CS350 Lecture 7
Software Design

Fall 2018

Doo-Hwan Bae

SoC, KAIST

bae@se.kaist.ac.kr

“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.”
- But, in 2030, when today’s 10-year-olds are in the job market, they’ll need to be creative, problem-solving design thinkers who can teach a machine how to do things.
- http://www.newsweek.com/2014/06/06/computer-programming-dying-art-252618.html

“Computer Programming is a Dying Art”

(Source: Newsweek, by Kevin Maney, 2014)

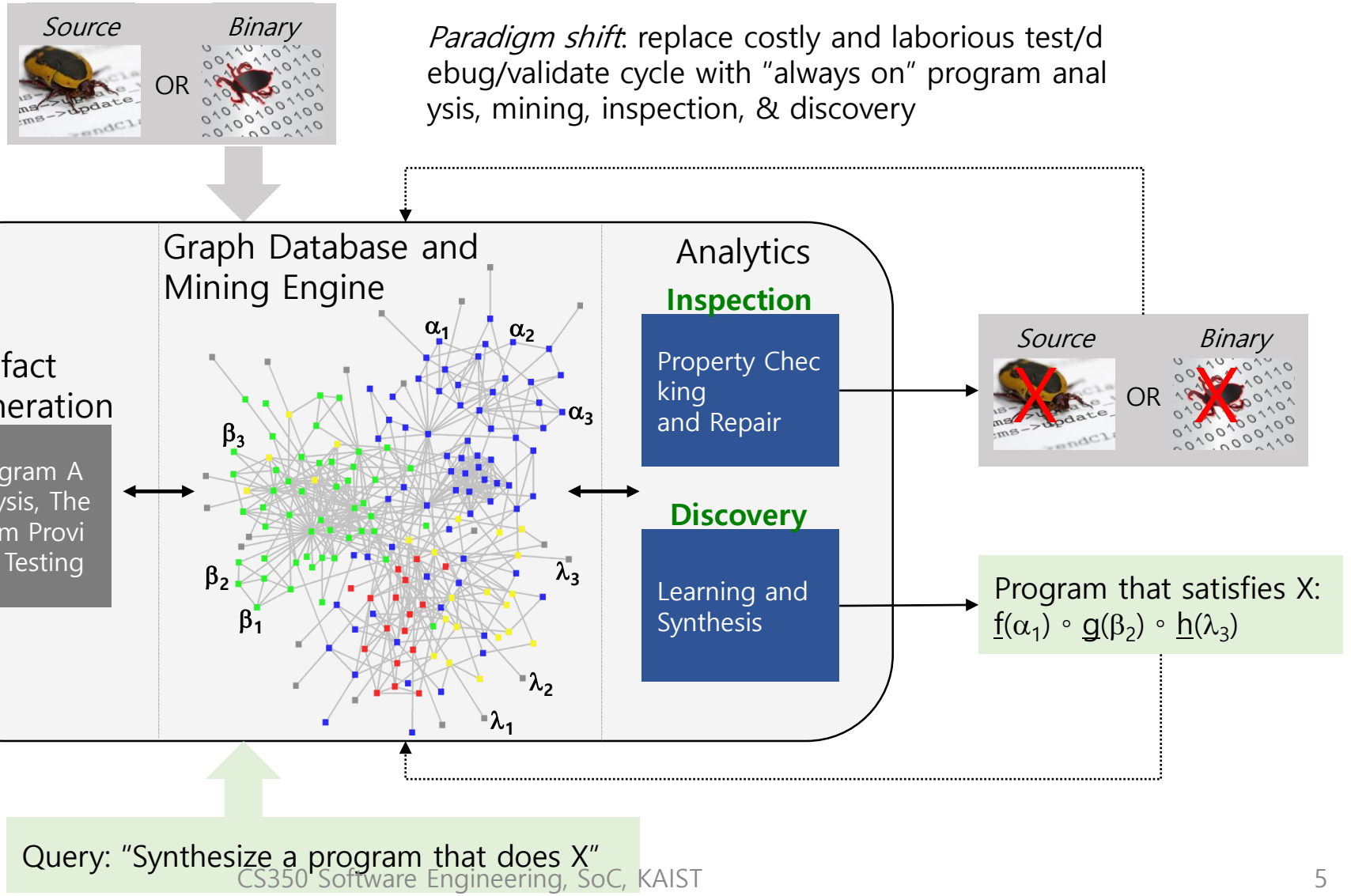
- IBM Research chief John Kelly likes to say we’re at the dawn of an era of brain-inspired cognitive computers—and the beginning of the end of the 60-year reign of programmable computers that required us to tell them what to do step by step.
- The next generation of computers will learn from their interactions with data and people.
- In another decade or so, we won’t program computers—we’ll teach them

“Computer Programming is a Dying Art”

(Source: Newsweek, by Kevin Maney, 2014)

- MUSE(Mining and Understanding Software Enclaves)
- http://www.darpa.mil/program/mining-and-understanding-software-enclaves
- The MUSE program is interested in close and continued collaboration of experts from a range of fields, including but not limited to: programming languages, program analysis, theorem proving and verification, testing, compilers, software engineering, machine learning, databases, statisticians, systems and a multitude of application domains. The program intends to emphasize creating and leveraging open source technology.

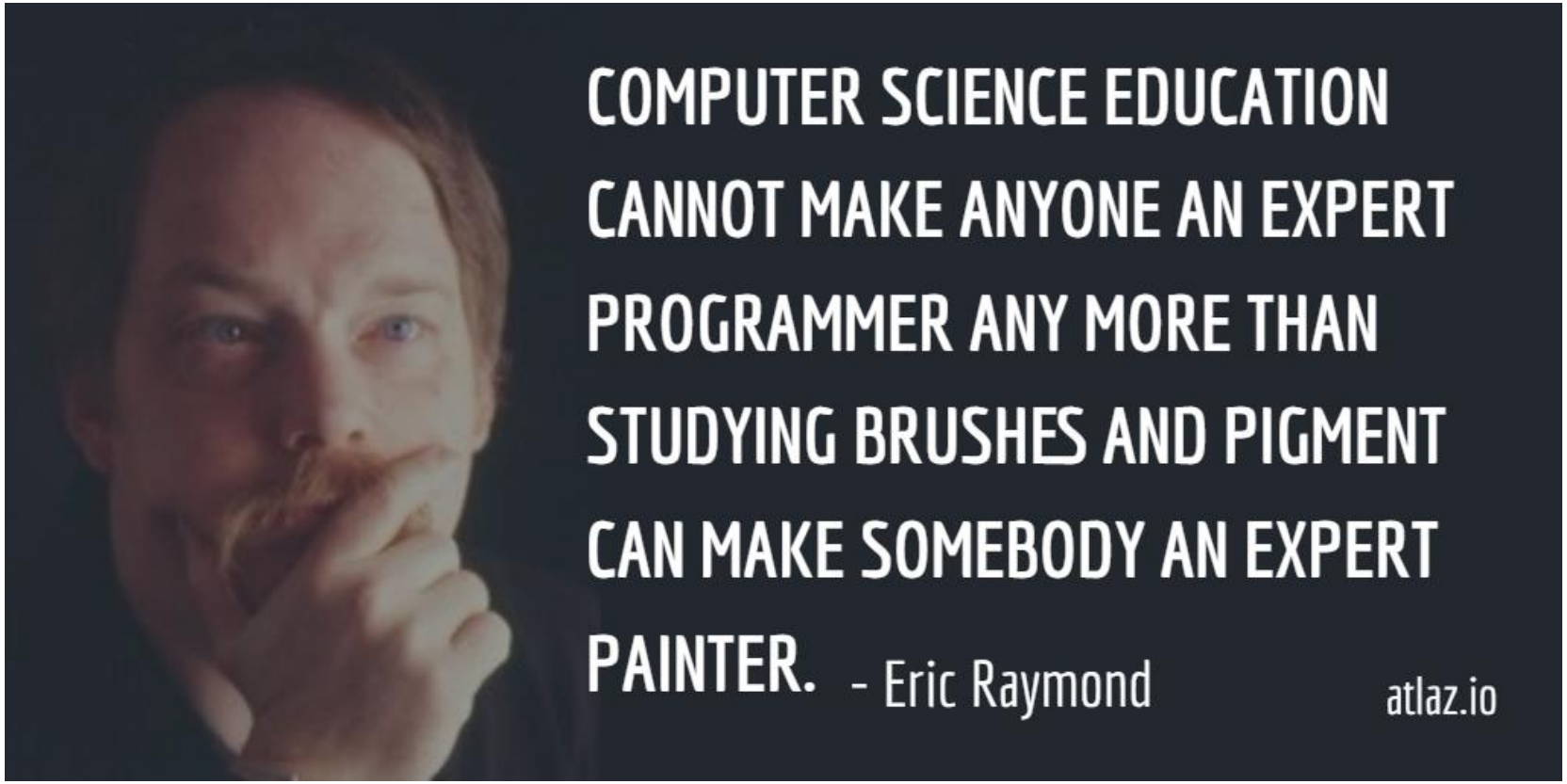
A New SW Development Paradigm: MUSE?



Art in Programming/Software Engineering

- Is a programming *art*?
  - Programming consists of a problem understanding and proposing a source code to it.
  - Proposing a source code includes
    - Logic/data structure design/selection + coding& debugging
    - Analysis/design activities are included, but mostly done implicitly.
  - Another Q: Is programming an art or science?
- If *artistic activity* exists in Software Engineering, what activity will be?
  - Engineering: Requirements, *Design*, Coding, Testing, Maintenance, …
  - Supporting and managing processes
- Can we teach/learn how to be an artist/ a programmer?

A Big Guy's Thought on Programming!



Teaching/Learning Artistic Activities in SW Development

- Drawing a picture vs. Programming/Software Engineering
  - Let us discuss Similarity vs. Difference

- What/How to learn to be a good programmer/software engineer?
  - Let us discuss

What is SW ‘Design’?

- A process to transform user requirements into some suitable form, which helps the programmer in software coding.

- Software design is the first step, which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

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 S/W that can be assessed for _____ .

Design Concepts

- Abstraction
- Stepwise refinement: decomposition
- Modularity ➔ Modularization
- Information hiding
- Top-down versus Bottom-up

Modularization (1/2)

- 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. These modules may work as basic constructs for the entire software.
- Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modularization(2/2)

- A complex problem is broken down into several manageble pieces.
- Let P1 and P2 be the two problems
- Let E1 and E2 be the effort required to solve P1 & P2, resp.
- If C(P1) > C(P2), then E(P1) > E(P2)
  - C: Complexity
  - E: Effort
- If C(P1+P2) > C(P1) + C(P2), then E(P1+P2) > E(P1)+E(P2)
- It is easier to solve a complex problem when it is broken into smaller pieces.

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.
- Information hiding is the idea that a design decision should be hidden from the rest of the system to prevent unintended coupling.
- Encapsulation is a programming language feature.
- Information hiding is a design principle.

  (source: wiki)

Effective Modular Design

- Functional independence
- Based on the concepts of abstraction and information hiding.
- Each module has
  - a specific functionality of requirements
  - a simple interface when viewed from outside.
- Measured by two qualitative criteria:
  - cohesion
  - coupling

Cohesion

- The measure of strength of the association of elements within a module.
- Modules whose elements are strongly and genuinely related to each other are desired.
- Levels of cohesion
  - Coincidental
  - Logical
  - Temporal
  - Procedural
  - Communicational
  - Informational
  - Functional

better quality

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

```
   ...
   ...
s: A;
   B;
   C;
   ...
   ...
t: A;
   B;
   C;
   ...
   ...
   ...
```

```
Subroutine P
      A;
      B;
      C;
End P;
      ...
s: P
      ...
      ...
      ...
t: P
      ...
      ...
```

Logical Cohesion

- The elements in a module perform similar or related functions that fall into the same logical class.
- Examples:
  - InputAll, OutputAll.
  - General error handling.

Temporal Cohesion

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

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;

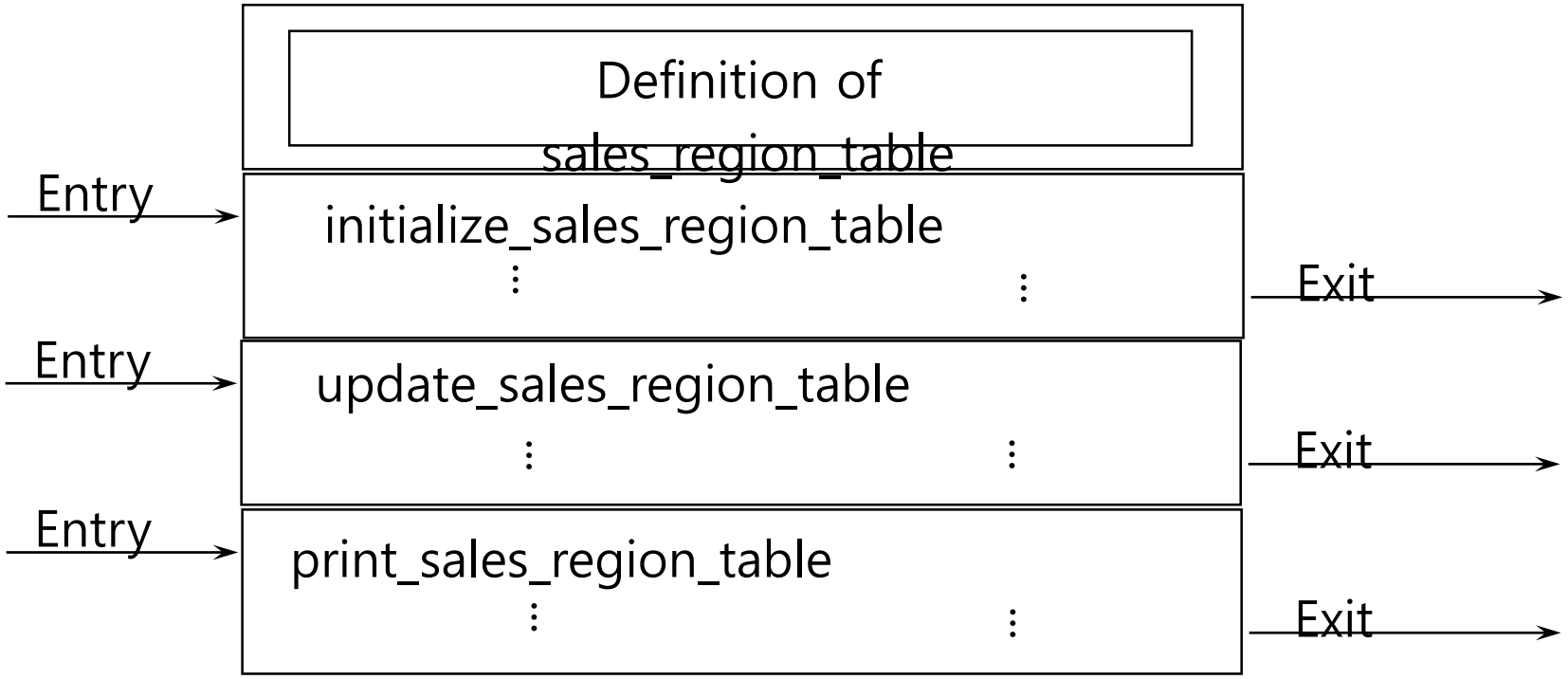
Communicational Cohesion

- All of the elements in a module operates on the same input data or produce the same output data.
- Examples:
  - Calculate new trajectory and send it to printer;
  - 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

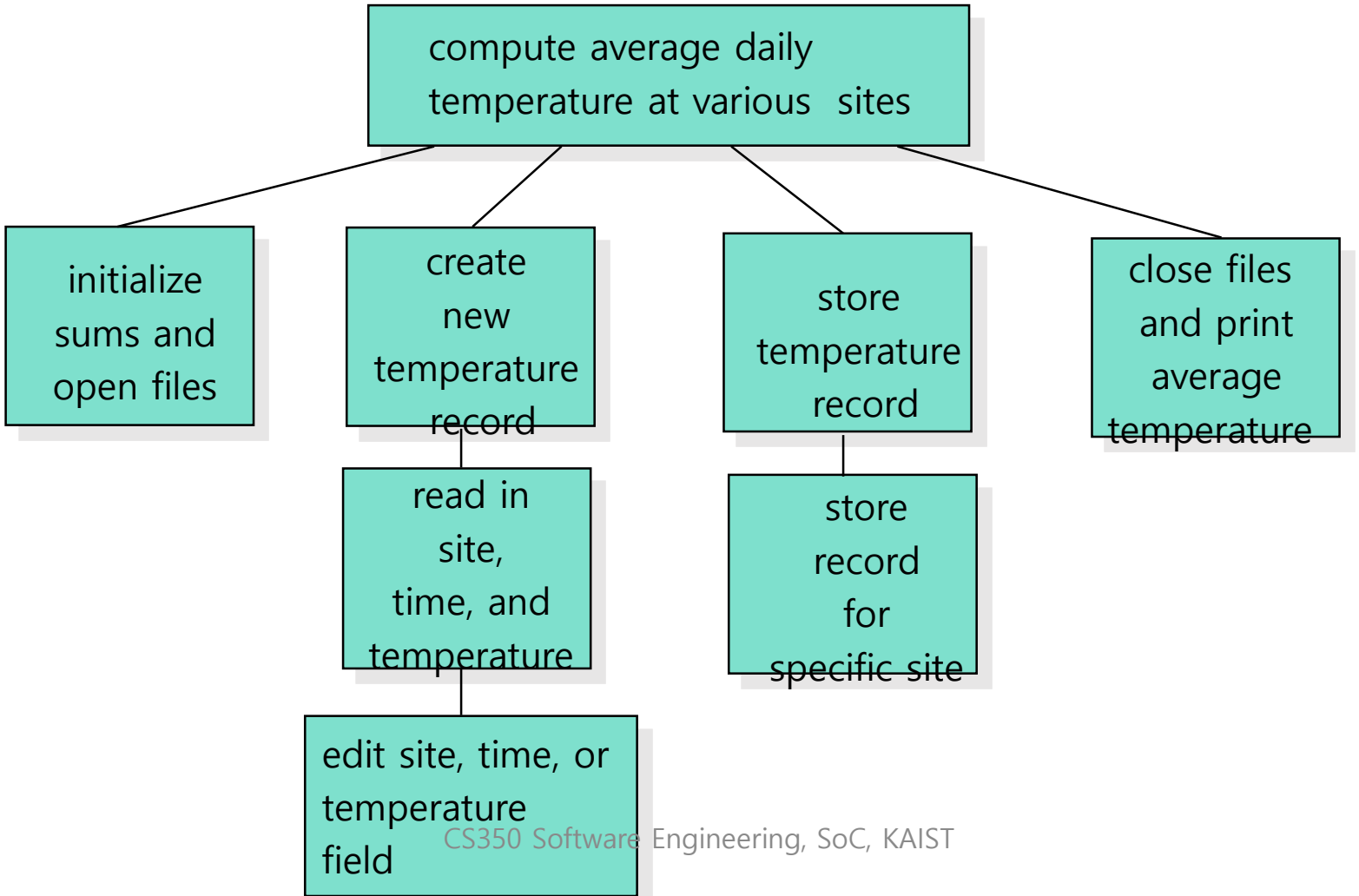
Module with Informational Cohesion



Functional Cohesion

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

Cohesion Exercise



Coupling

- The measure of the interdependence of one module to another.
- Low coupling minimize ripple effect.
- Levels of coupling
  - Content
  - Common
  - Control
  - Stamp
  - Data

better quality

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
- Example: ‘common’ in Fortran

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

Common Coupling

- Drawbacks:
  - Reduce readability.
  - Can introduce side effects.
  - Difficult to maintain.
  - Difficult to reuse.
  - May cause security problems.

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).

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?

Design Classification: Management Aspects

- Preliminary design: transformation of requirements into data and software architecture.
- Detailed design: refinements to the architectural representation that lead to detailed data structure and algorithmic representations for software

Design Classification: Technical Aspects

- Data design: transformation of the information domain model created during analysis into the data structures.
- Architectural design: definition of the relationship among major structural components of the software.
- Procedural design: transformation of structural components into a procedural description of the software.
- (User) Interface design: establishment of the layout and interaction mechanisms for human-machine interaction.

Data (Base) Design

- Select/Design data representation
- Concept of persistency
  - Operation(behavior) > Data
  - Operation = Data
  - Operation < Data

Architectural Design

- Goal:
  - To develop a modular program structure.
  - To represent the control relationships between modules.
  - Also, melds program structure and data structures, defining interfaces.
- A story:

You've saved money, purchased a beautiful piece of land, and have decided to build a house of your dreams. Having no experience in such matters, you visited a builder and explain your desire[e.g., size and number of rooms, etc. The builder listens carefully, asks a few questions, and then tells you that he'll have a design in a few weeks. As you wait anxiously for his call, you come up with many different images of your new house. What will he come up with? Finally, the phone rings and you rush to his office. Pulling out a large folder, the builder spreads a diagram of the plumbing for the second floor bathroom in front of you and proceeds to explain it in detail. "But what about the overall design!" you say.

"Don't worry,'' says the builder,

"We will get to that later."'

Procedural Design

- Define algorithmic details.
  - sequence, control, repetition.
- Notations:
  - Graphical design notation
    Flowcharts,
    Nassi-Shneiderman charts
  - Tabular design notation
    Decision tables
  - Program design languages
    Structured English
    Pseudocode

User Interface Design

- Step 1 Obtain background information
- Step 2 Define the semantics of the user interface
- Step 3 Define the syntax and format of the interface (including choices of interaction style)
- Step 4 Choose the physical devices
- Step 5 Developing the software
- Step 6 Integrate and install the system
- Step 7 Provide on-site and product support

Software Design Methods

- Data flow-oriented: Structured Design
- Object-Oriented
- Real-Time
- Distributed/Parallel

Structured Design

- Used in conjunction with Structured Analysis method.
- Design process

```
Refine data flow diagram
While not satisfactory do
  If type of flow is "transform" then
     Identify incoming/outgoing branches
       Map to transform structure
  else /*is "transaction", meaning a selection from many options*/
       Identify transaction center and data acquisition path
       Map to transaction structure
  endif
  Factor the structure
  Refine the structure using design heuristics
  Develop interface description andglobal data structure.
  Review
endwhile
Perform detailed design
```

Structured Design (Cont.)

- Components of Structured chart
  - Module 
  - Data/control flow 
  - Selection and iteration 
  - Reused modules 

Structured Design (Cont.)

- Design Heuristics(Guidelines)
  - Evaluate the first-cut program structure to reduce coupling and improve cohesion.
  - Attempt to minimize structures with high fan-out; strive for fan-in as depth increases.
  - Keep scope of effect of a module within the scope of control of that module.
  - Evaluate module interfaces to reduce complexity and redundancy and improve consistency.
  - Define modules whose function is predictable, but avoid modules that are overly restrictive.
  - Strive for single-entry-single-exit modules.
  - Package software based on design constraints and portability requirements.

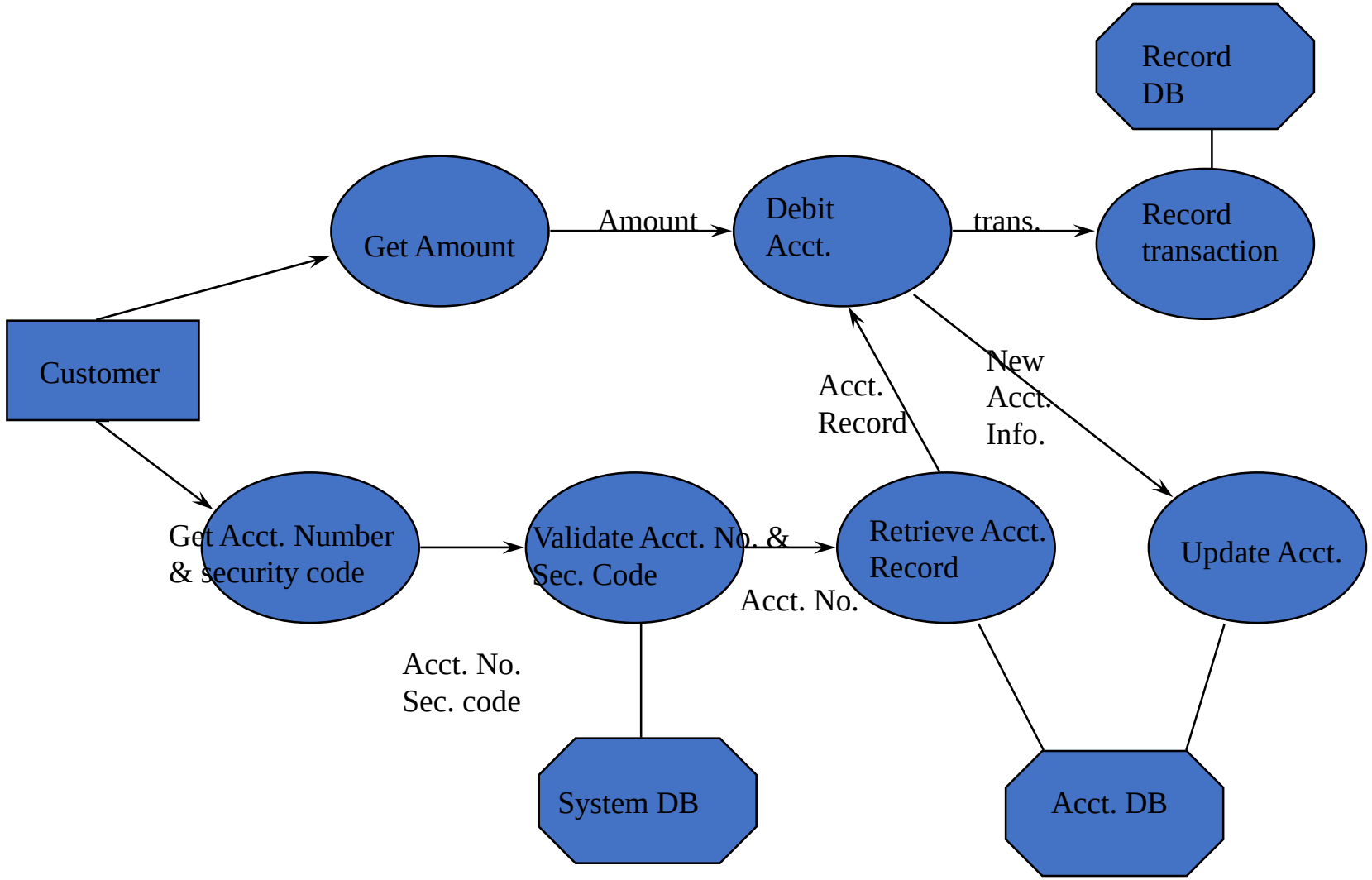
From DFD to Structured Chart

- Transform Analysis
- Transaction Analysis

An SA/SD Example

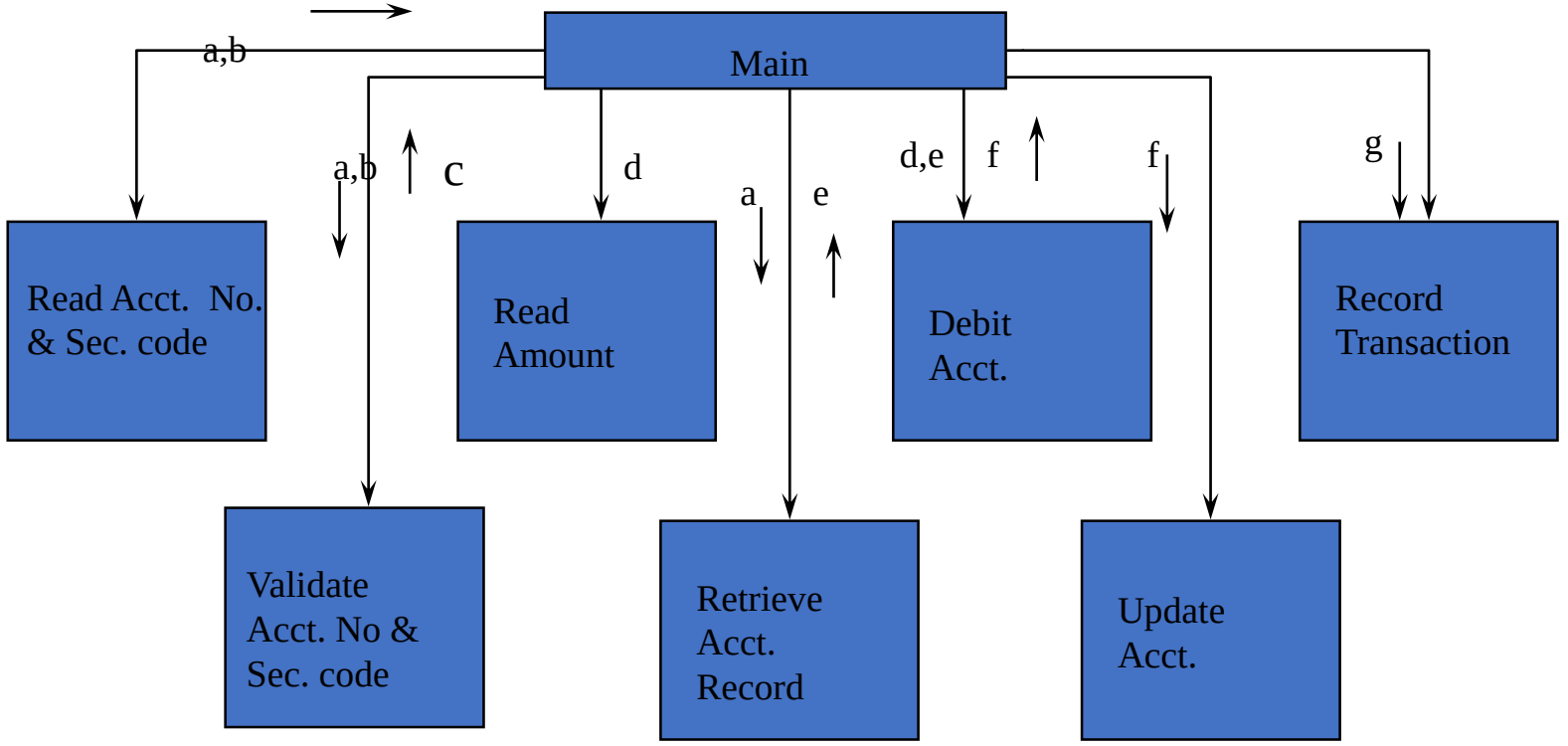
- Simplified Automatic Teller Machine (ATM) requirements :
  - ATM receives the customer's ATM card and security code, and validates the ATM card and security code.
  - Once they are validated, the account record is received from the account database, and then ATM asks the customer to enter an amount to be retrieved.
  - ATM processes the customer's transaction by debiting account, recording transaction to the transaction database and updating the account.

Data Flow Diagram for the ATM Example



Structured Chart for the ATM Example

(Not a good one!)



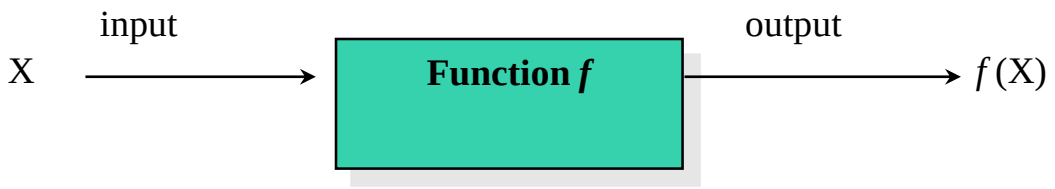
- a : Account number
- b : Security number
- c : Validation number
- d : Amount
- e : Account record
- f : New Account info.
- g : Transaction Info.

Object-Oriented Design

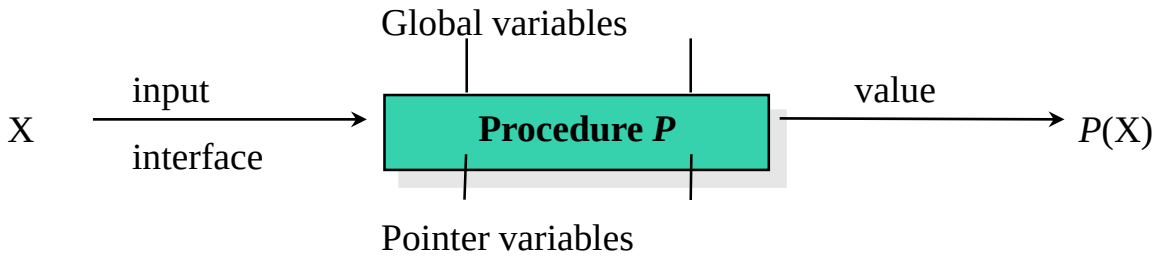
- Creates a representation of the real-world problem domain and maps it into a solution in such a way that the design interconnects data items and processing operations so that information and processing can be modularized together.
- A design method that supports the following principles efficiently:
  - Decomposability
  - Composability
  - Understandability
  - Continuity
  - Protection

Comparison of functions, procedures and objects.

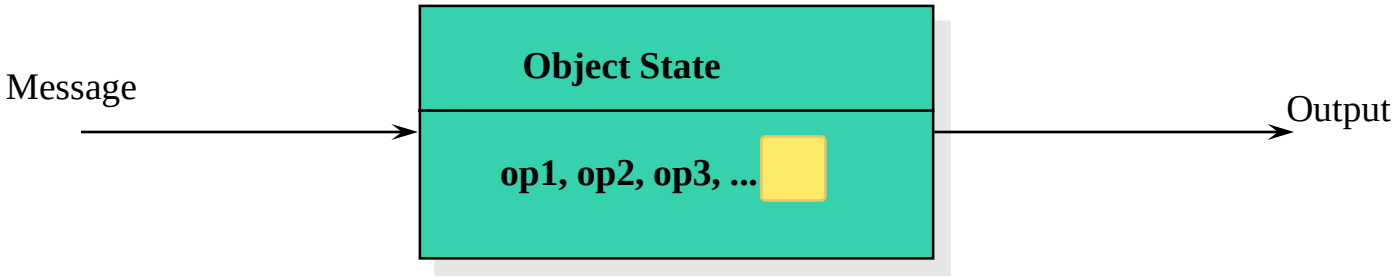
Functions



Procedures



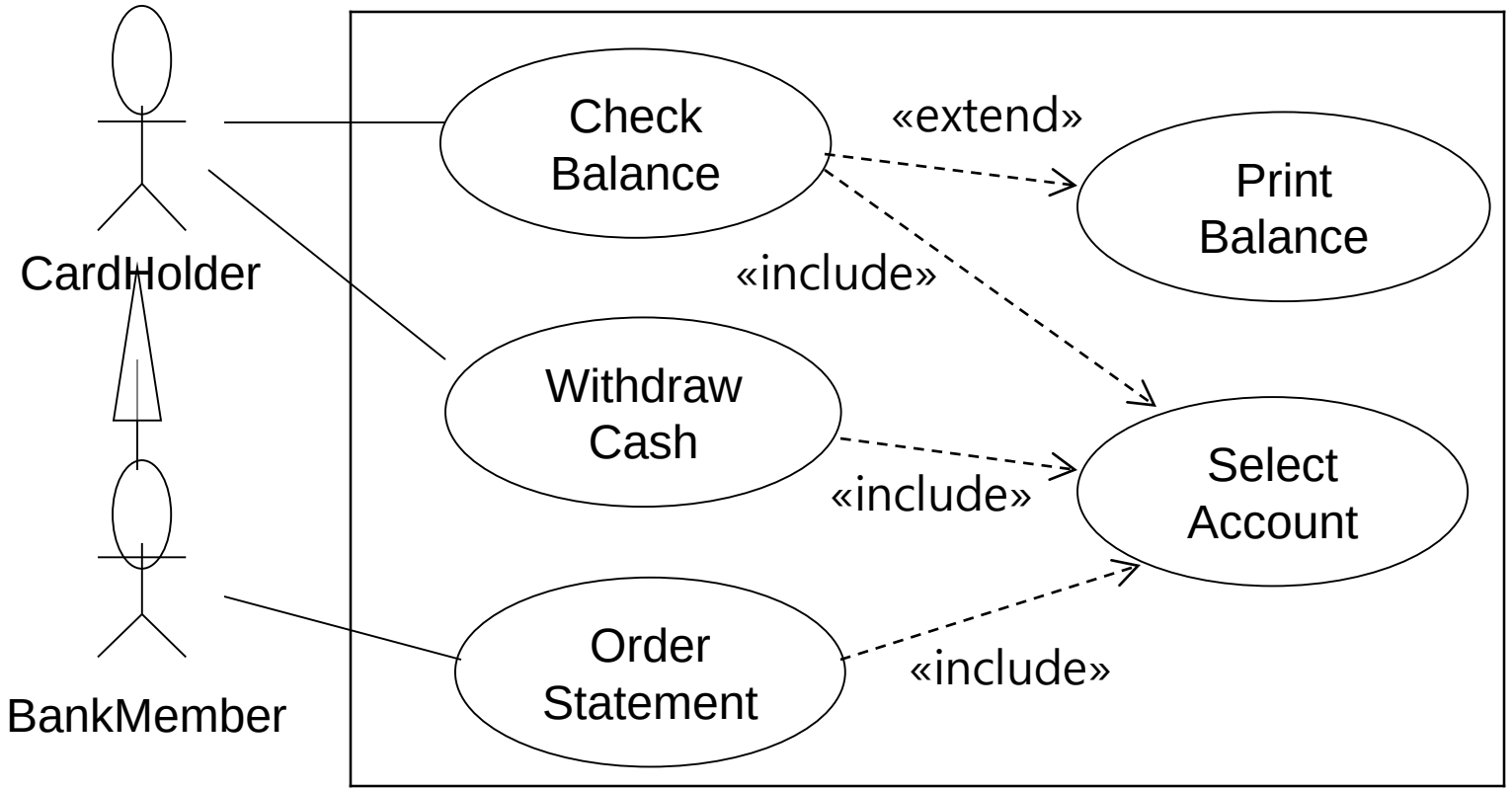
Objects



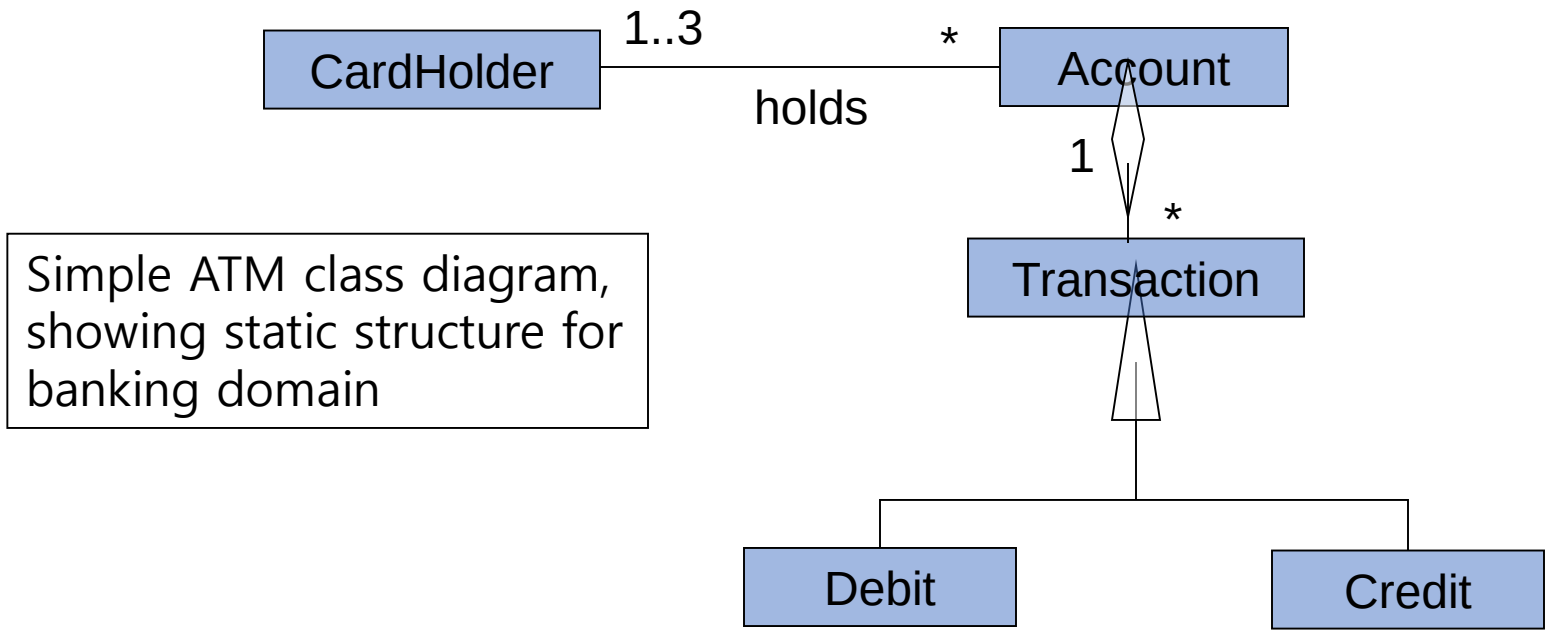
Object-Oriented vs. Functional

- Common OO approach steps
  - step1. Identify objects(attributes and methods)
  - step2. Determine the relationships among objects
  - step3. Design & implement objects
- Functional approach steps
  - step1. Identify functionality
  - step2. Determine the relationships among functions
  - step3. Design and implement functions

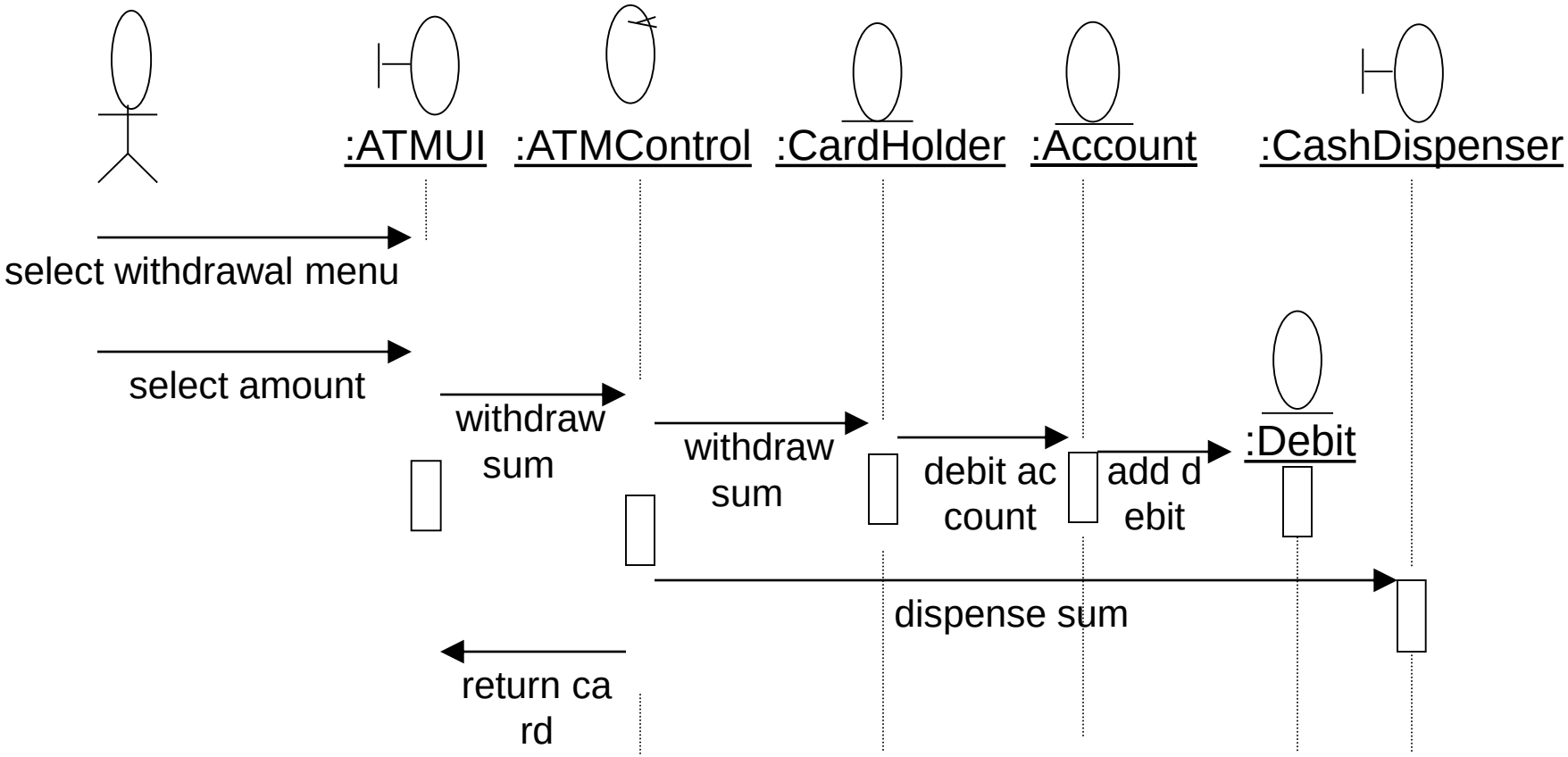
Use Case Diagram for the ATM Example



Class Diagram for the ATM Example



Sequence Diagram for the ATM Example



Summary

- OO approach supports SE principles better than old approaches
- Design is a place where SW quality is fostered
- Design experiences can be reused through design patterns
- Service-Oriented SE tries to enhance software development efficiency and cost-effectiveness by reusing large-granularity artifacts through standardization.
- What is next? MUSE!?