Software Engineering (Part 1)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover these software engineering topics:
  - Requirements analysis
  - Design
  - Implementation
  - Debugging
  - Testing
  - Evaluation
  - Maintenance
  - Process models

Stages of SW dev

How to order the stages

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

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

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.

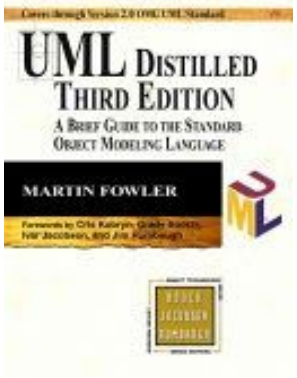
How should you structure the requirements that you've gathered?

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 Rumbauch

Ivar Jacobson

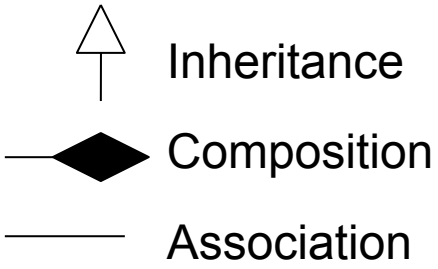
"The three amigos"

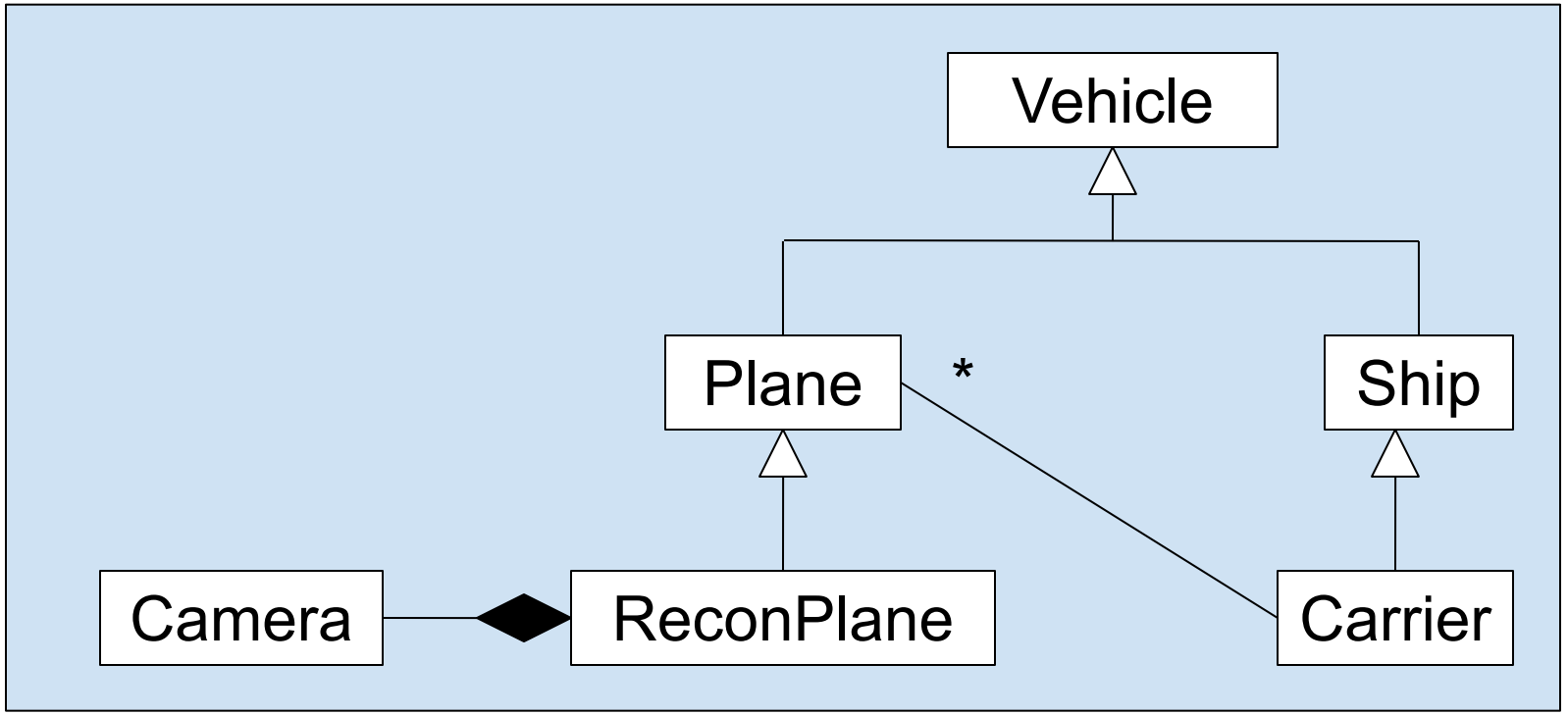
Requirements Analysis: Structuring

- 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 ***Prototype(s)***
  - Bridge from requirements analysis to design
  - 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: 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

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

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

Design: Heuristics

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

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) database.py

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

Design: General Heuristic 1

(B) database.py

```
def get_classes(query):
    …
    Use the database.
    …
    Return the classes.
```

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) database.py

```
def get_classes(query):
    …
    Use the database.
    …
    Return the classes.

def write_classes(classes):
    …
    Write the classes to stdout.
```

Design: General Heuristic 2

(B) database.py

```
def get_classes(query):
    …
    Use the database.
    …
    Return the classes.

def get_class_details(classid):
    …
    Use the database.
    …
    Return the class details.
```

Design: General Heuristic 3

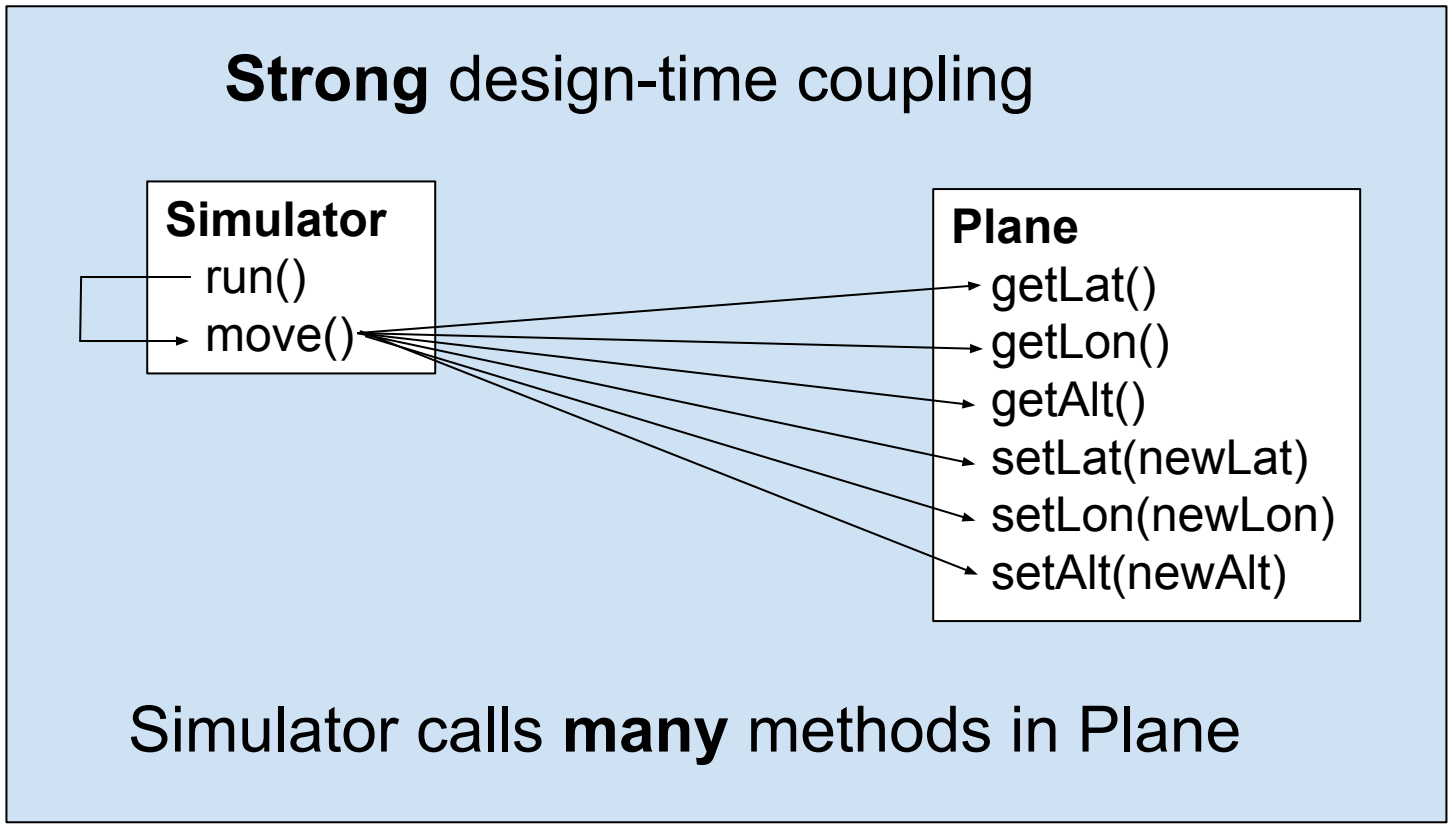
- Seek ***weak coupling*** among modules
  - Minimize interfaces
  - Encapsulate data
  - Hide design decisions

  - Empirically: **significant**

Design: General Heuristic 3a

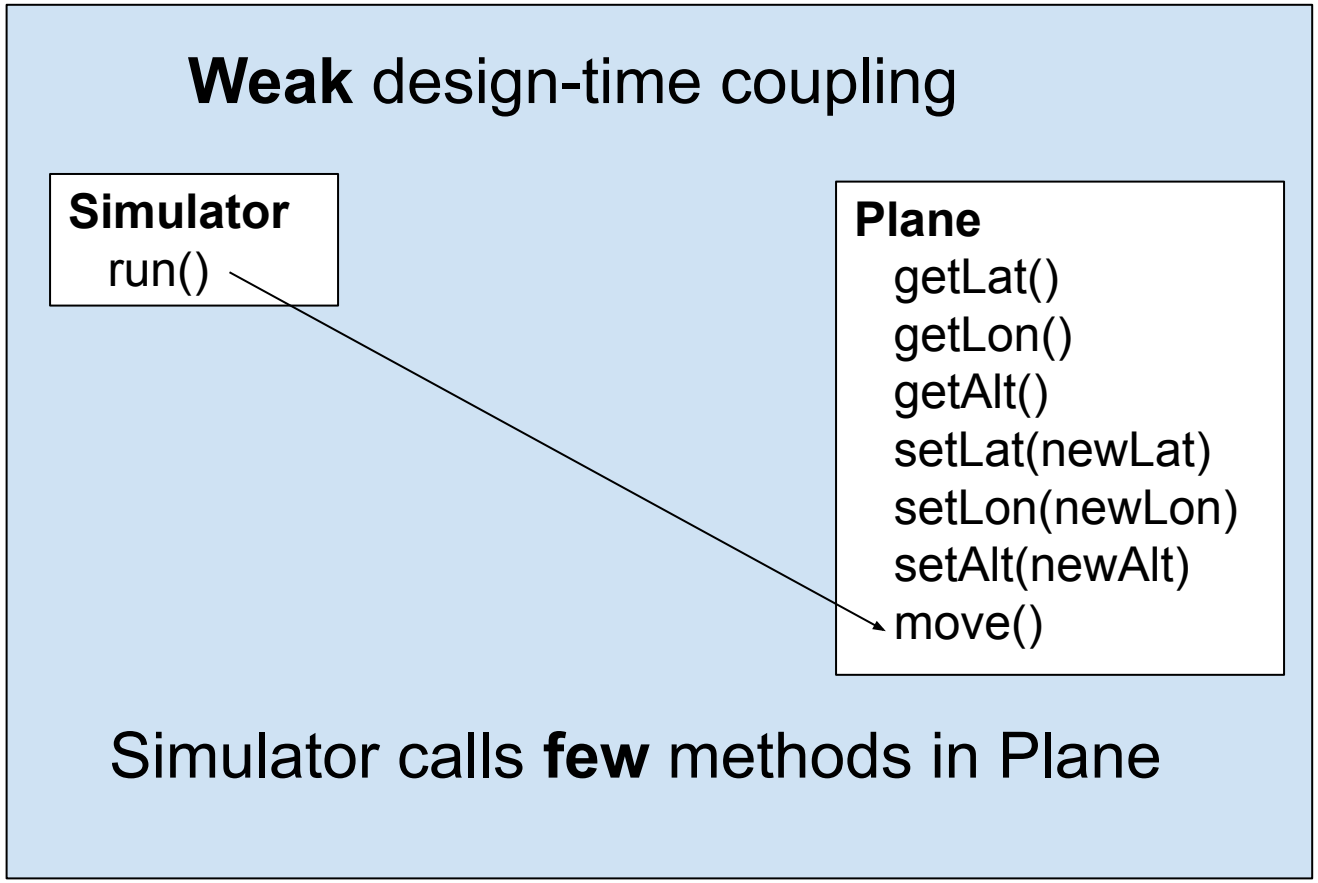
- Seek weak ***design-time coupling***

Design: General Heuristic 3a



No!

Design: General Heuristic 3a

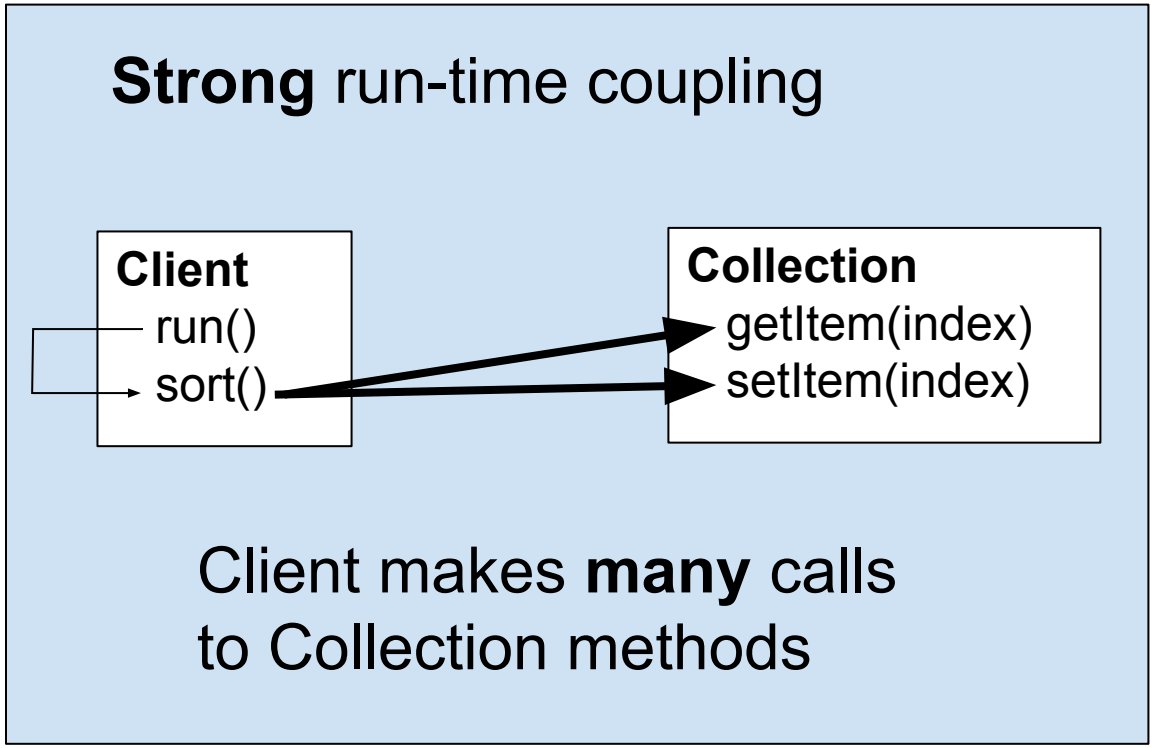


Yes!

Design: General Heuristic 3b

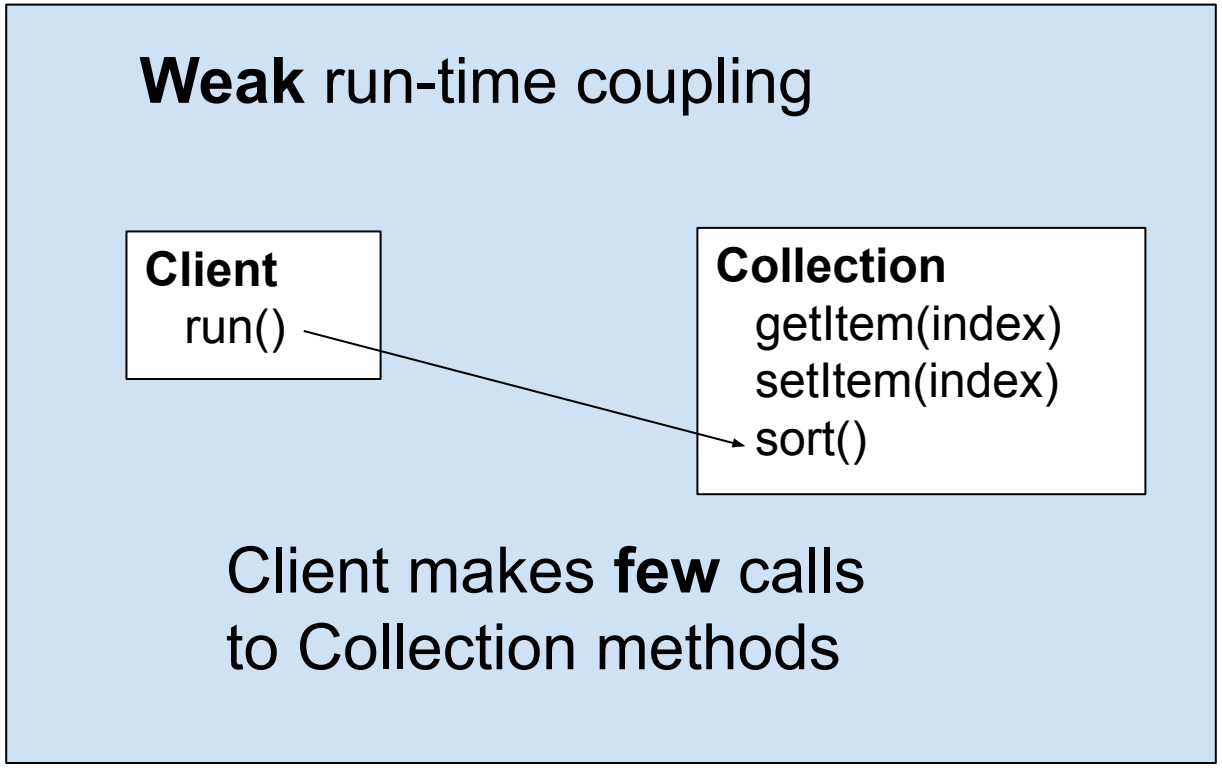
- Seek weak ***run-time coupling***

Design: General Heuristic 3b



No!

Design: General Heuristic 3b



Yes!

Design: General Heuristic 3

(A) database.py

```
class Database:
    …
    def connect():
        …
    def get_classes(query):
        …
    def disconnect():
        …
```

Design: General Heuristic 3

(B) database.py

```
def get_classes(query):
    Connect to the database.
    …
    Perform the query.
    …
    Disconnect from the database.
    …
    Return the classes.
```

Continued in
Software Engineering (Part 2)…