Chapter 4
Design Principles I
Correctness and Robustness

Key Concept: $\rightarrow$ Correctness $\leftarrow$

Goal: That each artifact satisfies designated requirements, and that together they satisfy all of the application’s requirements.

Adapted from Software Design: From Programming to Architecture by Eric J. Braude (Wiley 2003), with permission.
Key Concept: Correctness by Informal Methods

Simplify and modularize designs until they convince.

Invariants for Class Automobile

-- with variables mileage, VehicleID, value, originalPrice, and type:

1) mileage > 0
2) mileage < 100000
3) vehicleID has at least 8 characters
4) value >= -300
   ($300 is the disposal cost of a worthless automobile)
5) originalPrice >= 0
6) (type == "REGULAR" && value <= originalPrice) ||
   (type == "VINTAGE" && value >= originalPrice)

Introducing Interfaces

1 of 2

Shipment

setVehicle()
perishable()
getWidth()
printRoute()
describeType()
getLength()
getDuration()
setType()

2 of 2

Shipment

setVehicle()
perishable()
getWidth()
printRoute()
describeType()
getLength()
getDuration()
setType()
Key Concept: Interfaces

→ collections of function prototypes: Make designs more understandable.

Domain vs. Non-Domain Classes

- **Domain classes**: Particular to the application
  - Examples: BankCustomer, BankTransaction, Teller
  - Typically not GUI classes
  - Sufficient to classify all requirements (see chapter xx)
- **Non-Domain classes**: Generic
  - Examples: abstract classes, utility classes
  - Arise from design and implementation considerations
Alternative Modularizations

Alternative 1
- Mechanics
- Position
- Ground Control
- OnBoardNavigation

Alternative 2
- Control
- Trajectory
- Weather

Application tracking trajectory of rocket carrying orbit-bound satellite into position

Improving Robustness: Sources of Errors

Protection from faulty Input
- User input
- Input, not from user
  - Data communication
  - Function calls made by other applications

Protection from developer error
- Faulty design
- Faulty implementation

Constraints on Parameters

Example:
```java
int computeArea(int aLength, int aBreadth) {
...
}
```

- Capture parameter constraints in classes if feasible
- Integrate `computeArea( RectangleDimension aRectangleDimension )`
- Specify all parameter constraints in method comments
  - aLength >= 0 and aBreadth >= 0 and aLength >= aBreadth
- Callers obey explicit requirements on parameters
  - Problem is method programmers have no control over callers
- Check constraints first within the method code
  - `if (aLength <= 0) ...`
    - Throw exception if this is a predictable occurrence
    - Otherwise abort if possible
    - Otherwise return default if it makes sense in context
    - And generate warning or log to a file

Key Concept: → Robustness ←

-- is promoted by verifying data values before using them.
Wrapping Parameters

Replace

```java
int computeArea( int aLength, int aBreadth )
{
}
```

with

```java
int computeArea( Rectangle aRectangle )
{
}
```

-- where
class Rectangle
{
    // ...
    Rectangle( int aLength, int aBreadth )
    {
        if( aLength > 0 ) this.length = aLength;
        else .... // ...
    }
}

Key Concept: Robustness

→ is promoted by enforcing intentions.

How Much Design Detail Before Initial Coding?

Video Store Application: Sufficient Classes?

<table>
<thead>
<tr>
<th>Type of application</th>
<th>Very simple</th>
<th>Very complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inexperienced designer</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>Experienced designer</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Recommended % of design detail before starting to code:

- Video
- CheckOutDurationDisplay
- Customer
- CheckOutDisplay
- BarCodeReader
- RegisterNewVideoDisplay
Summary of This Chapter

- **Correctness of a Design or Code**
  - Supports the requirements
  - In general, many correct designs exist

- **Robustness of a Design or Code**
  - Absorbs errors
    - -- of the user
    - -- of developers

Adapted from *Software Design: From Programming to Architecture* by Eric J. Braude (Wiley 2003), with permission.