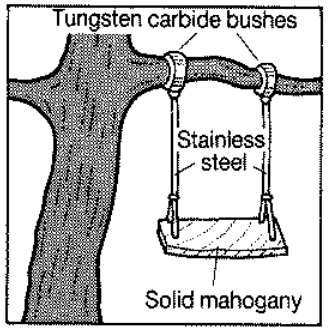
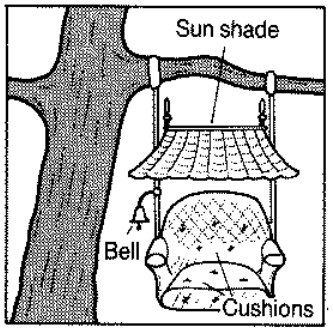
DETAILED SOFTWARE DESIGN[1]

Kevin P Dyer

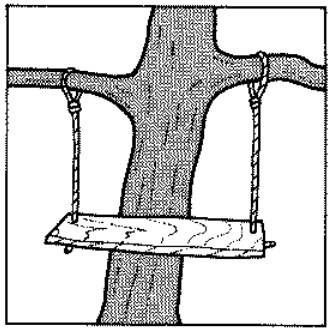
1 These slides are based on material from Wikipedia.



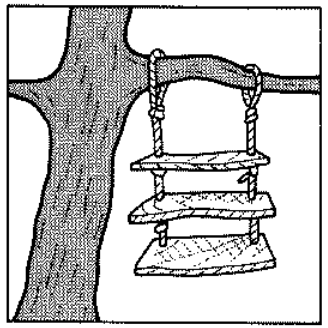
What Product Marketing specified



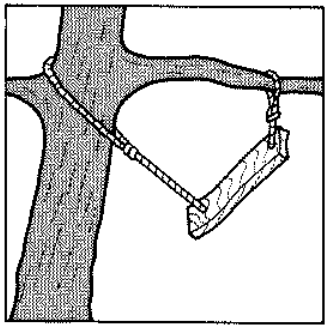
What the salesman promised



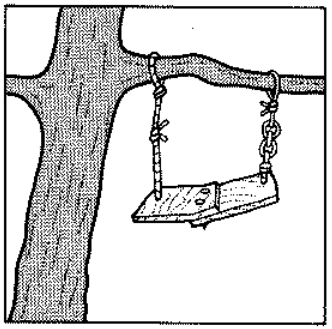
Design group's initial design



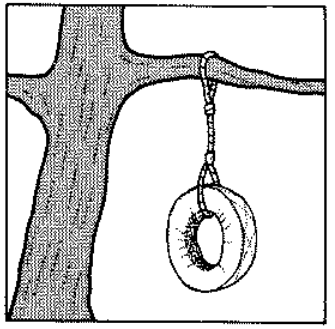
Corp. Product Architecture's modified design



Pre-release version



General release version



What the customer actually wanted

Detailed Software Design

- New Code vs. Legacy Code
- Internal vs. External

The Basics: Coding Standards

- Seriously, style matters
- Check your style with your editor
- Alternatively, use one of these tools (for C++):
  - Uncrustify
  - Astyle (Artistic Style)
  - Polystyle
  - SQCBW
  - GC! GreatCode
  - Pork
  - Vera++
  - Bcpp (C++ Beautifier)
  - KWStyle

Consistency is everything

- Make a decision about style and convention

- Examples:
  - **`_memberVariables`** `vs` **`localVariables`**
  - **`CONSTANTS`** `vs` **`variables`**
  - **`pPointers`** `vs` **`variables`**

Consistency is everything

```
<?php
$dirtyZipCode = $_GET[‘zipcode’];
$cleanZipCode = sanitizeInput($dirtyZipCode);

// a few hundred lines later…

<p><?=$dirtyZipCode?></p> // very very bad!
```

New Functionality

- How do you translate requirements or a high level design into real code?

The User Story

- A user story is one or more sentences in the everyday or business language of the user that captures what the user wants to achieve. … Each user story is limited, so it fits on a small paper note card—usually a 3×5 inches card—to ensure that it does not grow too large. The user stories should be written by the customers for a software project and are their main instrument to influence the development of the software.

The User Story

- User stories are a quick way of handling customer requirements without having to elaborate vast formalized requirement documents and without performing overloaded administrative tasks related to maintaining them. The intention of the user story is to be able to respond faster and with less overhead to rapidly changing real-world requirements.

The User Story

- Example template:
  - "As a <role>, I want <goal/desire> so that <benefit>"

- Example stories:
  - As a customer representative, I want to search for my customers by their first and last name.
  - Upon closing the application, the user is prompted to save (when ANYTHING has changed in the data since the last save!).

So…

- …you have a user story. Now what?

Create Prototypes

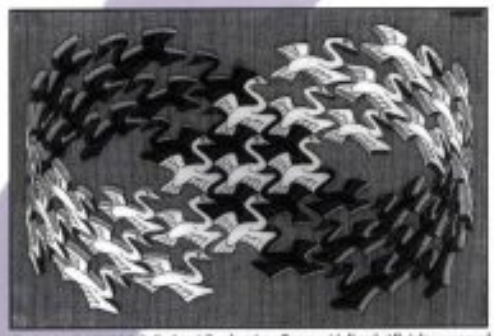
- Don’t be afraid to throw away your code and rewrite it

Copyrighted Material

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

Copyrighted Material

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Model-View-Controller

- The **view** renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. A viewport typically has a one to one correspondence with a display surface and knows how to render to it.
- The **controller** receives input and initiates a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input.
- The **model** is used to manage information and notify observers when that information changes. The model is the domain-specific representation of the data upon which the application operates. … When a model changes its state, it notifies its associated views so they can be refreshed.

Model-View-Controller

- Real World Uses
  - Web Applications
  - Mobile Applications
  - Desktop Applications
  - Almost anything with a GUI…
- When is MVC not right?
  - Video Games
  - Video Players
  - Applications that require high performance

Command Pattern

- In object-oriented programming, the **command** pattern is a design pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time. This information includes the method name, the object that owns the method and values for the method parameters.

Command Pattern

- Uses
  - Multi-level undo
  - Transactions
  - Progress bars
  - Wizards
  - GUI buttons/actions
  - Mobile applications
- You can even serialize and transmit Commands across the network

Singleton Pattern

- In software engineering, the **singleton** pattern is a design pattern used to implement the mathematical concept of a singleton, by restricting the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects (say, five).

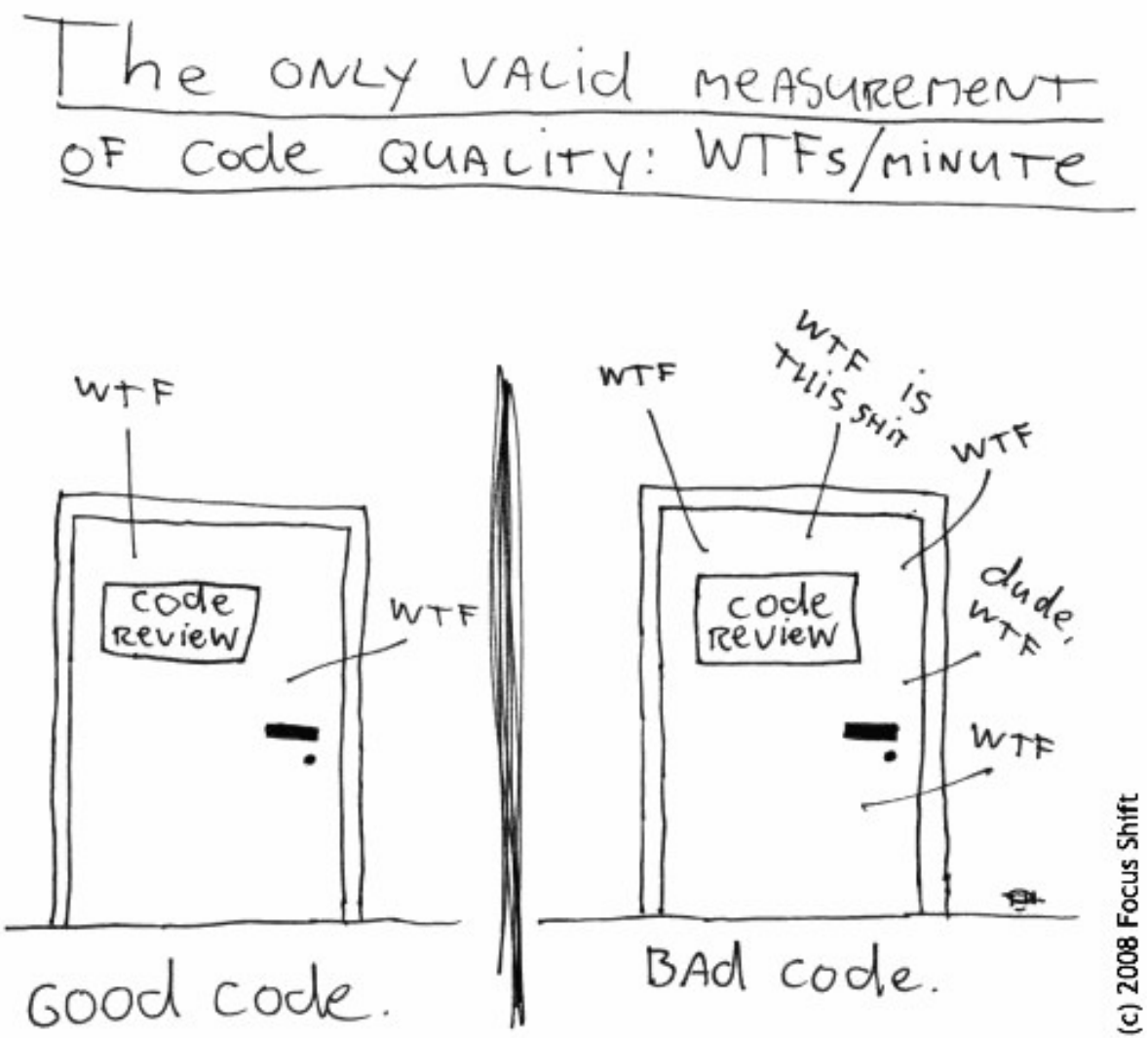
Singleton Pattern

```
// Header file (.h)
class Singleton
{
    private:
        Singleton() {}
        ~Singleton() {}
        Singleton(const Singleton &);
        Singleton & operator=(const Singleton &);

    public:
        static Singleton &getInstance();
};


// Source file (.cpp)
Singleton& Singleton::getInstance()
{
    static Singleton instance;
    return instance;
}
```

Singleton Pattern

- Pros
  - Restricts the number of instances of a specific resource
  - May save memory/resources if used in the right conditions
- Cons
  - Makes unit testing much harder, since you now have to consider the system state
  - A lock mechanism must be implemented in multi-threaded programs

What about legacy code?



What about legacy code?

- Interfaces are hard to get right
- Poor decisions linger for decades
- As a software engineer this will waste a lot of your time

Legacy Code: The minutiae

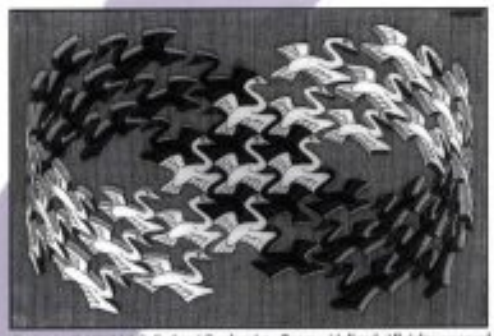
```
// oci8 PHP driver 1.4.1
// oci8_statement.c:1452
…
bindp->bind = NULL;
bindp->zval = var;
bindp->array.type = type;
bindp->type = 0; // added in version 1.4.2
…
```

Copyrighted Material

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

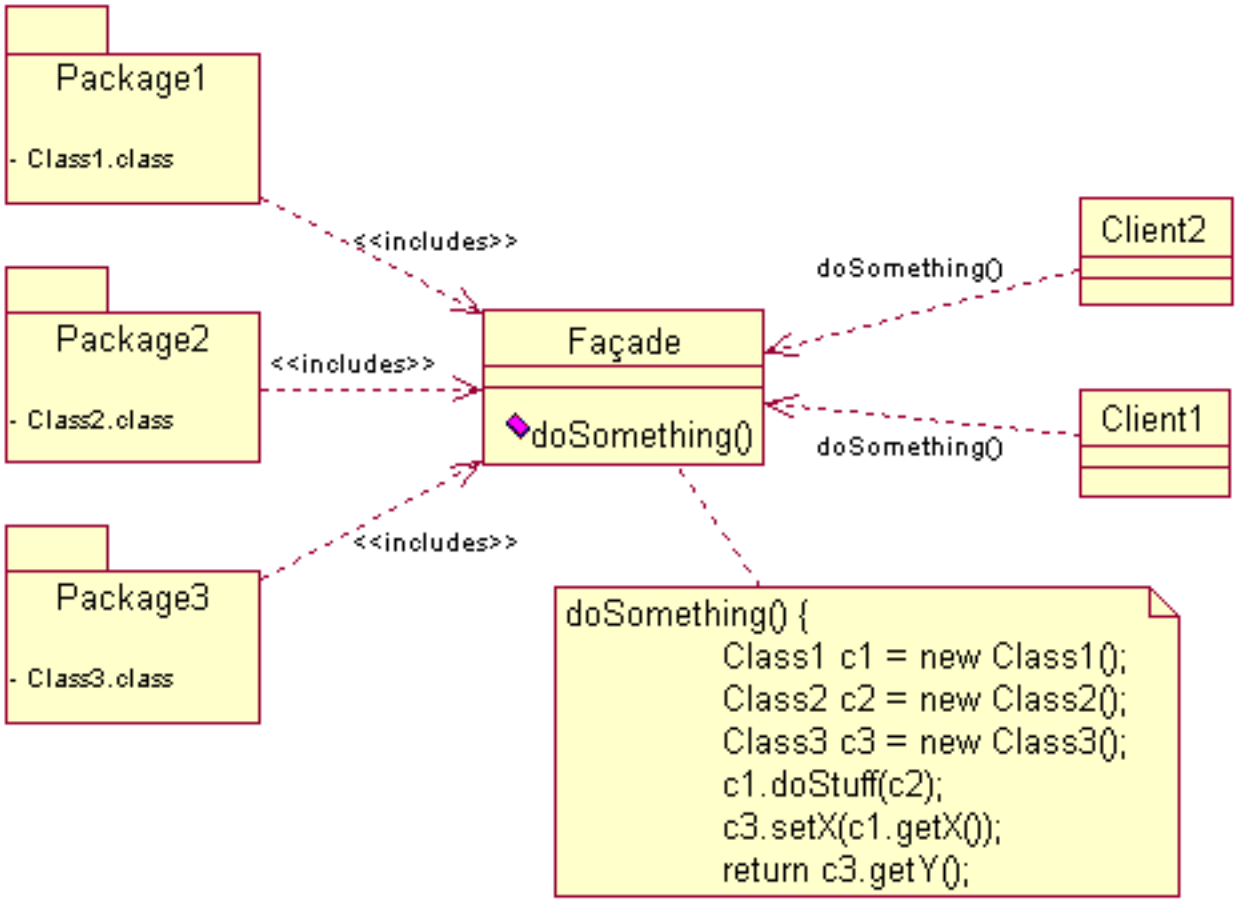
Copyrighted Material

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Façade Pattern

- Make a software library easier to use, understand and test, since the facade has convenient methods for common tasks;
- Make code that uses the library more readable, for the same reason;
- Reduce dependencies of outside code on the inner workings of a library, since most code uses the facade, thus allowing more flexibility in developing the system;
- Wrap a poorly-designed collection of APIs with a single well-designed API (as per task needs).

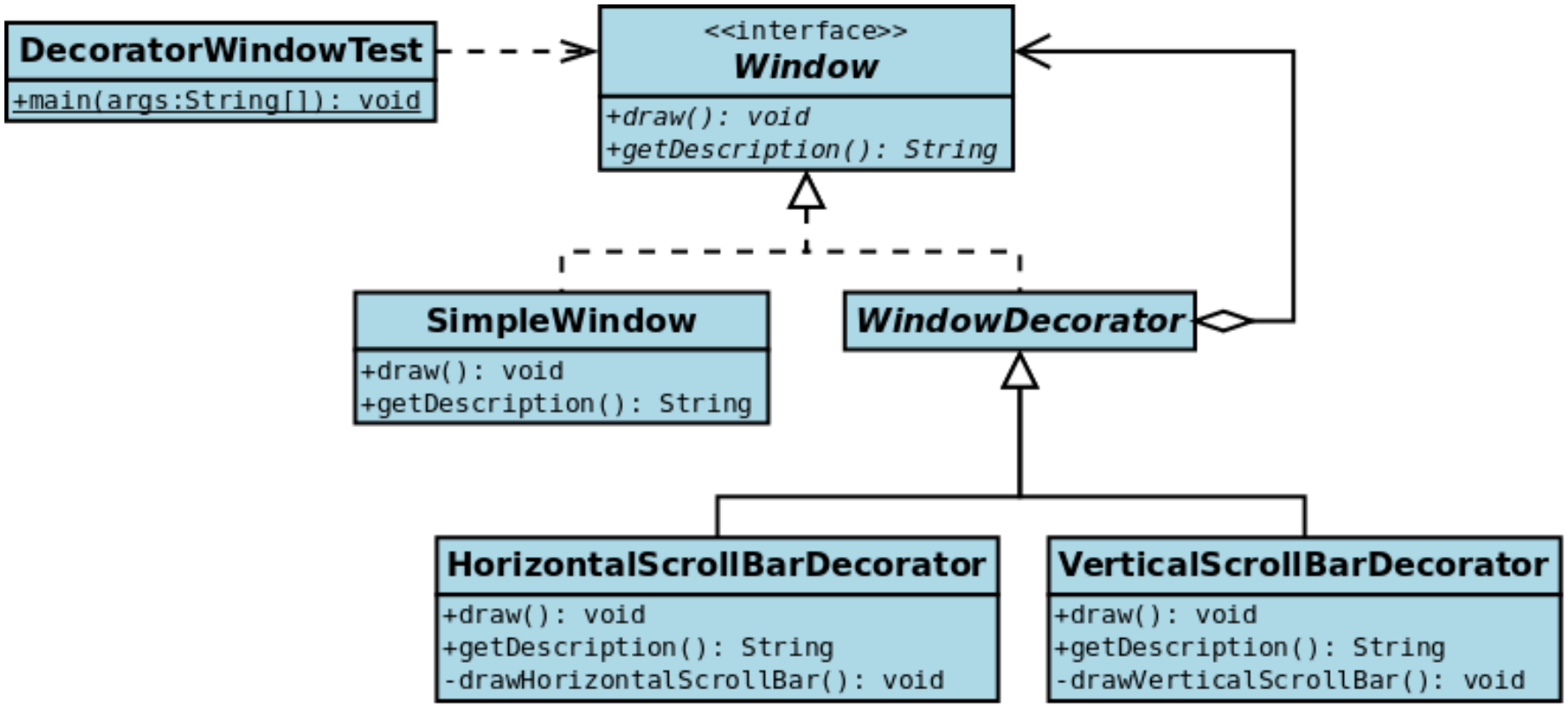
Façade Pattern



Decorator Pattern

- The decorator pattern is a design pattern that allows new/ additional behaviour to be added to an existing object dynamically.
- This pattern is designed so that multiple decorators can be stacked on top of each other, each time adding a new functionality to the overridden method(s).
- The decorator pattern is an alternative to subclassing. Subclassing adds behavior at compile time, and the change affects all instances of the original class; decorating can provide new behavior at runtime for individual objects.

Decorator Pattern



A RESTful Web Service

- Representational State Transfer by Roy Fielding (2000)

- Non-RESTful (direct access to a resource/script)
  - **read**: /v1/GetContent.ashx?id=[id]&format=[xml|json]
  - **delete**: /v1/DeleteContent.ashx?id=[id]&format=[xml|json]

- RESTful (a façade, which redirects the request to a resource)
  - **read**: /catalog/[id]
  - **read**: /catalog/[format]/[id]
  - **delete**: /catalog/delete/[id]
  - **delete**: /catalog/[format]/delete/[id]

Someone will actually use your code

- Consider how your interfaces, and logic deep in your code, will alter the appearance and behaviour of your program

In Practice: What works?

- Prototyping
- Using verbs/nouns for classes, variables, and external interfaces (keep a thesaurus at your desk!)
- Pair programming
- Code reviews
- Making tons of mistakes, but learning from them

In Practice: What doesn't work?

- Don't work on a project unless it excites you. Otherwise, it is too exhausting to care about the details.
- Don't start coding without well defined short-term goals (remember user stories!)
- Don't ignore high level designs or coding standards simply because they are tedious or boring

- “adding manpower to a late software project makes it later”