Software Engineering (Part 2)

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

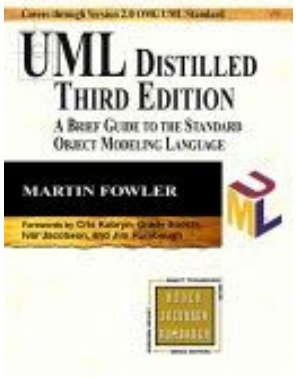
How to order the stages

Agenda

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

Design: OO

- ***Unified Modeling Language (UML)***



Grady Booch

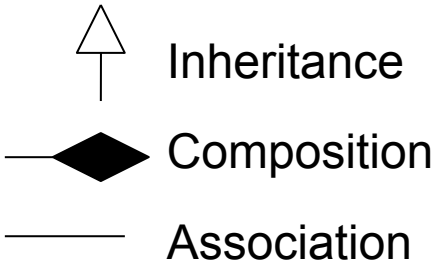
James Rumbauch

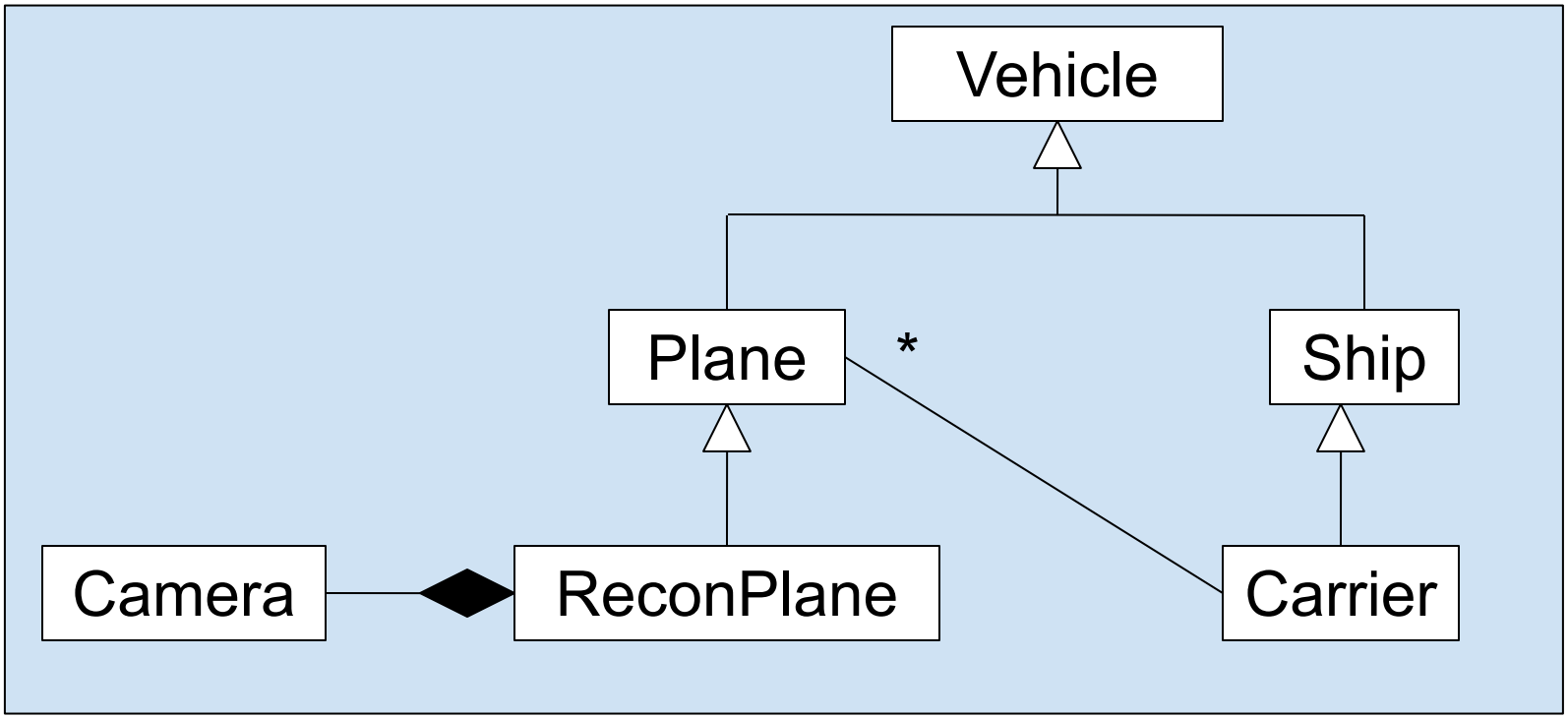
Ivar Jacobson

"The three amigos"

Design: OO

Unified Modeling Language (UML)





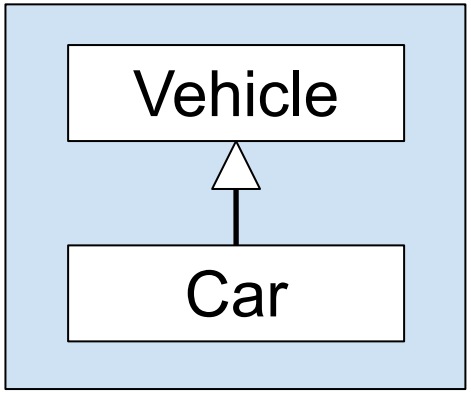
Design: OO Heuristic 1

- Use **inheritance** to model "**is a**"
  - Or "**is a kind of**"
- Use **composition** to model "**has a**"

Design: OO Heuristic 1

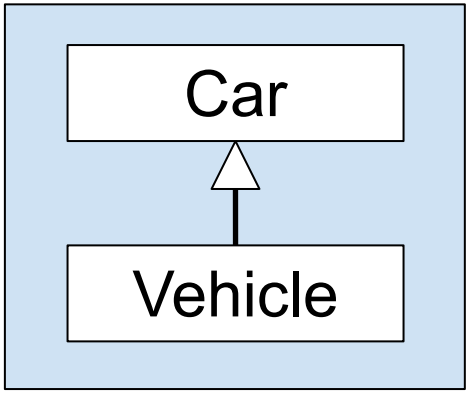
Which is proper?

A



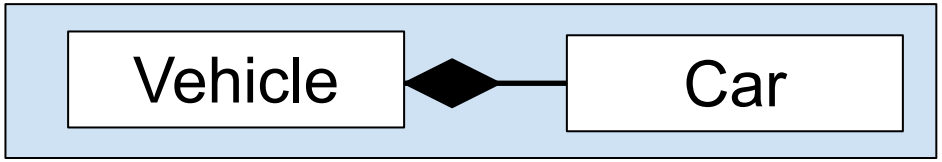
Car inherits from Vehicle

B



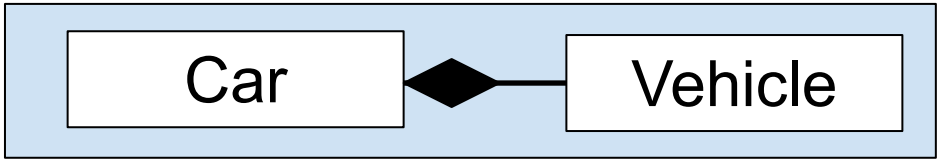
Vehicle inherits from Car

C



A Vehicle object is composed of a Car object

D

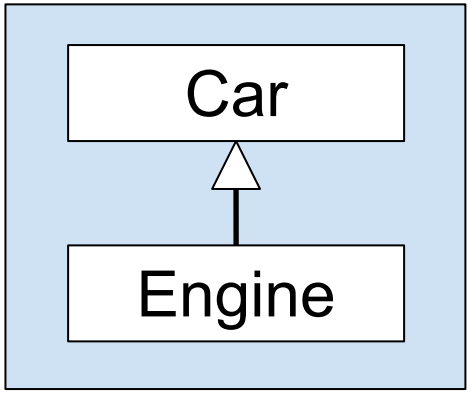


A Car object is composed of a Vehicle object

Design: OO Heuristic 1

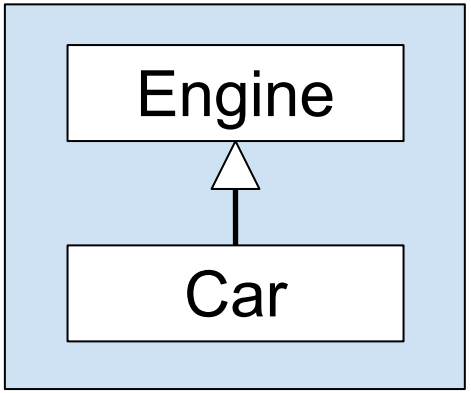
Which is proper?

A



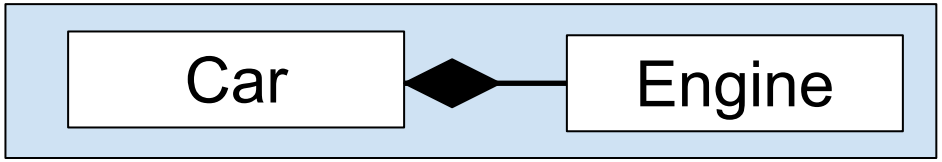
Engine inherits from Car

B



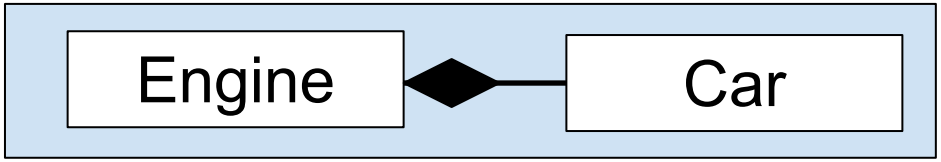
Car inherits from Engine

C



A Car object is composed of an Engine object

D



An Engine object is composed of a Car object

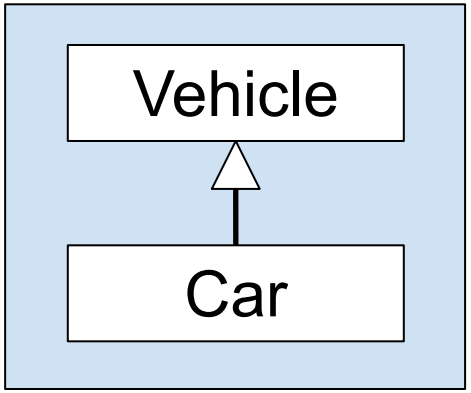
Design: OO Heuristic 2

- When designing inheritance hierarchies…
- Use the ***Liskov substitution principle***
  - Let p(t) be a property provable about objects t of type T. Then p(s) should be true for objects s of type S where S is a subtype of T

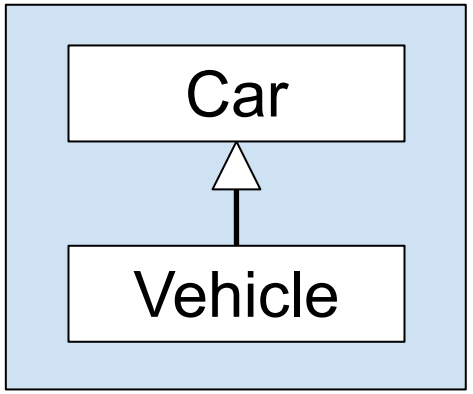
Barbara Liskov and Jeannette Wing.
"A behavioral notion of subtyping,"
ACM Transactions on Programming Languages and Systems,
Volume 16, Issue 6 (November 1994), pp. 1811 - 1841.

Design: OO Heuristic 2

- Use the Liskov sub principle (cont.)



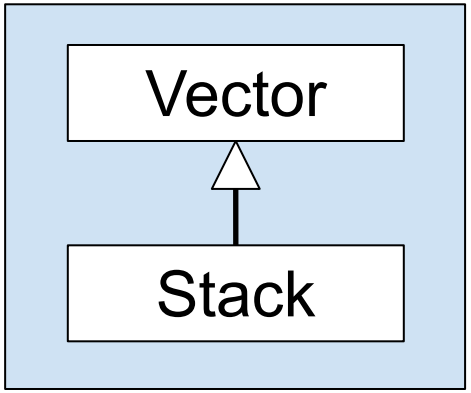
Suppose we have some code that uses a Vehicle object
Can we can replace the Vehicle object with a Car object and expect the code to work?
Yes!



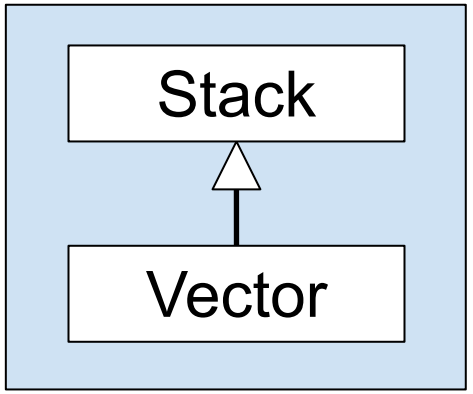
Suppose we have some code that uses a Car object
Can we can replace the Car object with a Vehicle object and expect the code to work? **No!**

Design: OO Heuristic 2

- Use the Liskov sub principle (cont.)



Suppose we have some code that uses a Vector object
Can we can replace the Vector object with a Stack object and expect the code to work? **No!**



Suppose we have some code that uses a Stack object
Can we can replace the Stack object with a Vector object and expect the code to work? **No!**

Design: OO Heuristic 3

- Favor composition over inheritance
  - Inheritance
    - ***White box reuse***
  - Composition:
    - ***Black box reuse*** => safer

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley. Reading, MA. 1995.

You've designed your application. What should your next step be?

Agenda

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

Implementation

- ***Implementation***
  - Alias **coding**

Implementation: Heuristic 1

- Consider **buying** instead of **building**
  - Does a commercial (or open source) system already exist that:
    - Fulfills (most) requirements?
    - (Approximately) implements your design?
  - If so, is it economically better to buy the commercial system instead of developing your own?

Implementation: Heuristic 2

- Avoid premature optimization
  - Clarity supersedes performance

Implementation: Heuristic 3

- Use generative AI – properly!
  - Use code composed by generative AI
    - But only if you understand it!!!

Aside: Vibe Coding

- ***Vibe coding***

Vibe coding describes a chatbot-based approach to creating software where the **developer describes a project or task to a large language model (LLM), which generates code** based on the prompt.

The developer does not review or edit the code, but solely uses tools and execution results to evaluate it and asks the LLM for improvements.

Unlike traditional AI-assisted coding or pair programming, **the human developer avoids examination of the code, accepts AI-suggested completions without human review, and focuses more on iterative experimentation than code correctness or structure.**

— https://en.wikipedia.org/wiki/Vibe_coding

You've implemented your system in code.
What's next?

Agenda

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

Debugging

- ***Debugging***
  - How can I fix the system?

Debugging: Heuristic 1

- Cycle frequently between implementation/coding and debugging
  - Wise:
    - Write a little, test a little
  - Unwise:
    - Write a lot, test a lot

Debugging: Heuristic 2

- Add internal tests
  - Validate parameter values
  - Check data structure invariants

Debugging: Heuristic 3

- Focus on recent changes
  - Use a version control system

Debugging: Heuristic 4

- Write many log messages
  - Maybe save them persistently

Debugging: Heuristic 5

- Use a debugger

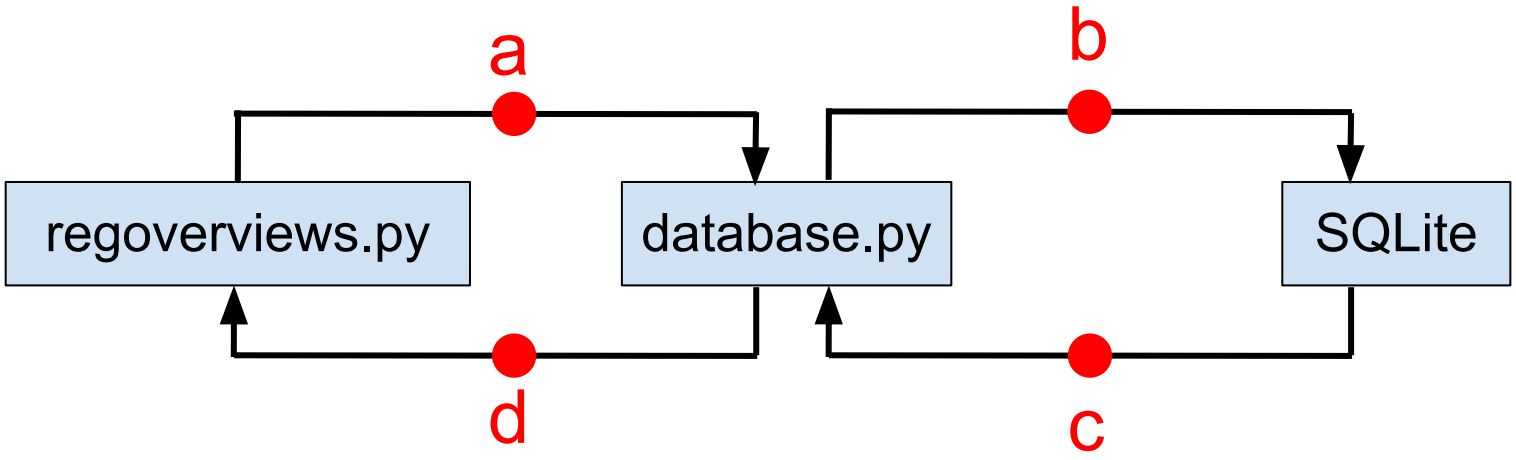
Language	Debugger	Reference
Assem lang	**gdb**	COS 217
C	**gdb**	COS 217
Python	**pdb**	Appendix of *The Python Language (Part 5)*
Java	**jdb**	https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html
JavaScript	**Chrome** **Firefox** **…**	https://www.w3schools.com/js/js_debugging.asp
JavaScript	**Node.js**	https://nodejs.org/api/debugger.html

Debugging: Heuristic 6

- Divide and conquer
  - Conduct the bug hunt in binary search fashion

Debugging: Heuristic 6

Asgt 1:



(a) Query data structure
(b) SQL statement
(c) Populated cursor
(d) List of classes

Debugging: Heuristic 7

- Use a **bug tracking system**
  - **Examples**: Issues (GitHub), Bugzilla (open source), Jira (Atlassian), Trello (Atlassian), Trac (open source), …
  - See https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems

You’re reasonably sure that your code is bug-free. What’s next?

Agenda

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

Testing

- **Debugging**: How can I **fix** the system?
- ***Testing***: How can I **break** the system?

Testing

- Testing taxonomy
  - Internal testing
  - External testing
    - White box
    - Black box
  - General strategies

Testing: Internal

- ***Internal testing***
  - Designing your code to test itself
  - Done by *programmers*

Testing: Internal

- Internal testing techniques
  - Check for function/method failures
  - Validate parameters
  - Check invariants
  - Leave testing code intact!!!

Testing: Internal

C: `assert` macro

```
assert(count >= 0);
```

Essentially same as:

```
if (count < 0)
{  fprintf(stderr,
      "assertion failed: (count >= 0),");
   fprintf(stderr,
      "function XXX, file YYY, line ZZZ.");
   exit(134);
}
```

Asserts are **enabled** by default; to **disable** asserts:

```
gcc -D NDEBUG somefile.c
```

Testing: Internal

Python: `assert` statement

```
assert count >= 0, 'count is < 0'
```

Essentially same as:

```
if count < 0:
    raise AssertionError('count is < 0')
```

Asserts are **enabled** by default; to **disable** asserts:

```
python -O somefile.py
```

Testing: Internal

Java: `assert` statement (since JDK 1.4)

```
assert count >= 0 : "count is < 0";
```

Essentially same as:

```
if (count < 0)
   throw new AssertionError("count is < 0");
```

Asserts are **disabled** by default; to **enable** asserts:

```
java -ea SomeFile.java
```

Testing: Internal

JavaScript (browsers):
`console.assert` function

```
console.assert(count >= 0, 'count is < 0');
```

Essentially same as:

```
if (count < 0)
   console.error('count is < 0');
```

Cannot be disabled???

Testing: Internal

JavaScript (Node.js): `assert` function

```
const assert = require('assert');
…
assert(count >= 0);
```

Essentially same as:

```
if (count < 0)
   throw new Error(
      'The expression evaluated to a falsy value');
```

Cannot be disabled!

Testing: Internal

- Assert controversy: enable or disable asserts in production code?

Testing: External

- ***External testing***
  - Designing code or data to test your code

Testing: External

- ***White box*** **external testing**
  - External testing with knowledge of structure of tested code
  - Done by **programmers**

Testing: External

- White box external testing techniques
  - ***Statement (coverage) testing***
    - Testing to make sure each **statement** is executed at least once
  - ***Path testing***
    - Testing to make sure each **logical path** is followed at least once

Testing: External

- White box external testing techniques
  - ***Boundary (corner case) testing***
    - Testing with input values at, just below, and just above limits of input domain
    - Testing with input values causing output values to be at, just below, and just above the limits of the output domain

Glossary of Computerized System and Software Development Terminology

Testing: External

- ***Black box*** **external testing**
  - External testing without knowledge of structure of tested code
  - Done by *quality assurance (QA) engineers*

Testing: External

- Black box external testing techniques
  - ***Use case testing***
    - Testing driven by use cases developed during design
  - ***Stress testing***
    - Testing with a large quantity of data
    - Testing with a large variety of (random?) data

Testing: General Strategies

- General testing strategies
  - Test incrementally
    - Use scaffolds and stubs
    - Do ***regression testing***
  - Let debugging drive testing
    - Reactive mode
    - Proactive mode: do ***fault injection***

Testing: General Strategies

- **General testing strategies**
  - Automate the testing
    - To test your **programs**: create **scripts**
    - To test your **modules**: create software **clients**
    - Compare implementations when possible

Testing: General Strategies

Kinds of automated testing:

	Unit Testing	System/Integration Testing
What	Test **modules** (classes, functions, methods)	Test **programs**
When	Often compose tests for module X **before** composing X	Typically compose tests for program X **after** composing X
Who	Often performed by a **single programmer**, or a small group	Often performed by a **QA organization**

Testing: Summary

- Testing taxonomy
  - Internal testing
  - External testing
    - White box
    - Black box
  - General strategies

Continued in
Software Engineering (Part 3)…