

CS30500:

Introduction to Software Engineering

Lecture #14

Prof. In-Young Ko

School of Computing

Last Class

- Requirements Models
 - Scenario-based modeling
 - Class-based modeling
 - Behavioral modeling
- Domain modeling
- Traceability

Today's Plan

- Design Concepts
- Modularization
 - Information hiding
 - Cohesion
 - Coupling

DESIGN CONCEPTS

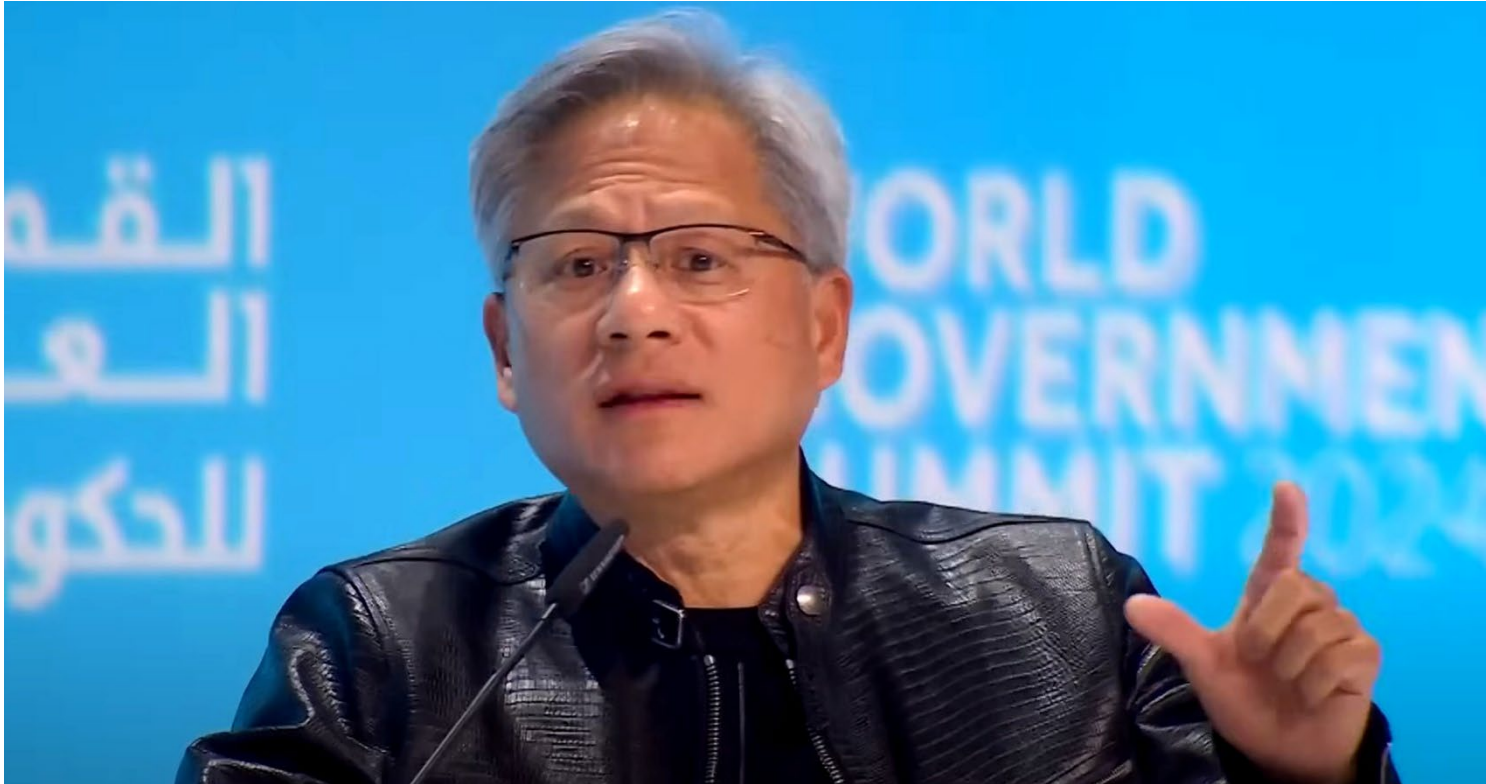
“Computer Programming is a Dying Art”

([Source: Newsweek, by Kevin Maney, 2014](#))

- “There is definitely a need for people to learn kind of a computer science way of thinking about problems, but not necessarily the language du jour,” says Erik Brynjolfsson, a professor at the MIT Sloan School of Management
- Irving Wladawsky-Berger, formerly of IBM and now at New York University, “We should definitely teach **design**. This is not coding, or even programming. It requires the ability to think about the problem, organize the approach, know how to use design tools.”

“Don't Learn to Code”

Nvidia's founder **Jensen Huang**
@World Government Summit 2024 in Dubai



<https://www.youtube.com/watch?v=yj73GIEKmLI>

Coding with AI

Add tailwindcss to this project

I'll help you add Tailwind CSS to your project. First, let me check if this is a Vite project and examine the current configuration

▼ Listed directory Current directory 22 results

Read file package.json

I see this is a Next.js project. Let's install Tailwind CSS and its dependencies, then configure it. I'll first check the current package.json to ensure we don't have any conflicts.

Finds Context

Using custom retrieval models, Cursor can understand a codebase. This reduces the need to manually add context.

Now I'll install Tailwind CSS and its dependencies. I'll run the necessary commands to set it up.

```
$ pnpm add _D tailwindcss postcss autoprefixer
```

Skip Run Command ⌘↵

Waiting for approval.

Runs Commands

Cursor can automatically write and run terminal commands. By default, you'll be asked to confirm all commands.

⚠ Property 'testCppConfig' does not exist on type
⚠ Property 'testCppConfig' does not exist on type

I see that there are some linter errors because the type

> Searched codebase "CppConfigResponse type definition"

Based on the search results I can see that the CppConfig doesn't have a testCppConfig field. Let me modify the

TS ComposerViewsService.ts +12 -4

```
printCppConfig(): void {  
  const cppConfig = this.reactiveStorageService.get('cppConfig')  
  if (cppConfig?.testCppConfig !== undefined) {  
    console.log('Test CPP Config', cppConfig.testCppConfig)  
  }  
}
```

Loops on Errors

Cursor can detect lint errors automatically and apply fixes, reducing the need for manual debugging.

CHAT

COMPOSER

We should show a strikethrough in the Tab status element when the user has run out of free credit

I'll help you implement the strikethrough styling for the Tab status element when users have run out of free credits. Let me first search for relevant code.

> Searched codebase "Tab status element credits display" • 25 results

Let me search for more specific code related to the Tab status element and credits.

> Searched codebase "Tab status element credits remaining display" • 25 results

Let me search for more specific code related to the credits display.

> Searched codebase "Tab status element credits remaining text" • 25 results

Let me search for one more piece of code related to the credits display.

Agent

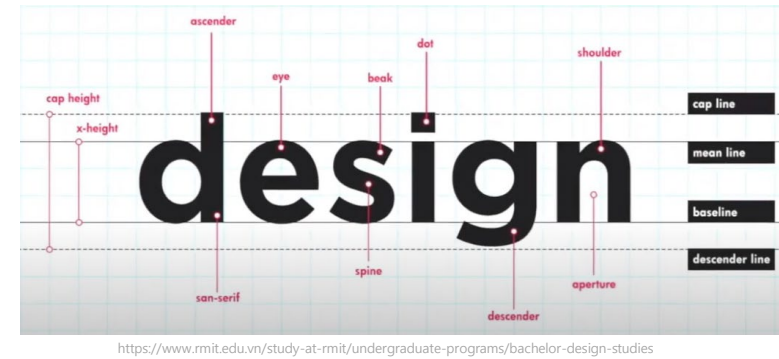
Cursor's agent mode can complete tasks end to end. It does this quickly, while keeping programmers in the loop. Try it out by selecting 'agent' in Composer.

<https://www.cursor.com/features>

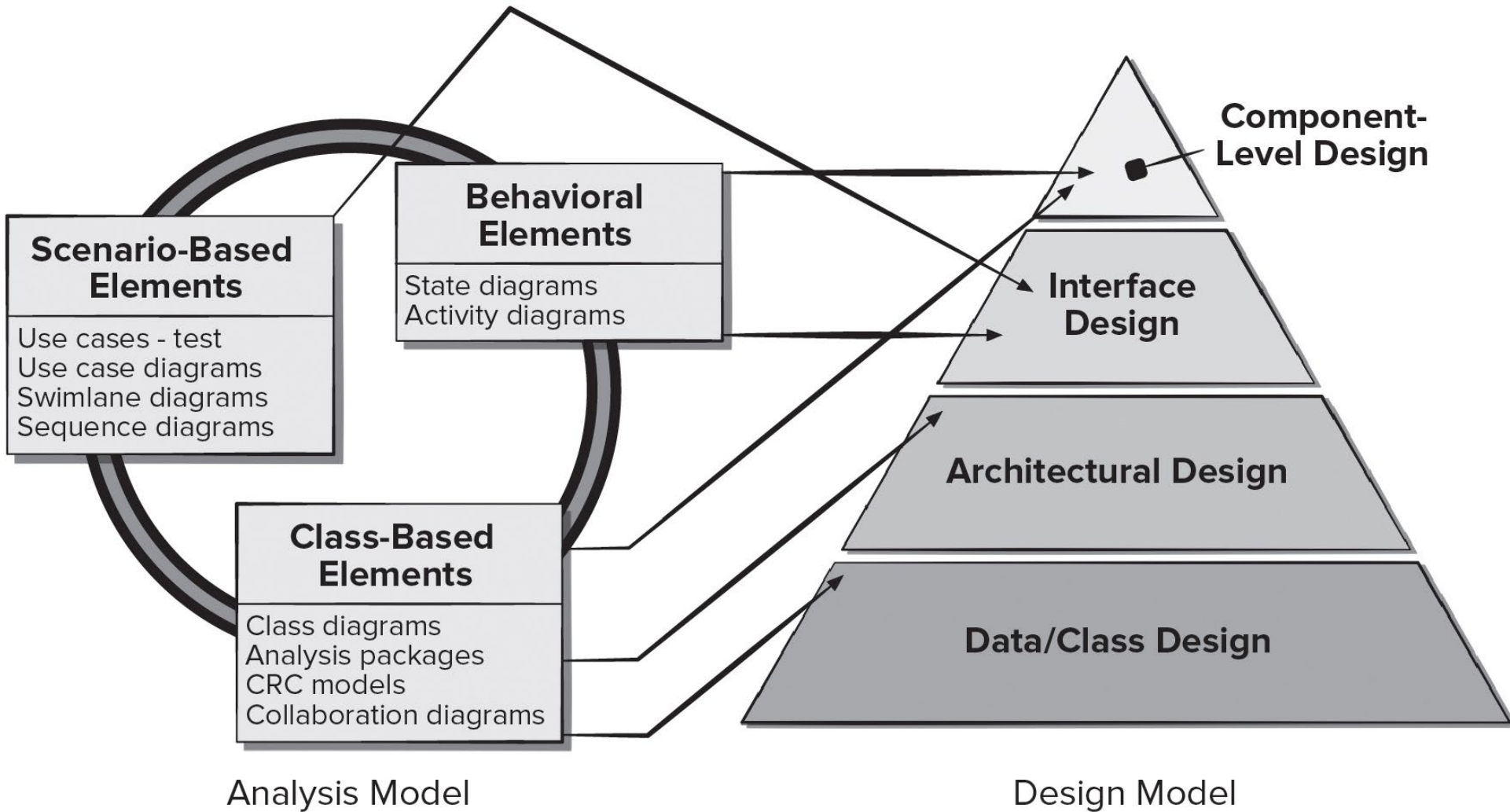
https://www.youtube.com/watch?v=LR04bU_yV5k

What is software design?

- The last SE action within the modeling activity and sets the stage for construction (code generation and testing)
- Transforms requirements into the form that can help developers to build a software system effectively
 - **Data/Class** design – transforms analysis classes into implementation classes and data structures
 - **Architectural** design – defines relationships among the major software structural elements
 - **Interface** design – defines how software elements, hardware elements, and end-users communicate
 - **Component-level** design – transforms structural elements into procedural descriptions of software components



Mapping Requirements Model to Design Model



Why is design important?

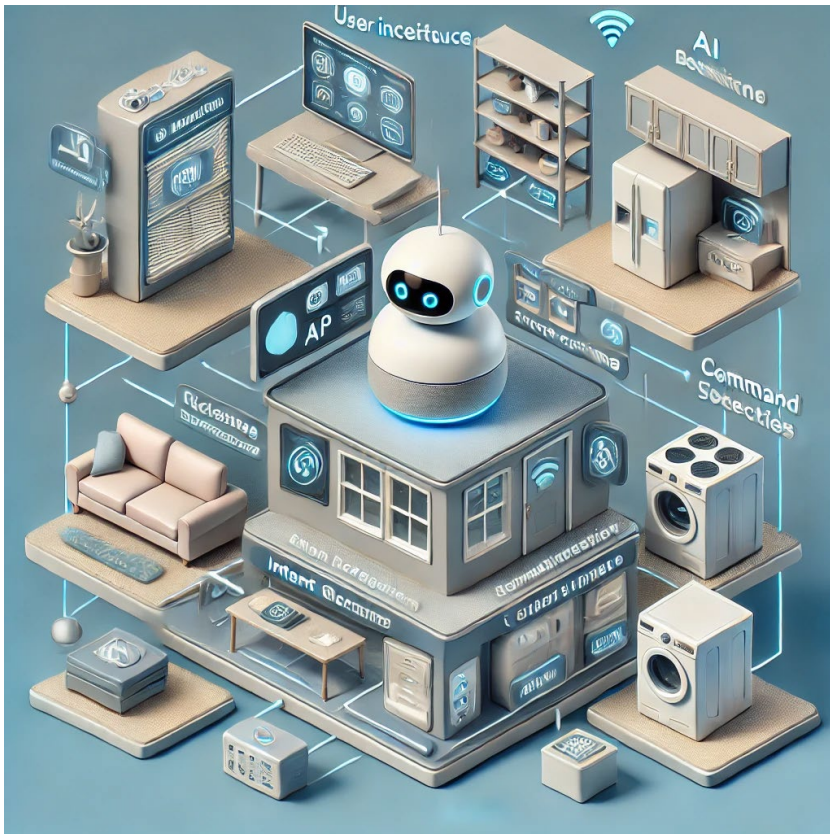
- The first step to propose a solution for the problem under development
- What is importance of software design?
 - Design is the place where _____ is fostered.
 - Design provides us with representations of software that can be assessed for _____ .

Design and Quality

- The design must implement all of the **explicit requirements** contained in the analysis model, and it must accommodate all of the **implicit requirements** desired by the customer
- The design should be a readable, understandable guide for those who generate code and for those who test and subsequently support the software
- The design should provide **a complete picture** of the software, addressing the data, functional, and behavioral domains **from an implementation perspective**

As a Software Engineer...

- Your task is to design and develop a smart home system where a butler robot understands the user's needs and seamlessly controls and coordinates home appliances.



[Generated by ChatGPT 4o]



Architectural design of the smart home system featuring the AI butler robot

Design Concepts (1/2)

- **Abstraction**

- Data – named collection of data describing data object
- Procedural – named sequence of instructions with specific and limited function

- **Architecture**

- Overall structure or organization of software components
- Ways components interact
- Structure of data used by components

- **Design Patterns**

- Describe a design structure that solves a well-defined design problem within a specific context

- **Separation of Concerns**

- Any complex problem can be more easily handled if it is subdivided into pieces

- **Modularity**

- Compartmentalization of data and function

Design Concepts (2/2)

- **Information Hiding**

- Controlled interfaces which define and enforces access to component procedural detail and any local data structure

- **Functional Independence**

- Single-minded (*high cohesion*) components with aversion to excessive interaction with other components (*low coupling*)

- **Stepwise Refinement**

- Incremental elaboration of detail for all abstractions

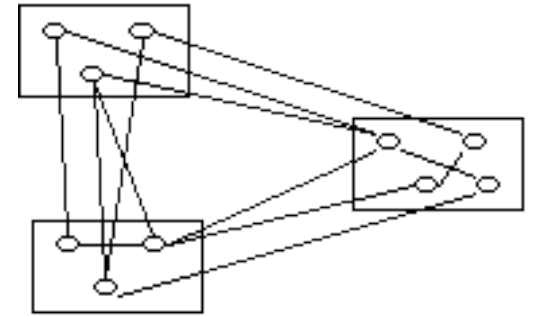
- **Refactoring**

- A reorganization technique that simplifies the design without changing functionality

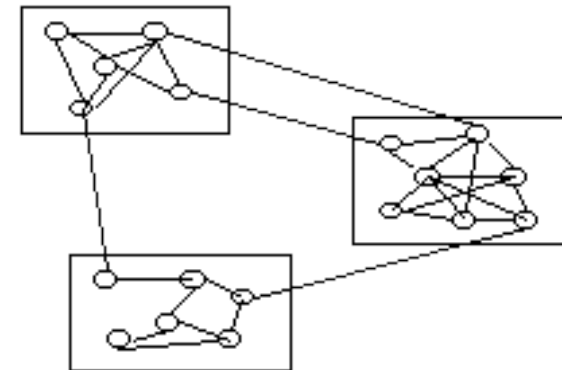
- **Design Classes**

- Provide design detail that will enable analysis classes to be implemented

Low Cohesion
High Coupling



High Cohesion
Low Coupling



[PrMa20]

MODULARIZATION

Modularization

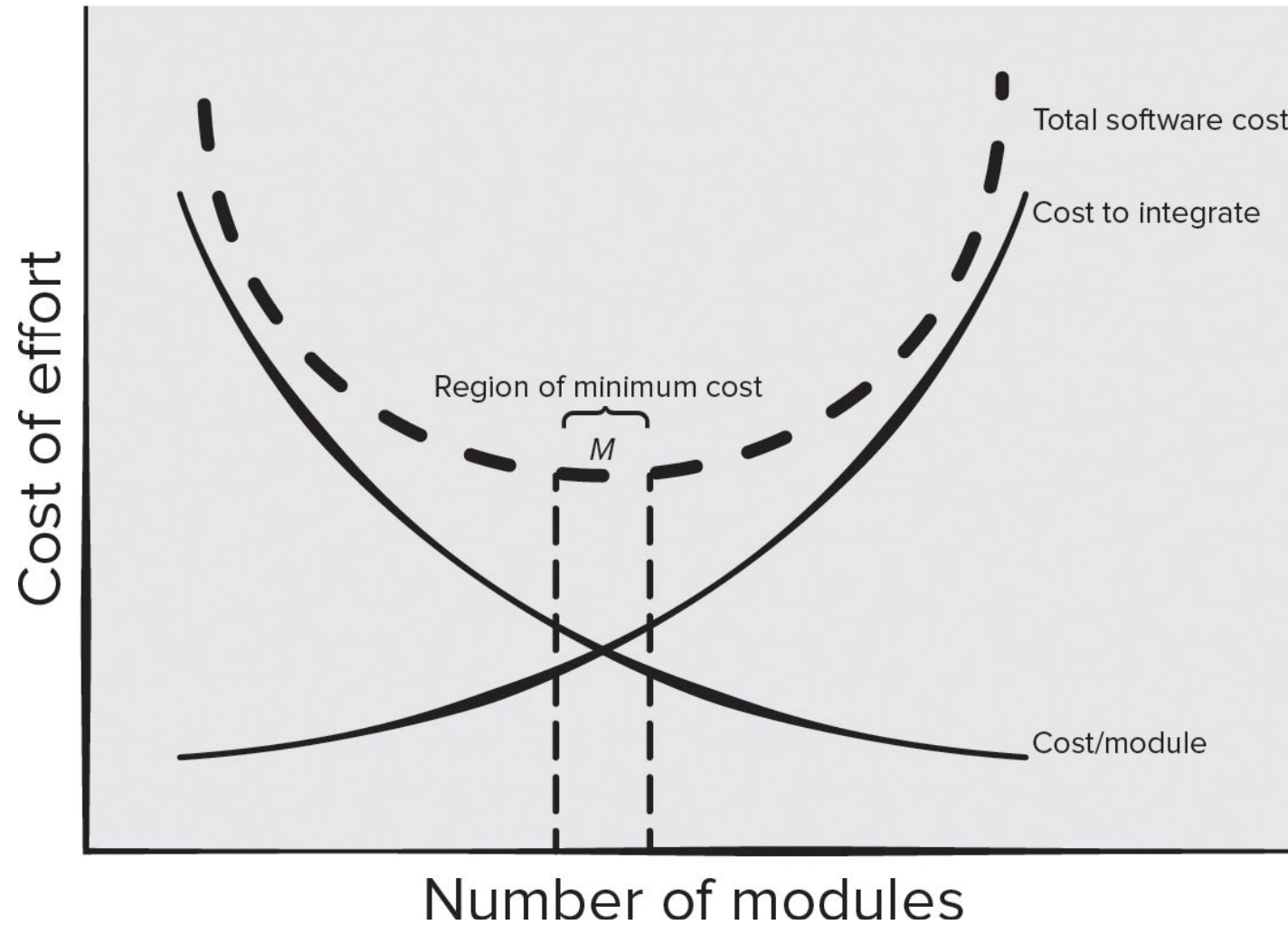
- **Modularization** is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently
- **Modules** may work as basic constructs for the entire software
- **Modules** can be executed and/or compiled separately and independently



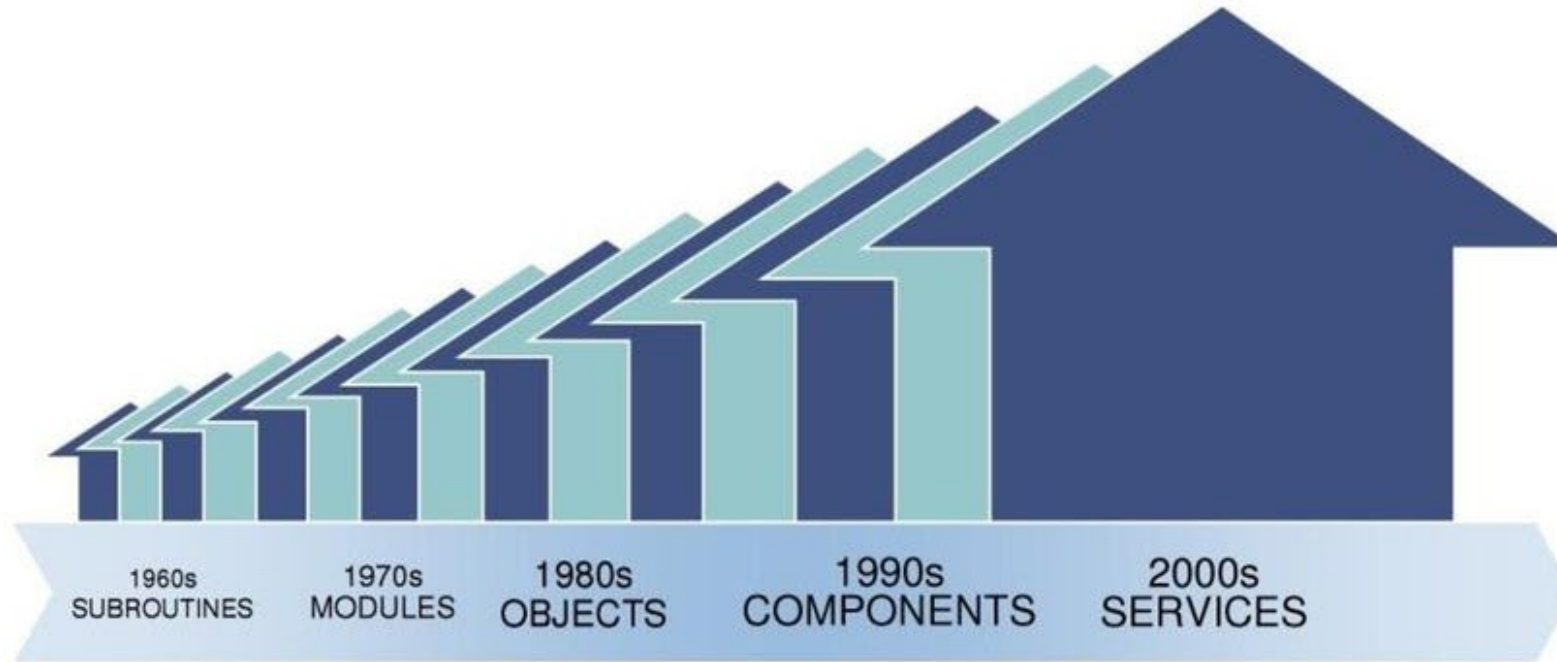
<https://www.linkedin.com/pulse/how-innovations-modularization-driving-construction-tannis/>

https://www.tutorialspoint.com/software_engineering/software_design_basics.htm
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Modularity and Software Cost



Software Reuse History



Software Engineering Institute

CarnegieMellon

Practical Guide to Modular Design

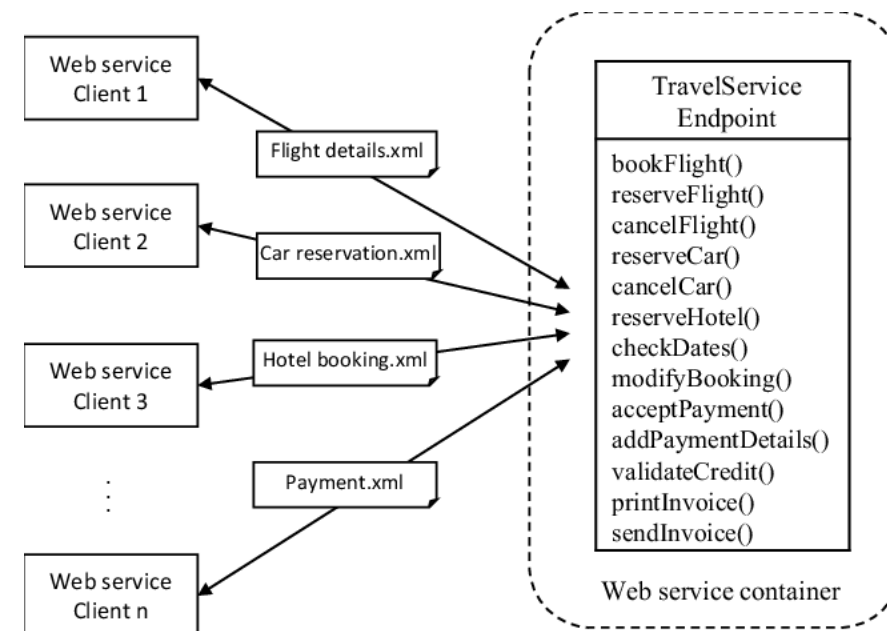
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

1. **Purpose**: should have a single responsibility
2. **Interface**: easy to use, understand, and ensure correctness
3. **Information hiding**: expose as little as possible
4. **Dependency**: minimize dependencies between modules

Avoid followings:

- Circular dependencies
- Not testing in isolation
- Too many small modules
- Having modules changing together frequently
- Unstable interface
- Unhealthy inheritance hierarchy
- God module (class)
- Duplicated functionality

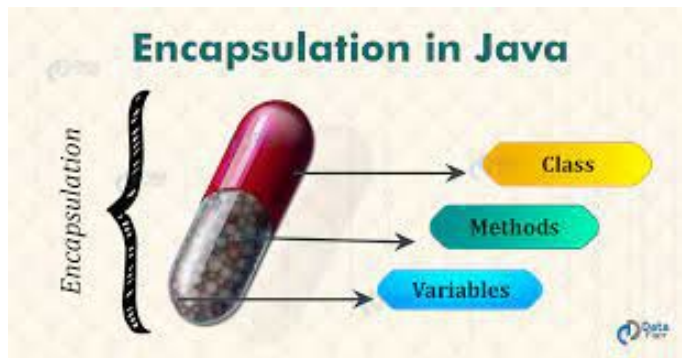
A God Module



https://www.researchgate.net/figure/An-example-of-god-object-Web-service_fig1_307572331

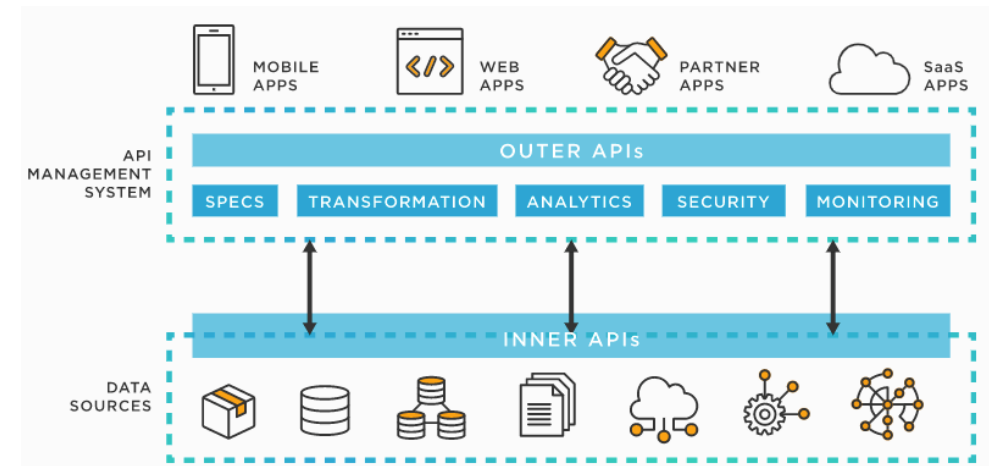
Encapsulation vs. Information Hiding

- **Encapsulation** means drawing a boundary around something
- It means being able to talk about the inside and the outside of it
- Object-oriented languages provide a way to do this
- **Encapsulation is a programming language feature**



<https://data-flair.training/blogs/encapsulation-in-java/>

- **Information** hiding is the idea that a design decision should be hidden from the rest of the system to prevent unintended coupling
- **Information hiding is a design principle**



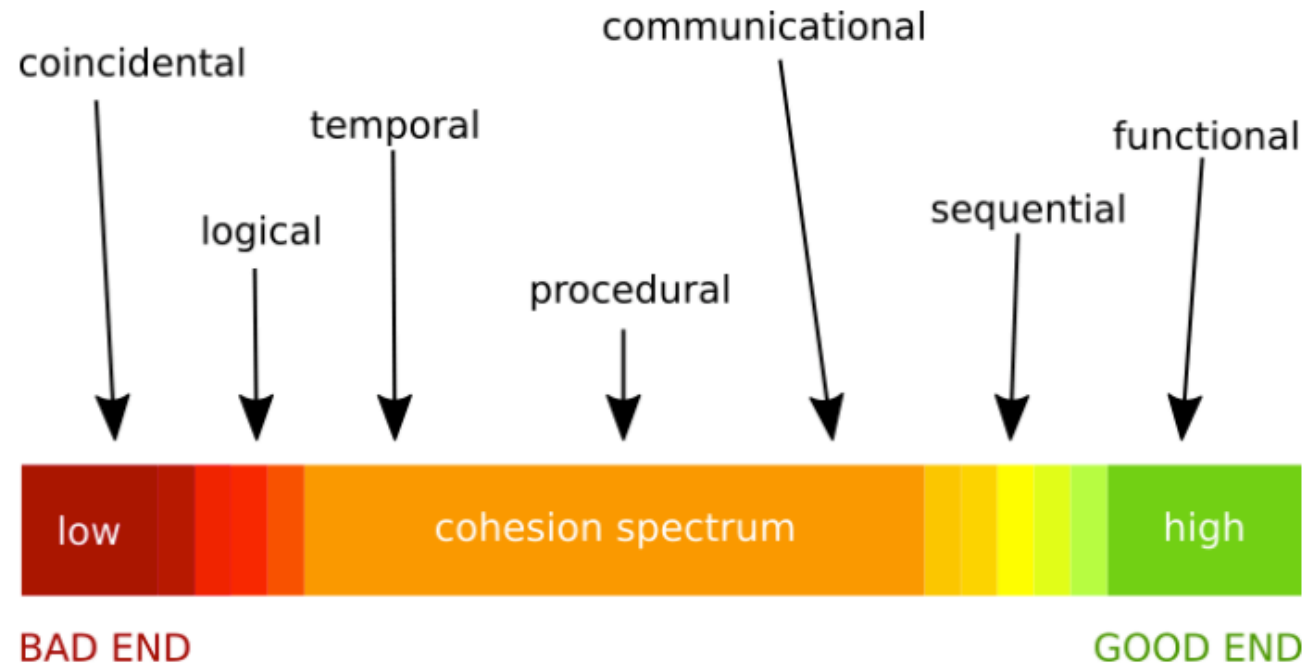
<https://www.tibco.com/reference-center/what-is-an-api>

(source: C2 wiki)

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Cohesion

- The measure of strength of the association of elements within a module
- Highly cohesive modules are desired
- Levels of cohesion:



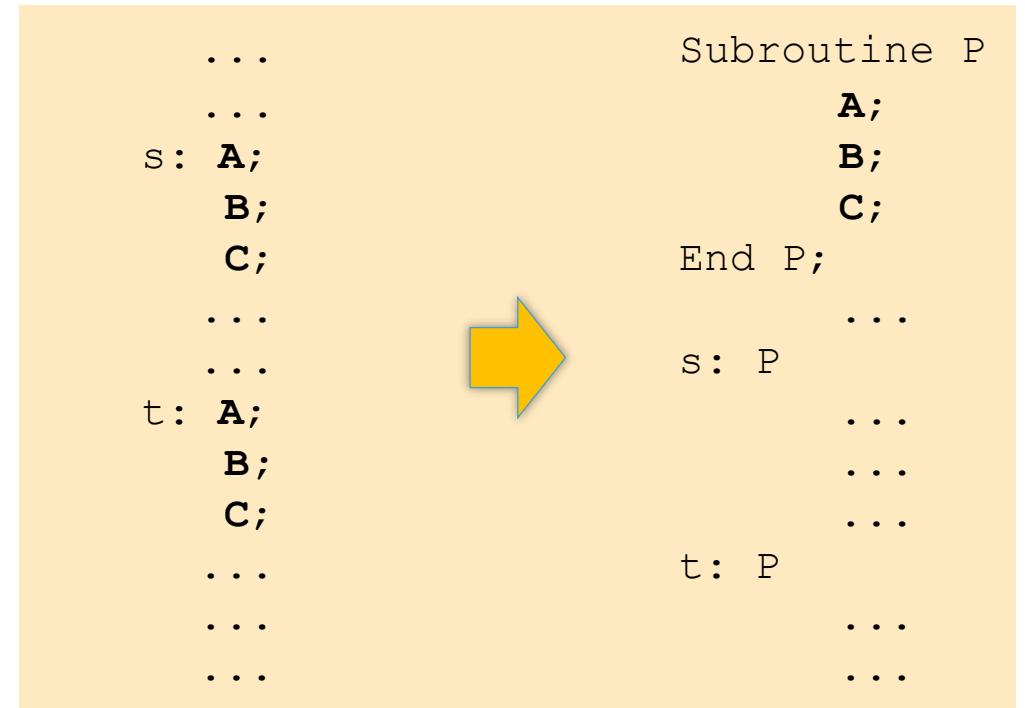
<https://rajasagar.medium.com/do-you-care-about-cohesion-in-javascript-functions-8b1dc26e1ca0>

David Stotts, Intramodule Cohesion, COMP145, University of North Carolina (<https://www.cs.unc.edu/~stotts/COMP145/cohesion.html>)

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Coincidental Cohesion

- The elements in a module are not related but simply bundled together
- Where does this type of cohesion come from?
 - Sloppy maintenance
 - Subroutinization



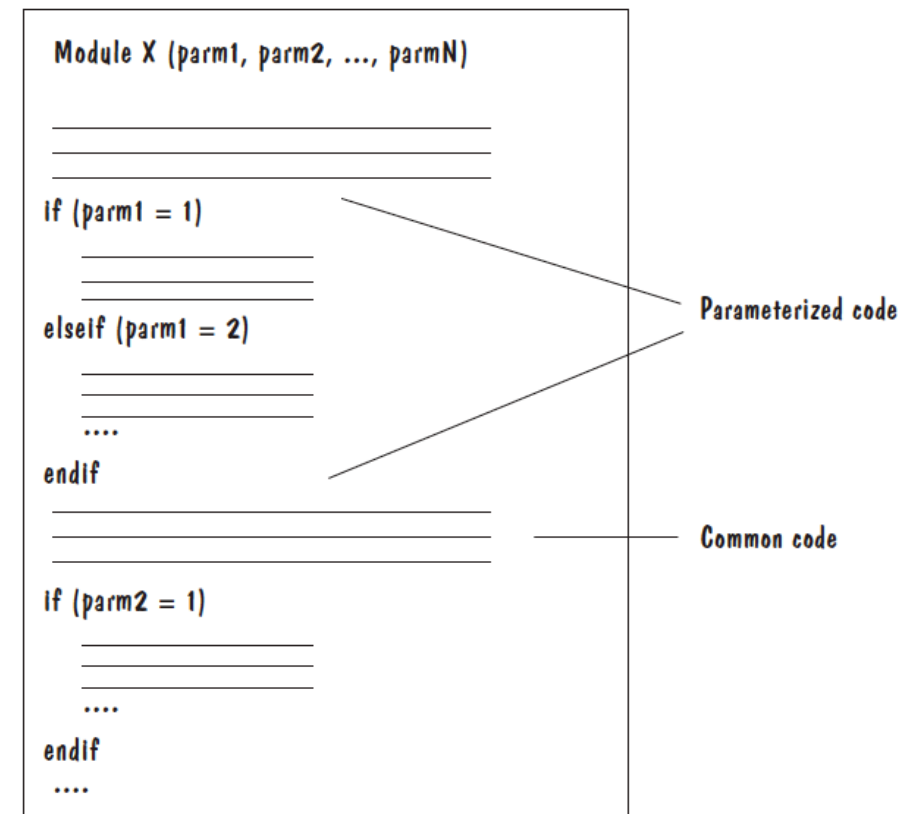
Logical & Temporal Cohesion

- **Logical Cohesion**

- The elements in a module perform similar or related functions that fall into the same logical class
- Examples:
 - Generalized transaction processing
 - General error handling

- **Temporal Cohesion**

- All the elements in the module are activated at a single time
- Examples:
 - Initialization
 - Termination

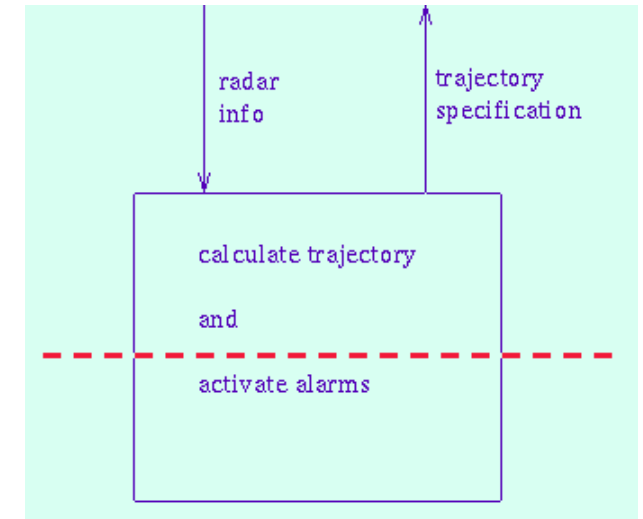


Pfleeger, Shari Lawrence and Lawrence Pfleeger.
"Software Engineering: Theory and Practice"

Procedural & Communicational Cohesion

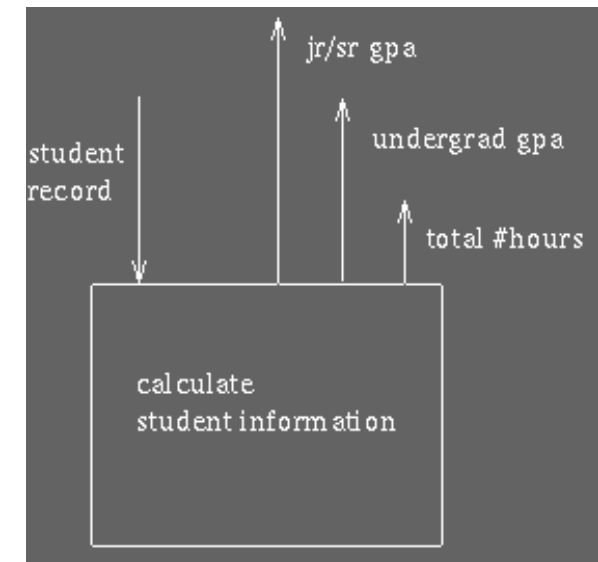
- **Procedural Cohesion**

- The elements in a module are involved in different activities, but are invoked as a sequence
- Examples:
 - Calculate trajectory and activate alarms
 - Read Part_Number from DB and Update Repair_Record on Maintenance_File



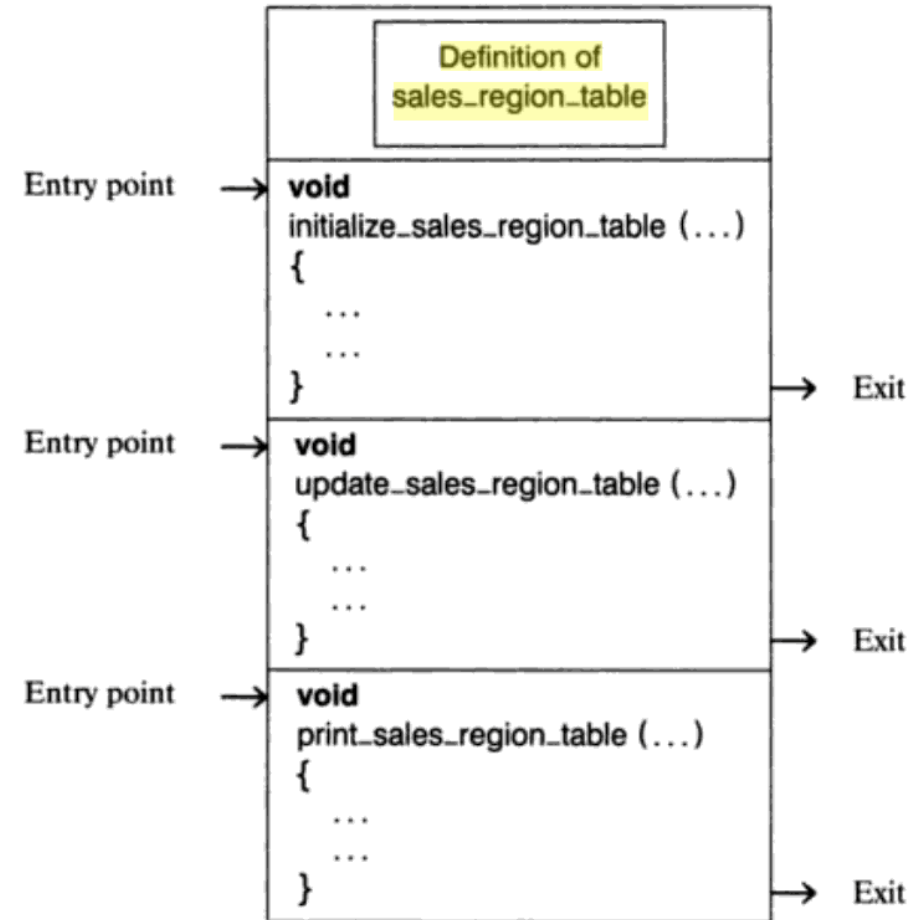
- **Communicational Cohesion**

- All of the elements in a module operates on the same input or produce the same output
- Examples:
 - Calculate student information
 - Update record in DB and write it to audit file



Informational Cohesion

- A module performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data structure
- Essentially an implementation of an abstract data type
- Example: Class in OO



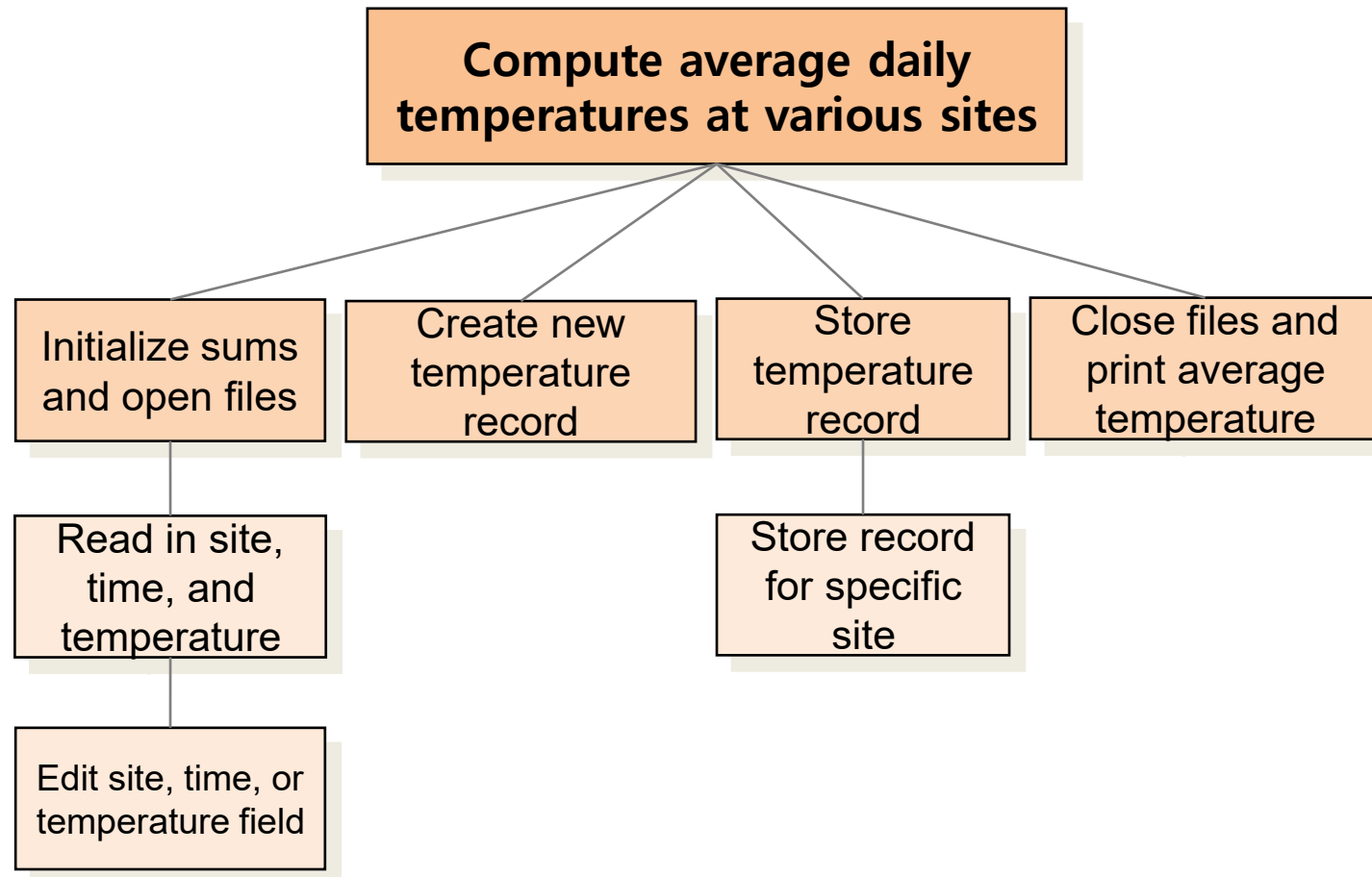
Functional Cohesion

- A module performs exactly one action or achieves a single goal
- Improve reusability
- Easy to maintain
- Easy to extend
- Examples:
 - Get temperature of furnace
 - Compute orbital of electron
 - Calculate sales commission

Cohesion Exercise

Identify the type of cohesion in the following statements:

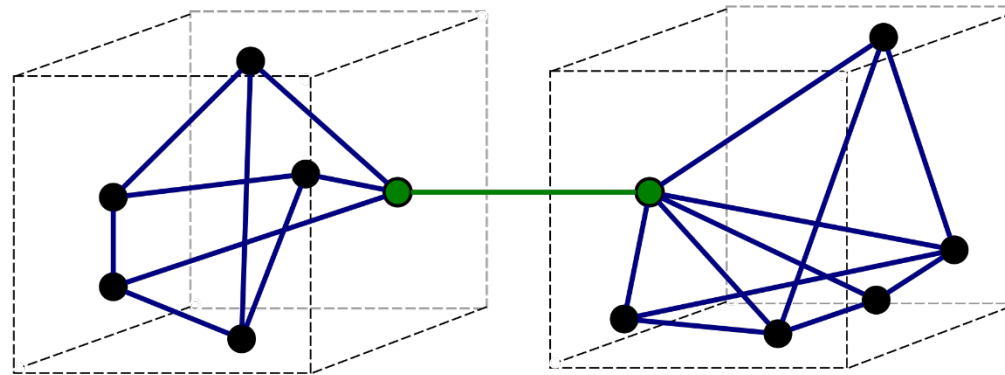
1. Compute average daily temperatures at various sites
2. Initialize sums and open files
3. Create new temperature record
4. Store temperature record
5. Close files and print average temperatures



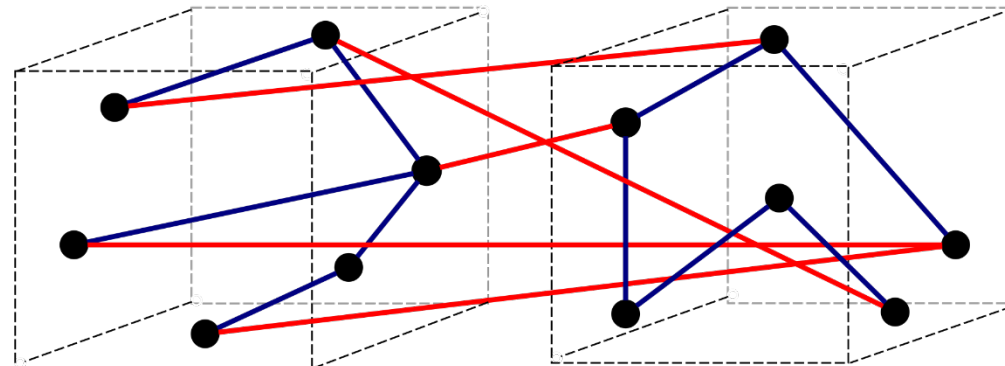
Coupling

Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

- The measure of the interdependence of one module to another
- Low coupling minimize ripple effect



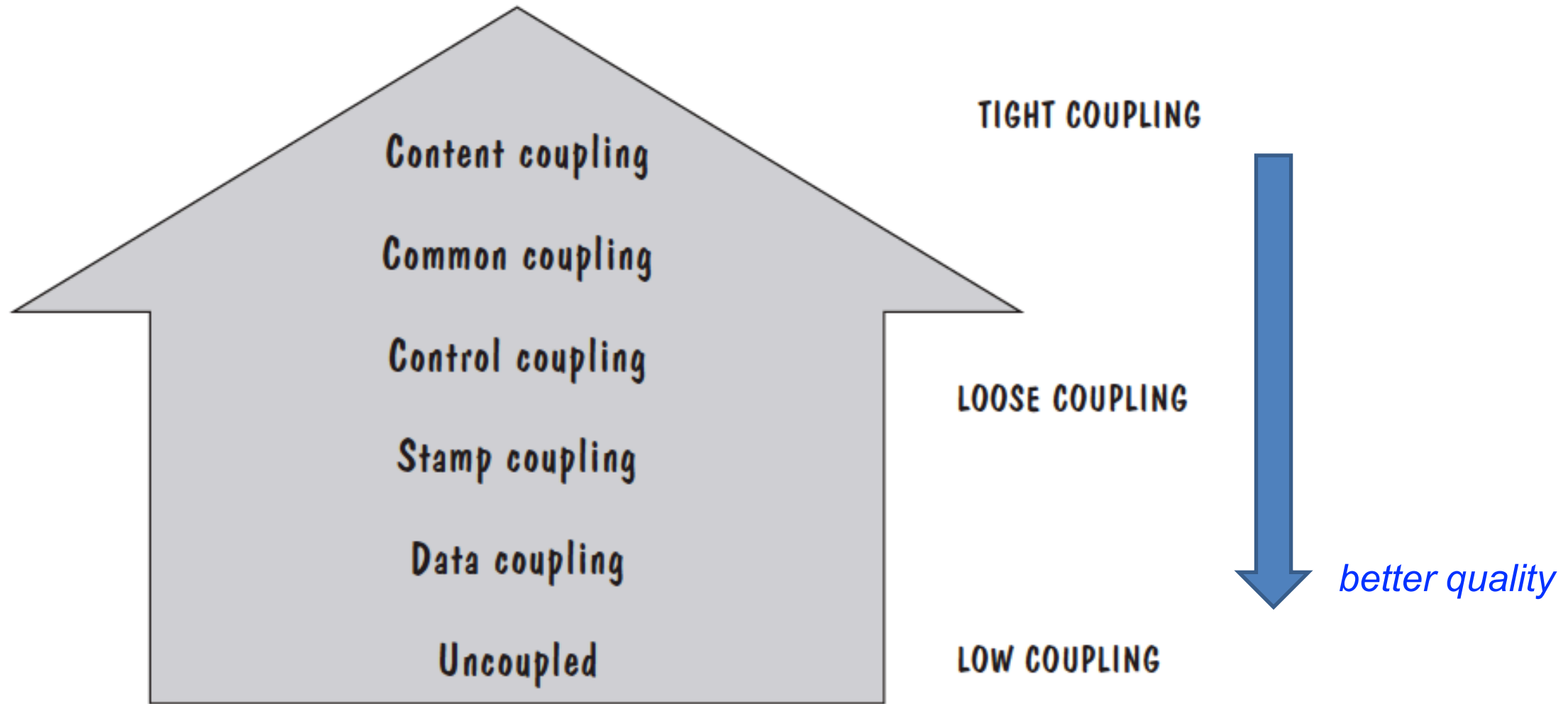
a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29

Levels of Coupling



Stephen R. Schach, "Practical Software Engineering", Richard D Irwin

David Stotts, Intramodule Coupling, COMP145, University of North Carolina (<https://www.cs.unc.edu/~stotts/COMP145/cohesion.html>)

Content Coupling

- One module branches into another module
- One module references or alters data contained inside another module
- Examples:
 - Module P modifies a statement of module Q
 - Module P refers to local data of module Q
 - Module P branches to a local label of Module Q

Common Coupling

- Two modules share the same global variables (data)
- Example: 'common' in Fortran
- Drawbacks:
 - Reduce readability
 - Can introduce side effects
 - Difficult to maintain
 - Difficult to reuse
 - May cause security problems

```
WHILE (global_variable == 0)
{
    if(parameter_xyz > 25)
        function_3( );
    else
        function_4( );
}
```

Control Coupling

- Two modules communicate through control flags
- One module explicitly controls the logic of the other
- Q: control-coupled?

Module P calls module Q, and passes back a flag to P that says, "I am unable to complete my task.", then Q is passing data.

If the flag means, "I am unable to complete my task; accordingly, write error message ABC123."

← control coupled

- Drawbacks:
 - Two modules are not independent
 - Generally associated with logical cohesion

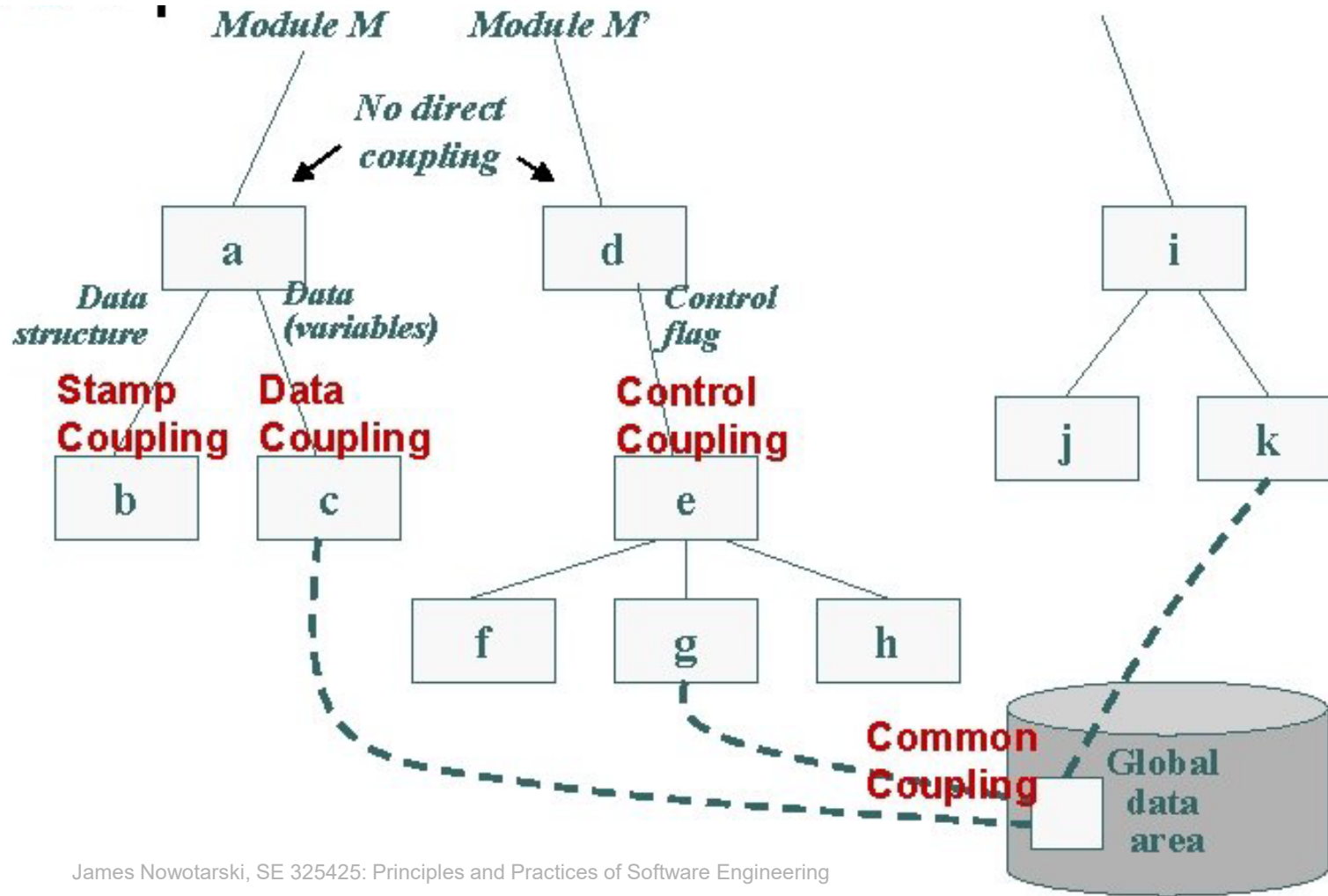
Stamp Coupling

- Two modules communicate via a passed data structure which contains more information than necessary
- Example:
 - Calculate_withholding(employee_record)
- Drawbacks:
 - Difficult to understand the interface
 - Difficult to reuse

Data Coupling

- Two modules communicate via parameters
- Should be a goal of design
- Easy to maintain
- Examples:
 - Display_time_of_arrival (flight_number)
 - Compute_product(first_number, second_number, result)

Coupling Examples



James Nowotarski, SE 325425: Principles and Practices of Software Engineering

Questions

- Is functional cohesion always good?

There might be some occasions in which we have to choose another type of cohesion instead of functional cohesion.

In what case and why?

- Is data coupling always good?

There might be some occasions in which we have to use common coupling, i.e., global variables.

In what case and why?

QUESTIONS?