

Software Engineering (Part 1)

Copyright © 2025 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover these software engineering topics:

Stages of SW dev

How to order the stages

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- Process models

Objectives

Software Engineering lecture slide decks:

Part 1	Requirements analysis Design (general)
Part 2	Design (object-oriented) Implementation Debugging
Part 3	Testing Evaluation
Part 4	Maintenance Process models

Software Engineering

- Composing code is a part of what a software engineer does
- Let's consider all of the parts...

You've decided to create a software system.
What's your first step?

Agenda

- **Requirements analysis**
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- Process models

Requirements Analysis

- *Requirements analysis*
 - **Who** are the system's users?
 - **What** should the system do to fulfill the users' needs?

What kinds of requirements should you gather?

Requirements Analysis: Kinds

- Always:
 - **Functional** requirements
- Sometimes:
 - **Data** requirements
 - **Environmental** requirements
 - **Usability** requirements

How should you go about gathering those requirements?

Requirements Analysis: Gathering

- Questionnaires
 - Interviews
 - Focus groups
 - Direct observation
 - Studying documentation
 - Researching similar products
-
- Users visit
the pgmmers
- Pgmmers visit
the users

Yvonne Rogers, Helen Sharp, Jenny Preece. *Interaction Design: Beyond Human-Computer Interaction (3rd Edition)*. Wiley, 2011.

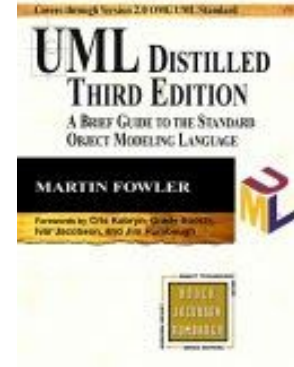
How should you structure the requirements that you've gathered?

Requirements Analysis: Structuring

- Create models of the user's domain
 - A popular set of modeling notations...
 - *Unified Modeling Language (UML)*

Requirements Analysis: Structuring

- *Unified Modeling Language (UML)*



Grady
Booch



James
Rumbaugh



Ivar
Jacobson

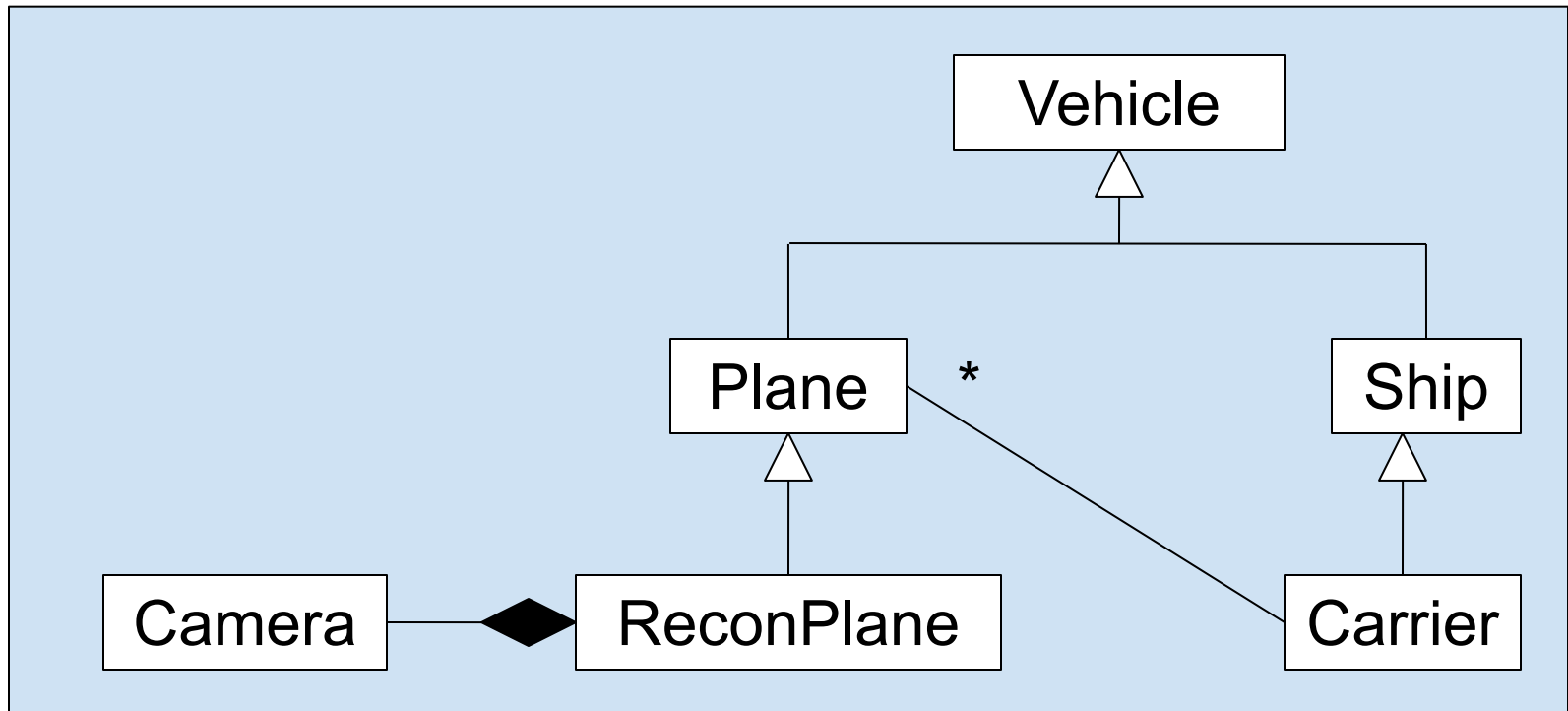
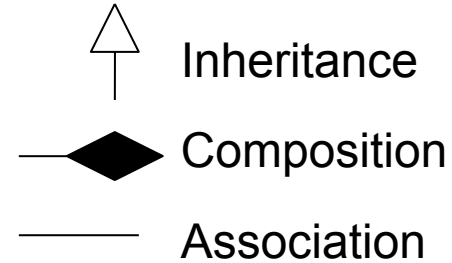
“The three amigos”

Requirements Analysis: Structuring

- Create ***Class Model(s)***
 - A UML notation
 - Describes classes of objects in the user's domain

Requirements Analysis: Structuring

Class model example:



Requirements Analysis: Structuring

- Create ***Scenarios***
 - A story describing a user interaction with the (anticipated) system to achieve some goal

Requirements Analysis: Structuring

- Create *Wireframes* and *Storyboards*
 - Low-tech
 - High-tech

Requirements Analysis: Structuring

- Create (tentative) *database schema*
 - Tables, fields
 - Relationships among tables
 - Primary and foreign keys

Requirements Analysis: Structuring

- Create *Prototype(s)*
 - Low-fidelity
 - High-fidelity

You probably can't fulfill all of the user's requirements. And you certainly can't fulfill all of the user's requirements right away. How should you prioritize the requirements?

Requirements Analysis: Prioritizing

- The *MoSCoW method*
 - Define each system feature as:
 - **M**: must have
 - **S**: should have
 - **C**: could have
 - **W**: won't have (this time)

Requirements Analysis: Conclusion

- In the **academic** world:
 - Student programmers often are given requirements
- In the “**real**” world:
 - (Senior) programmers often must know how to **gather**, **structure**, and **prioritize** requirements

You've determined the kinds of requirements that are relevant, gathered them, structured them, and prioritized them. What should you do next?

Agenda

- Requirements analysis
- **Design**
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- Process models

Design

- *Design*
 - **How** should the system work?

How should you specify the system's design?

Design: Use Cases

- Create ***use cases***
 - A use case is an elaboration of a scenario
 - A use case is *detailed* enough to be testable by QA engineers

Design: Models

- Create ***Specification Class Model(s)***
 - Conceptual class model (deja vu)
 - Models concepts/classes in the user's domain
 - Specification class model
 - Models concepts/classes in the program

What heuristics should you keep in mind when designing the system?

Design: Heuristics

- **Use design heuristics**
 - Some are general
 - Some are specific to OO programming

Design: General Heuristic 1

- (Kernighan) **Detect** errors low; **handle** errors high
- (Dondero) **Detect** errors low; **handle** errors as low as you can

Brian W. Kernighan and Rob Pike.
The Practice of Programming.
Addison-Wesley. Reading, MA, 1999.

Design: General Heuristic 1

(A) Asgt 1: database.py

```
def get_overviews(query) :  
    ...  
    try:  
        Use the database.  
    except Exception as ex:  
        Write error msg to stderr.  
        sys.exit(1)  
    ...  
    Return the class overviews.
```

Design: General Heuristic 1

(B) Asgt 1: database.py

```
def get_overviews(query):  
    ...  
    Use the database.  
    ...  
    Return the class overviews.
```

Design: General Heuristic 2

- Seek *strong cohesion* within modules
 - The components (fields, functions/methods) of a module should be related to each other
 - Empirically: not significant

Design: General Heuristic 2

(A) Asgt 1: database.py

```
def get_overviews(query):  
    ...  
    Use the database.  
    ...  
    Return the class overviews.  
  
def write_overviews(classes):  
    ...  
    Write the class overviews to stdout.
```

Design: General Heuristic 2

(B) Asgt 1: database.py

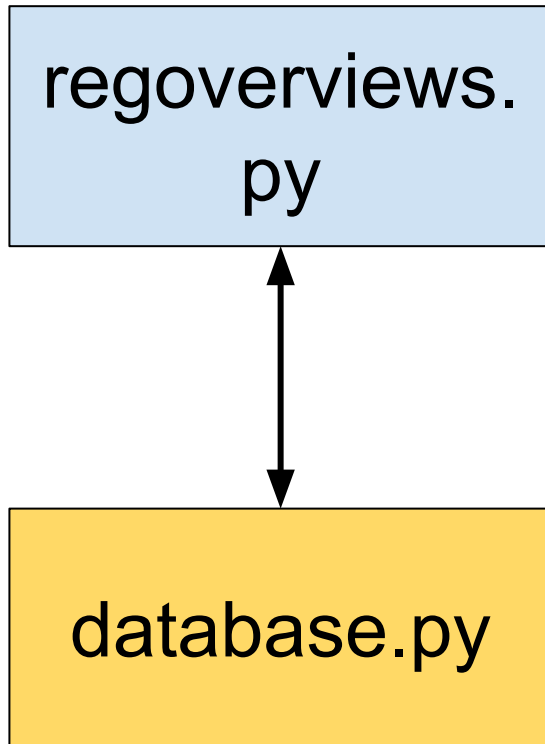
```
def get_overviews(query) :  
    ...  
    Use the database.  
    ...  
    Return the class overviews  
  
def get_details(classid) :  
    ...  
    Use the database.  
    ...  
    Return the class details.
```

Design: General Heuristic 3

- Seek *weak coupling* among modules
 - Minimize interfaces
 - Encapsulate data
 - Hide design decisions
 - Empirically: **significant**

Design: General Heuristic 3

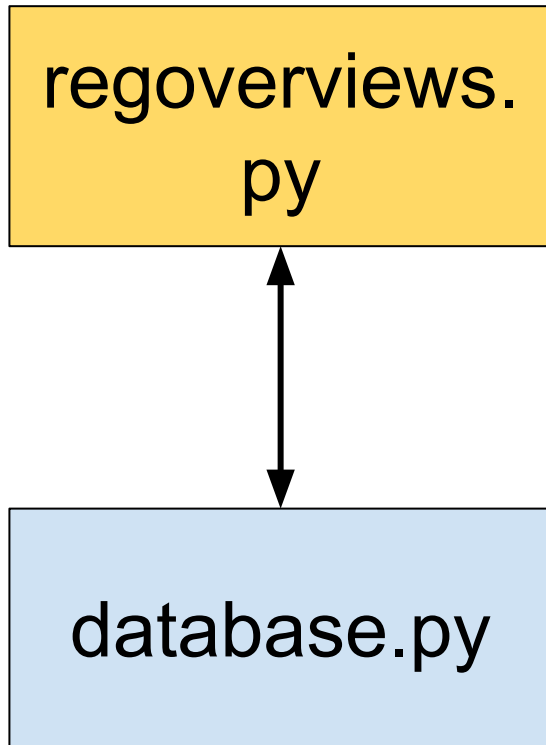
Asgt 1:



Hides design decisions
Which DBMS?
What table schema?
...

Design: General Heuristic 3

Asgt 1:

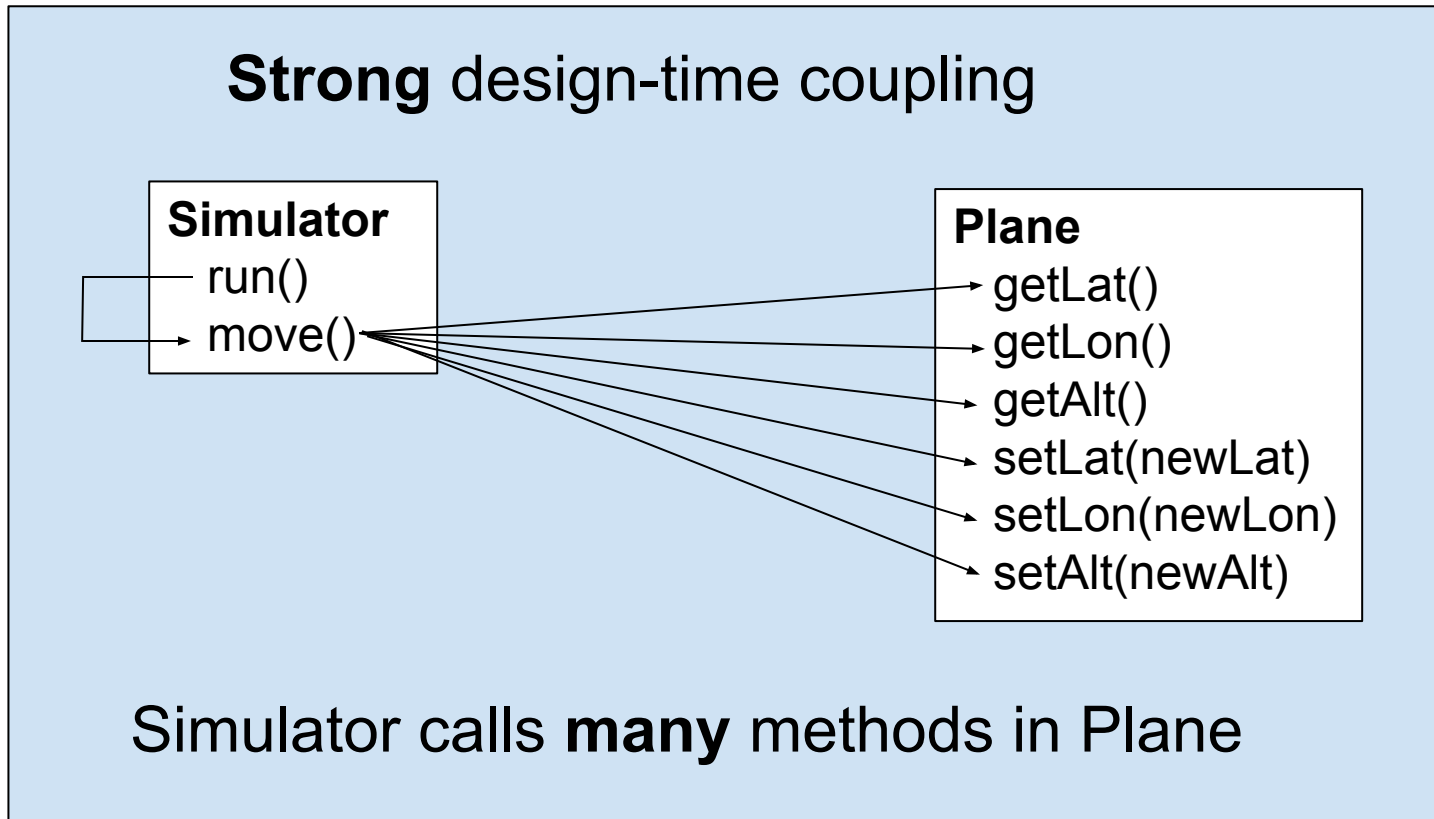


Hides design decisions
Command-line UI?
Web app UI?
Desktop/laptop UI?
...

Design: General Heuristic 3.1

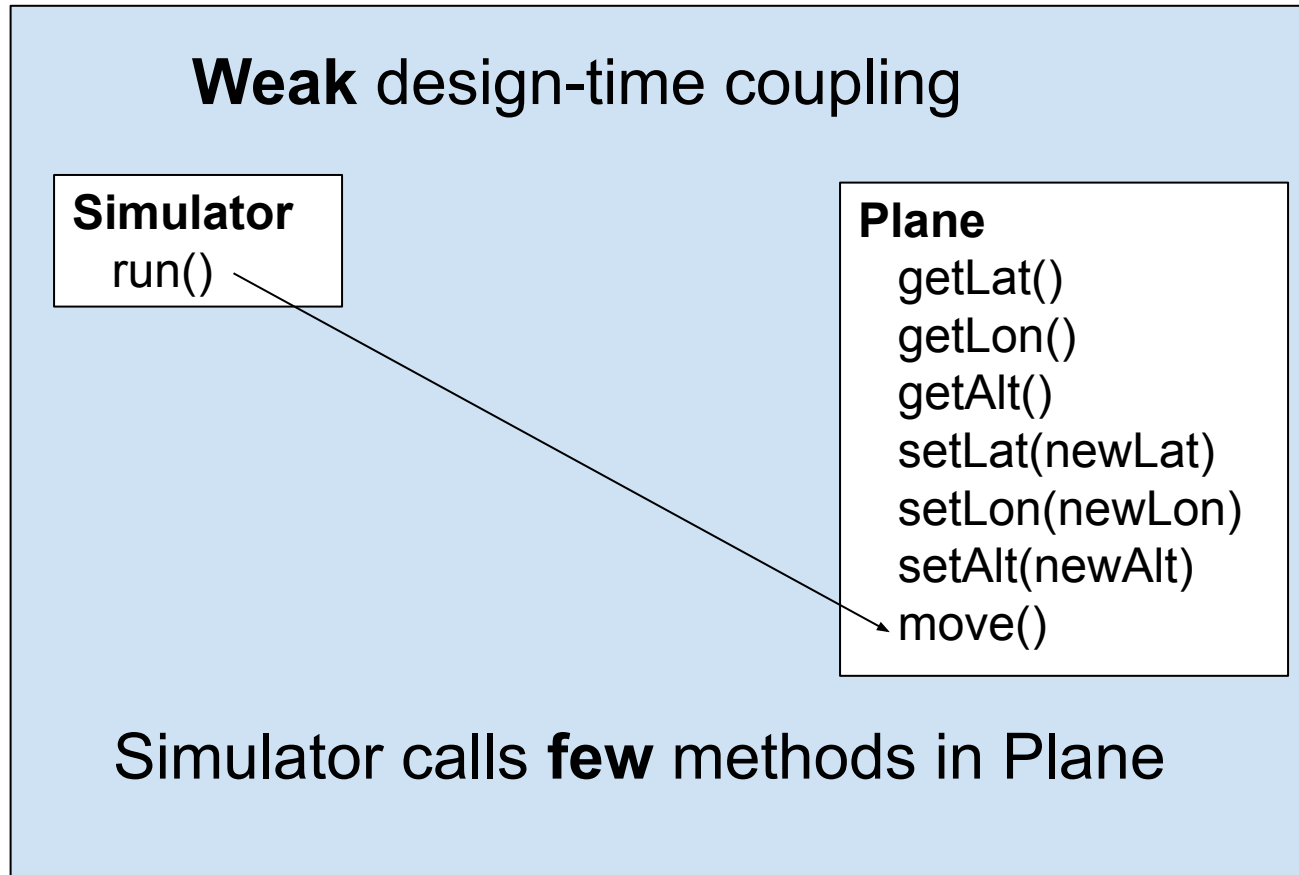
- Seek weak *design-time coupling*

Design: General Heuristic 3.1



No!

Design: General Heuristic 3.1



Yes!

Design: General Heuristic 3.1

(A) Asgt1: database.py

```
class Database:  
    ...  
    def connect():  
        ...  
    def get_overviews(query):  
        ...  
    def get_details(classid):  
        ...  
    def disconnect():  
        ...
```

Design: General Heuristic 3.1

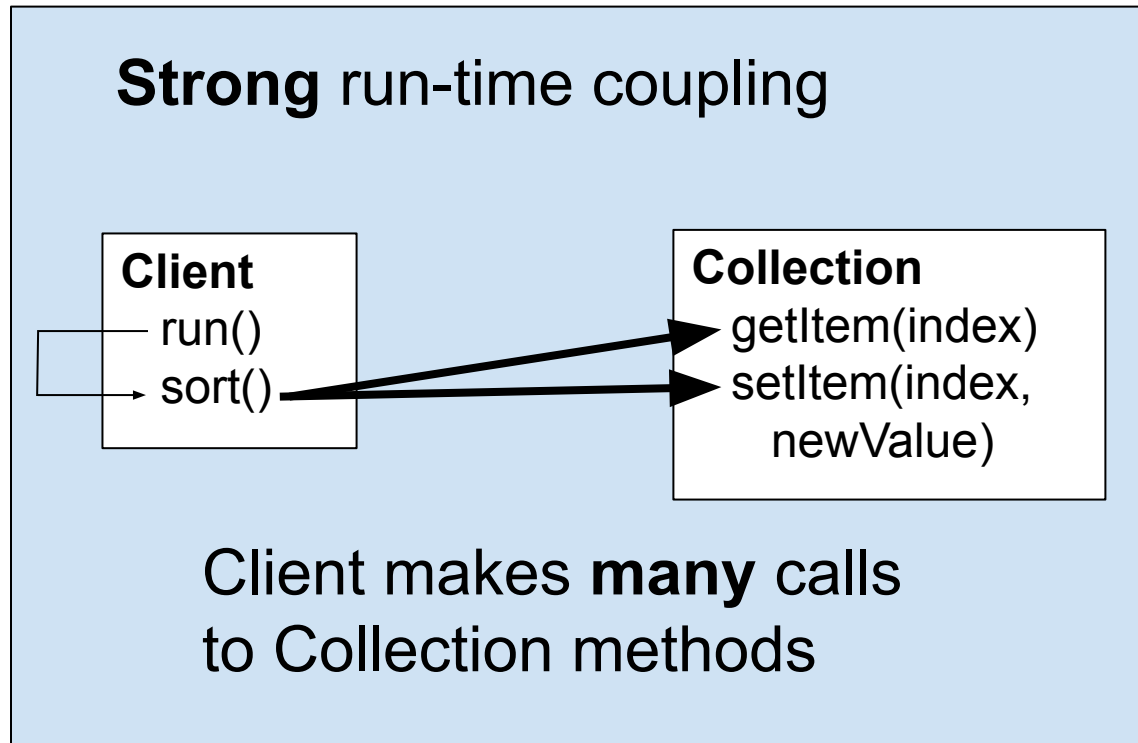
(B) Asgt1: database.py

```
def get_overviews(query):  
    Connect to the database.  
    Perform the query.  
    Disconnect from the database.  
    Return the class overviews.  
  
def get_details(classid):  
    Connect to the database.  
    Perform the query.  
    Disconnect from the database.  
    Return the class details.
```

Design: General Heuristic 3.2

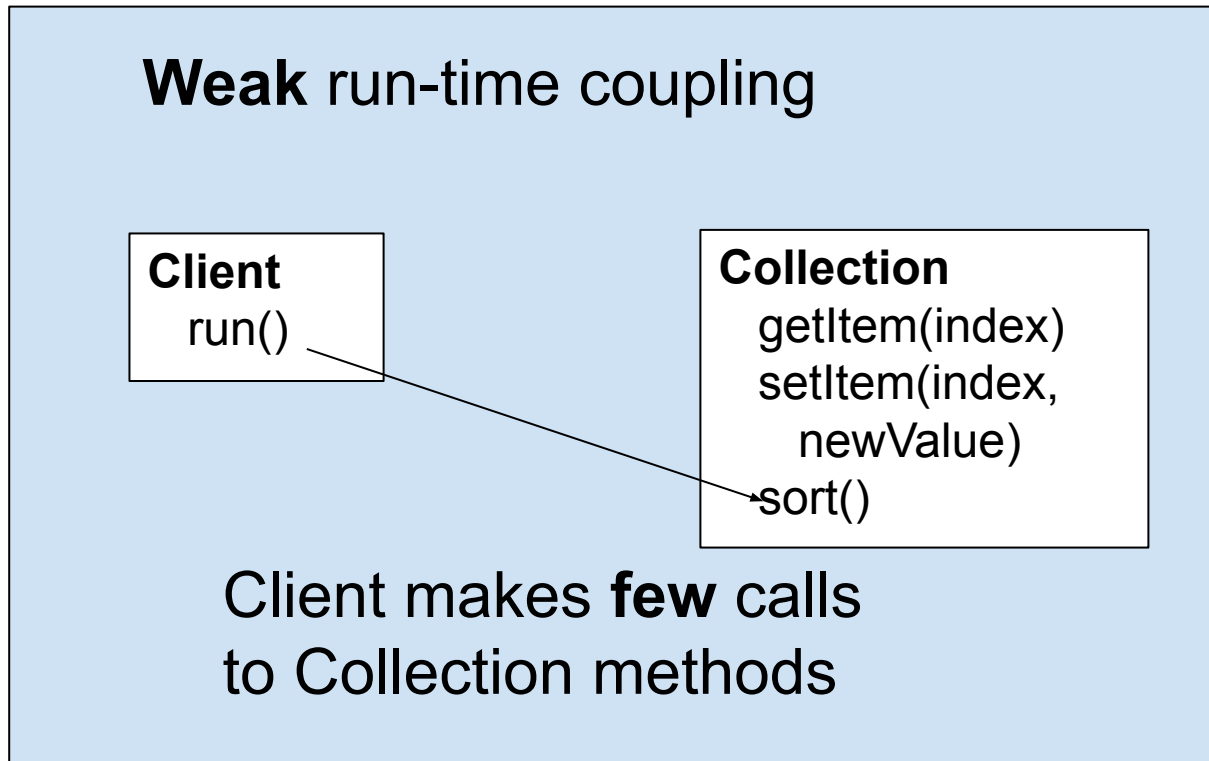
- Seek weak *run-time coupling*

Design: General Heuristic 3.2



No!

Design: General Heuristic 3.2

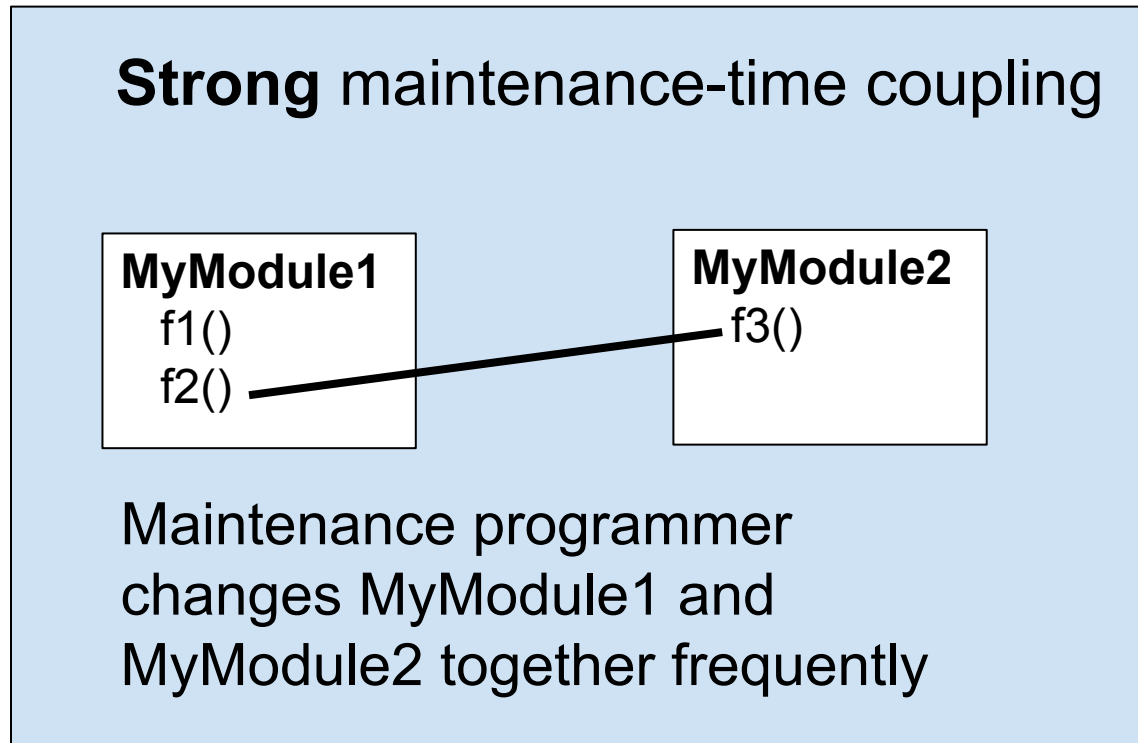


Yes!

Design: General Heuristic 3.3

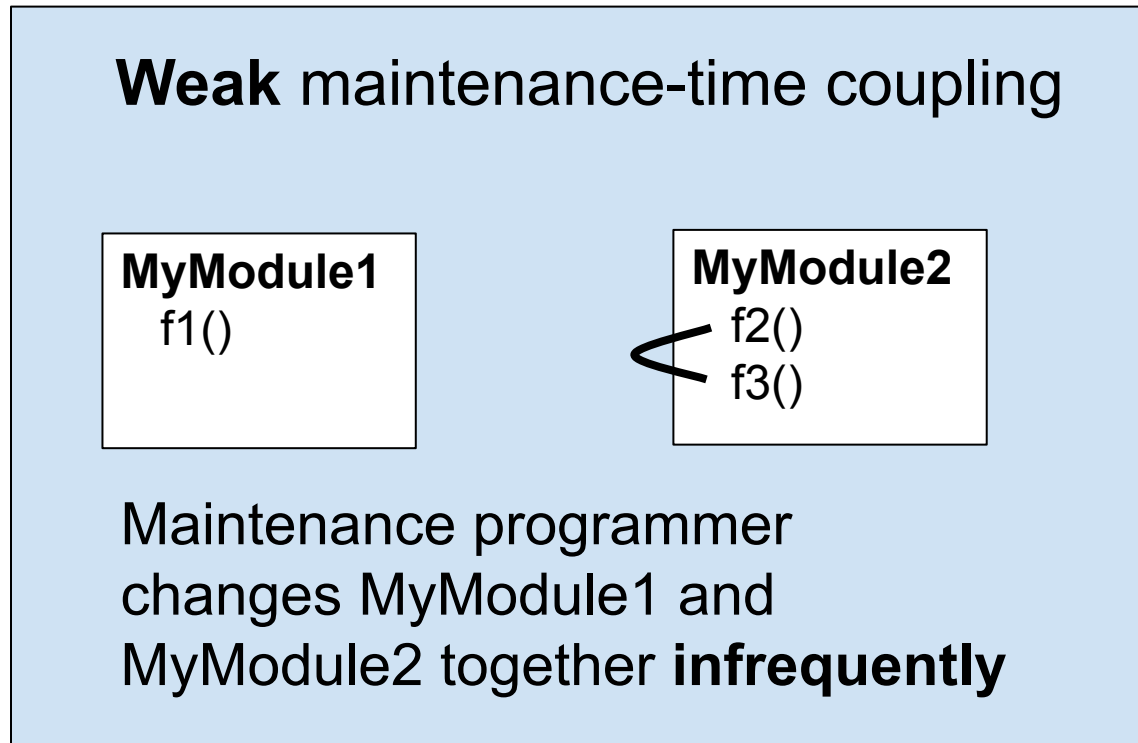
- Seek weak *maintenance-time coupling*

Design: General Heuristic 3.3



No!

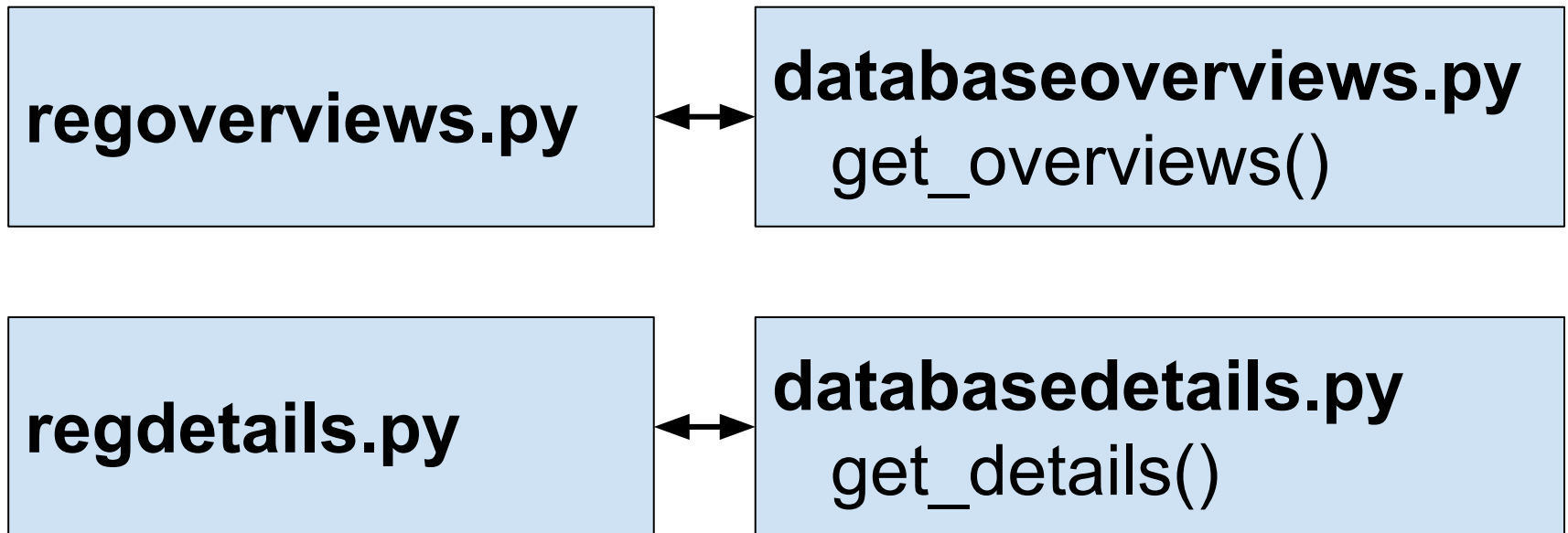
Design: General Heuristic 3.3



Yes!

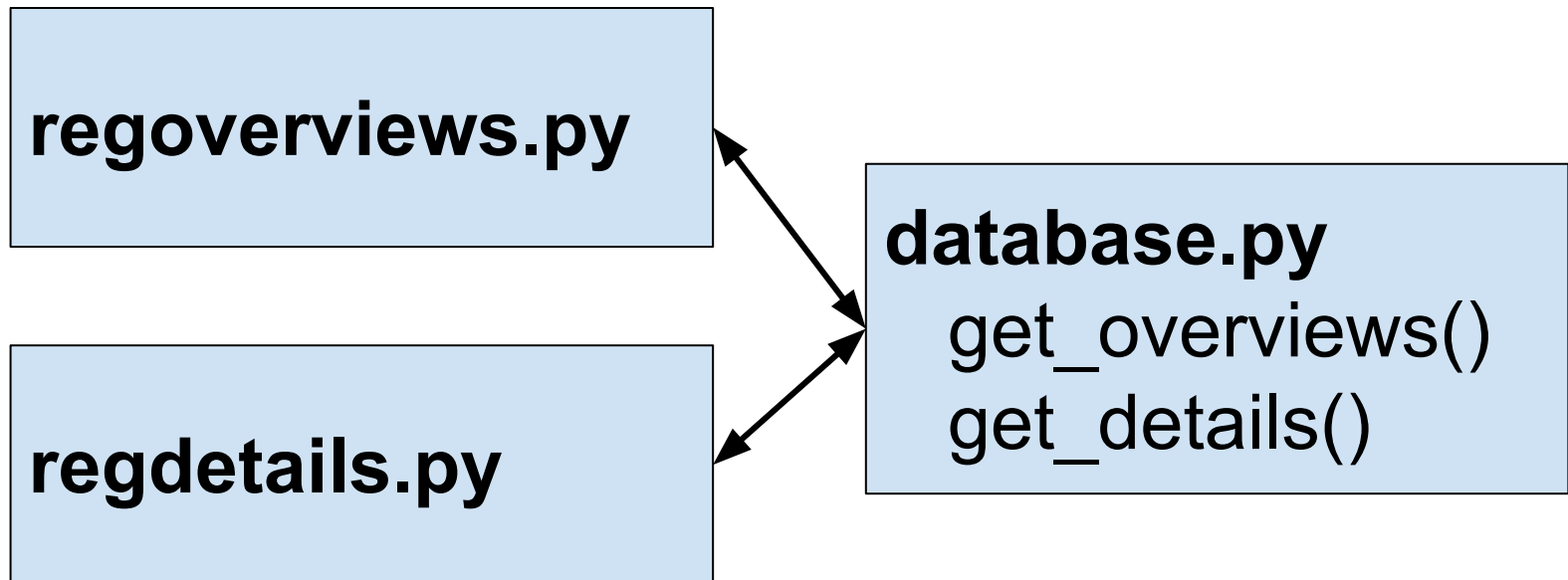
Design: General Heuristic 3.3

(A) Asgt 1:



Design: General Heuristic 3.3

(B) Asgt 1:



Continued in
Software Engineering (Part 2)...