ECE 321 Software Requirements Engineering

Lecture 2A: Software engineering principles

Outline

- **Rigor and formality**
- **Separation of concerns**
- **Modularity**
- **Abstraction**
- **Anticipation of change**
- **Generality**
- **Incrementality**
- **Information hiding**

Rigor and formality

- Software engineering is a **creative design activity**
- But... it must be practiced **systematically**

Rigor and formality

- **Rigor**
  - A necessary complement to creativity that increases one’s confidence in one’s developments
- **Formality**
  - Rigor at the highest degree
  - Driven and evaluated by mathematical laws

How much rigor is enough?

- An engineer must be able to identify and understand the level of rigor and formality that is necessary
- Depends on
  - the conceptual difficulty
  - criticality

Examples of rigor and formality

- Formal/mathematical analysis of correctness
- Systematic (rigorous) test data derivation
- Rigorous documentation of development steps helps project management

Separation of concerns

- Divide the different aspects of a problem
- Allows you to concentrate on each one individually

How can we separate concerns?

- Separation in **time**
- Separation in terms of **quality attributes** that should be treated separately
- Different **views** of the software
- Different **parts** of the same system

Examples of separation of concerns

- Keep product requirements separate
  - Functionality
  - Performance
  - Usability
- Go through phases one after the other (e.g., waterfall model – separation in time)

Software modularity

- A complex system can be divided into simpler pieces called **modules**
- A system that is composed of modules is called **modular**
- Focus on one module and ignore details of the others while doing so

Top-down modular design

- Decomposing a complex system into simpler pieces
- “Divide and conquer”

Bottom-up modular design

- Composing a complex system from existing modules
- COTS-based development
    - Commercial-of-the-shelf

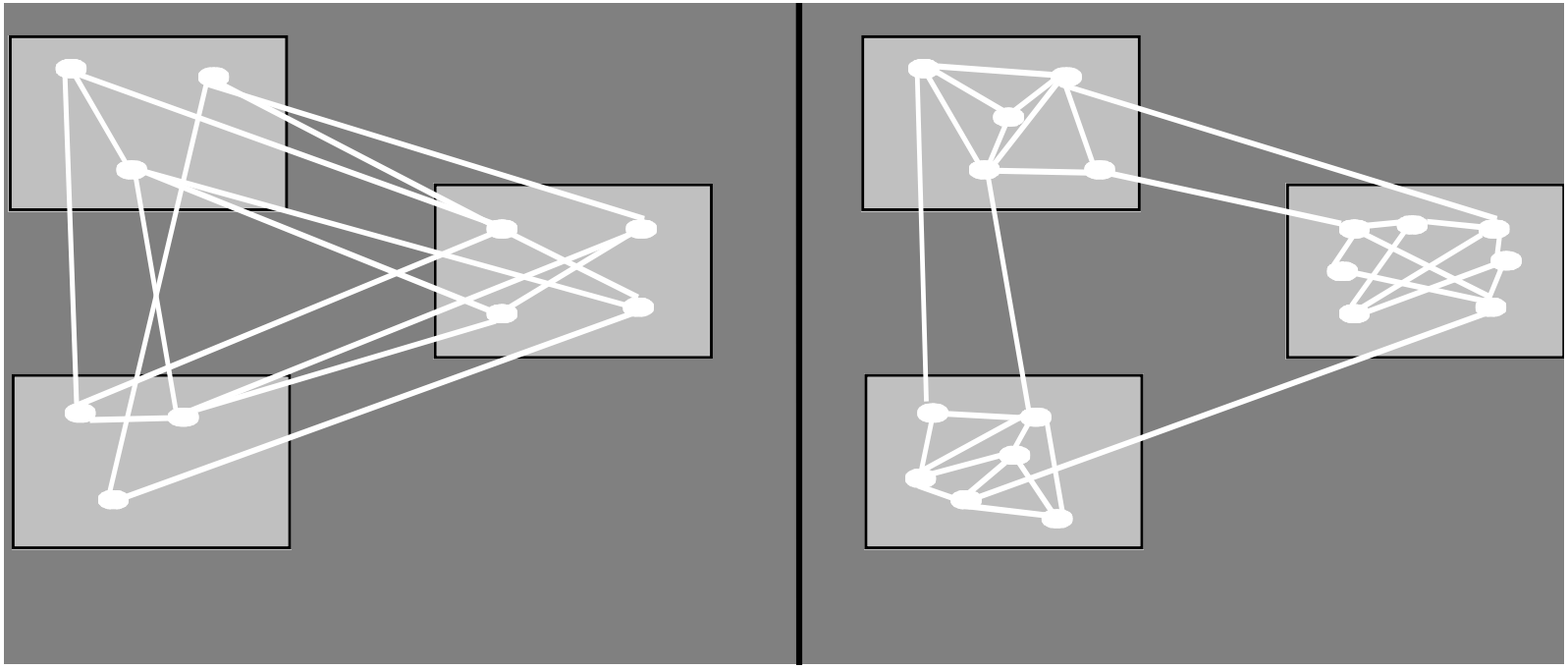
Benefits of modularity

- Understanding the system in term of its pieces
  - Necessary when applying modifications
- A system can be modified by changing only a small number of its modules
- Allows working in different teams

Cohesion and coupling

- Each module should be highly cohesive
  - Module understandable as a meaningful unit
  - Components of a module are closely related to one another
- Modules should exhibit low coupling
  - Low interactions with each other
  - Understandable separately

A visual representation of coupling



High coupling

Low coupling

Software abstraction

- Fundamental technique for understanding and analyzing complex problems
- Identifying the important aspects of a phenomenon and ignoring its details

Examples of abstraction

- User interface of a watch (its buttons)
- Racing car vs. regular car
- Cost estimation – only the key factors are taking into account

Anticipation of change

- Software will change :)
- Make sure to anticipate that it will
- Should be integrated in the requirements or in the design

How to anticipate for change?

- Use appropriate tools to manage versions
  - Version control systems: Git or SVN
  - Not only for source code... also for documentation, requirements, etc.

Generality

- Some problems are instances of a more general problem
- Make sure to create a solution that is reusable
- Sometimes the general problem is easier to solve than the special case (not always though)

Incrementality

- Process should proceed in a stepwise fashion (increments)
- Deliver subsets of a system early
- Deal with functionality first, then turn to performance

Information hiding

- One of the most important concepts you will learn about
- **Secrets** are the key to designing systems and modules

What should we keep a secret?

- Decide for each module which features should be known outside the system/module and which should be a secret

Benefits of information hiding

- Ease of modification/debugging
- Hiding details of complex logic/data
- Isolation of things that are likely to change
- Increase of reliability

When is information hiding a good idea?

ALWAYS

Summary of software engineering principles

- The discussed principles apply to all software development activities
- Always keep them in mind
- Information hiding is the key concept to almost everything in software