed to deliver software systems.
  - Cannot focus on just one.
  - We control process in an effort to control the quality of the product.
- Control of process will not ensure a quality product.
Question

• When you build a software system, what things do you keep in mind as you develop the system?
Principles

- Separation of Concerns
- Complexity
- Abstraction
- Modularity
- Information Hiding
- Anticipation of Change
- Rigor and Formality
- Generality
Separation of concerns

• Deal with different individual aspects of a problem, concentrating on each separately.
• If possible, separate problem into unrelated parts.
  – e.g. separate the user interface from the data access code
• Not always possible
  – e.g. design decisions such as memory size affect things like error recovery policies
How to Separate

• Separate the concerns:
  – Isolate the issues
  – Deal with each issue separately
How to Separate

• Separate the concerns:
  – Isolate the issues
  – Deal with each issue separately

• How to isolate:
  – Time: (eg software lifecycles)
  – Qualities (eg focus on correctness. Focus on performance)
  – Views of software (eg data flow, flow of control)
  – Parts of system (see modularity)
Problem:

• How can you assure the entire system is good if you only look at pieces?
  – May miss global optimizations if we only look at pieces.

• Design at a high level, then decompose
Complexity

• Loosely, the amount of time or resources needed to solve a problem
• In our context, the software attributes that affect the effort needed to implement or change software
• (Not the same as complexity studied in CS3)
Complexity

• Two attributes are size and structure
• Structure includes things like
  – Number of operators
  – Level of nesting
  – Interactions between components
  – Structure of data representation
Complexity

- General rule: the more the software needs to do, the more complex it becomes
Abstraction

• Identify the important aspects of some phenomenon and ignore the details
• It is our only tool for mastering complexity.
  – variables as abstraction of memory
  – Memory as an abstraction of circuits
  – Circuits as an abstraction of electron probability distributions
  – electron probability distributions as abstractions of subatomic particles
  – comments should provide abstraction of program code
Traversing Levels of Abstraction

• As a developer, you must be able to traverse levels of abstraction easily.
• Example: Customer calls on phone and says, “the accounts payable report is broken.”
• You must be able to start at this level of abstraction, then work down succeeding levels to get to the uninitialized variable in the PROCA routine.
Modularity

• Break system into modules
• Two phases
  – Deal with details of module in isolation
  – Deal with overall characteristics of all modules and their relationships
• Bottom up: (order listed)
  – Good reuse and composeability
• Top down: (inverse order)
  – good decomposition and modularity, independent parts
Desirable characteristics

• High cohesion, low coupling.
• Cohesion:
• Coupling:
Desirable characteristics

• High cohesion, low coupling.
• Cohesion: clustering of related parts.
  – All parts in a module work towards a common goal.
• Coupling: interrelation between modules.
  – We'd like to see code that is grouped into modules with minimal interaction between modules.
Information Hiding

• Hide at least some design decisions inside components
• Other components cannot depend on these design decisions
• This leads to lower coupling
• Example: hide a data structure inside an object: no other object need know if it’s a linked list, array, or hash table
Anticipation of change

• Useful software changes.
  – Students don’t get much experience in college

• Ability to evolve is not free.
  – Effort is required to identify where changes might occur
  – Effort is required to design so that changes can be implemented easily (low coupling).

• Change affects process also:
  – Employee turnover
  – Process improvement
Rigor and Formality.

• Rigor: Strict precision; Exactness.
• Formality is the highest degree of rigor.
  – One can be rigorous without being formal. Example: an intro calculus textbook is usually rigorous, but not formal. The proofs are just sketches. Hardly ever are the proofs done in a formal mathematical logic.
1. Rigor and Formality.

- Rigor: Strict precision; Exactness.
- Formality is the highest degree of rigor.
  - One can be rigorous without being formal. Example: an intro calculus textbook is usually rigorous, but not formal. The proofs are just sketches. Hardly ever are the proofs done in a formal mathematical logic.

- We can specify rigorously in natural language.
  - not amenable to tools.
- We can specify formally in a logical language.
  - may be amenable to tools.
Creativity vs Rigor

• Software development is creative.
  – Creative processes tend to be neither precise nor accurate.

• Rigor is necessary to produce reliable products.

• Rigor enables creativity in engineering.

• Rigor varies in degree: use the appropriate amount.
  – Eg doghouse vs skyscraper
Rigor vs. Formality

• Typically, only the source and object codes are formal. (formal language, formal semantics)

• We may need to resort to formality at times.
  – Correctness proofs.

• We may choose to use formality.
  – Formal specifications (Z, Petri nets)

• But in all cases, we want rigor.
Generality

• When asked to solve a problem, look for the more general problem. The more general problem might be as easy to solve. The solution is then a candidate for reuse.

• Big tradeoffs here.
  – Text books say be as general as possible.
  – Practice says be more specific.
  – In practice, the more general, the harder, slower, more complex. Not always the case.
  – The more general, the more reusable.
Groups

• For each of the six principles
  – Come up with an example in some software you know about.
Examples

• Spreadsheets:
• Data bases
• Unix tools (awk, sed, sort, . . .)
• instead of a specific solution, build a general one that can be tailored.