

OOAD

"There is no silver bullet."
—Fred P. Brooks, Jr.

OO Design is an Art



4-Feb-20 7:50 PM

2

OOAD books and classes can only enhance the innate talents of individuals and make them the best OO engineers they can be.

The education of artists is not focused on technique, process or tools.

The majority of an art education combines ideas, history, appreciation, experience and constructive criticism.

Different

- Advocacy of a local rather than global focus
- Practitioners of rapid prototyping instead of structured development



4-Feb-20 7:50 PM



3

Collaborative rather than imperial management style

OOD -

Commitment to design based on coordination and cooperation rather than control

Driven by internal capabilities instead of conforming to external procedures.

Decide Classes

- How to decide what should be a class?
 - Nouns
 - Value is a group of items.
 - Functions associated with an item.



4-Feb-20 7:50 PM

4

Q022: telLocalNumber

Q01: IsSameString

Class should be highly cohesive

A Class should have a single well focused purpose.

A Class should do one thing and do it well.

Find the classes

- Selling soft drinks on a vending machine.
 - Software will control the functions of the vending machine.
 - First the user enters some money. The machine displays the money entered so far. The products that can be bought light up. The user chooses his option. The vending machine dispenses the product and the change.

4-Feb-20 7:50 PM

5

Answer: VendingMachine, MoneyBox, Screen, PriceList, SoftDrink, SoftDrinkList, SoftDrinkDispenser, Safe

Metaphor

- Object is like a person.
 - Both are specialists and lazy
 - Both don't like to be micromanaged.
 - Both take responsibilities

4-Feb-20 7:50 PM

6

Distributed cooperation and communication must replace hierarchical centralized control as an organizational paradigm. E.g. A traffic signal and cars.


Like people, software objects are specialists and lazy. A consequence of both these facts is the distribution of work across a group of objects. Take the job of adding a sentence to a page in a book. Granted, it might be quite proper to ask the book, "Please replace the sentence on page 58 with the following." (The book object is kind of a spokesperson for all the objects that make up the book.) It would be quite improper, however, to expect the book itself to do the work assigned. If the book were to do that kind of work, it would have to know everything relevant about each page and page type that it might contain and how making a simple change might alter the appearance and the abilities of the page object. Plus the page might be offended if the book attempted to meddle with its internals.

The task is too hard (lazy object) and not the book's job (specialist object), so it delegates—merely passes to the page object named #58 the requested change. It's the page object's responsibility to carry out the task. And it too might delegate any part of it that is hard—to a string object perhaps.

Objects, like the people we metaphorically equate them to, can work independently and concurrently on large-scale tasks, requiring only general coordination. When we ask an object collective to perform a task, it's important that we avoid micromanagement by imposing explicit control structures on those objects. You don't like to work for a boss who doesn't trust you and allow you to do your job, so why should your software objects put up with similar abuse?

Metaphors

- Software is a Theater, Programmer is a director
- Ants, not Autocrats



4-Feb-20 7:50 PM
7

Q57srp – Students

Q58srp – Servers

Hierarchical and centralized control is the anathema in OO. Complex systems are characterized by simple elements, acting on local knowledge with local rules, give rise to complicated patterned behavior. Object can inherit like a person.

For example, suppose an airplane object has a responsibility to report its location. This is a hard task because the location is constantly changing; a location is a composite structure (latitude, longitude, altitude, direction, speed, and vector); the values of each part of that structure come from a different source; and someone has to remember who asked for the location and make sure it gets back to them in a timely fashion. If the task is broken up so that

- The airplane actually returns a location object to whoever asked for it after appending its ID to the location so that there is no confusion about who is where. (We cannot assume that our airplane is the only one reporting its location at any one time.)
- An instrument cluster keeps track of the instruments that must be asked for their current values and knows how to ask each one in turn for its value (a collection iterating across its contents).

- An instrument merely reports its current value.
- A location object collects and returns a set of label:value pairs (altitude:15,000 ft.).

None of the objects do anything particularly difficult, and yet collectively they solve a complicated problem that would be very hard for any one of them to accomplish individually.

Guidelines

- DRY



4-Feb-20 7:50 PM

8

Q021: BookRentals; Q11: BookRental; Q33 – SurveyData; Q37 – FOC, TT

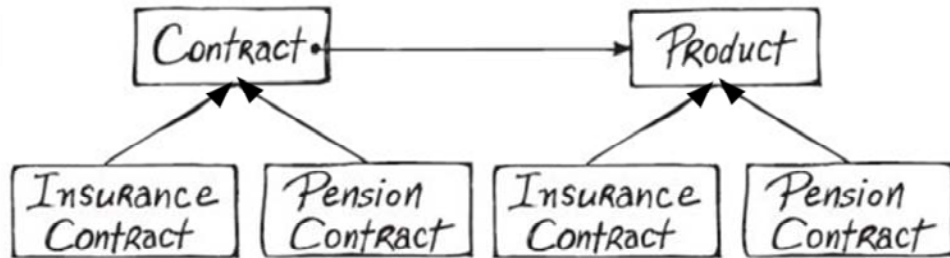
Q10 – JButton – Only for Java.

Q34 – replace; Q35 – BookRental

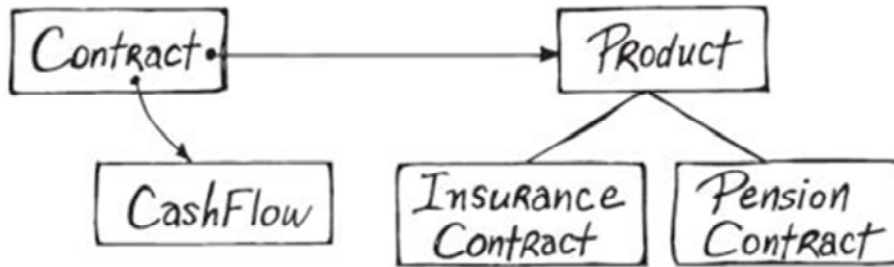
Q70 – Array Scan, Java only.

There are three numbers in software: 0, 1, and infinity. 0 represents the things we do not do in a system (we do those for free). 1 represents the things we do once and only once. But at the moment we do something twice, we should treat it as infinitely many and create cohesive services that allow it to be reused.

Improve



Eliminate Parallel Hierarchies



Reduce Coupling - Problem

```
Server s = new Server();
```

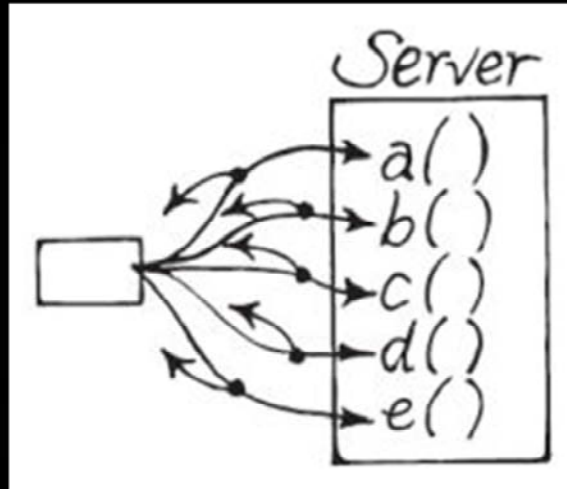
```
s.a(this);
```

```
s.b(this);
```

```
s.c(this);
```

```
s.d(this);
```

```
s.e(this);
```



Reduce Coupling - Solution

```
Server s = new Server(this);
```

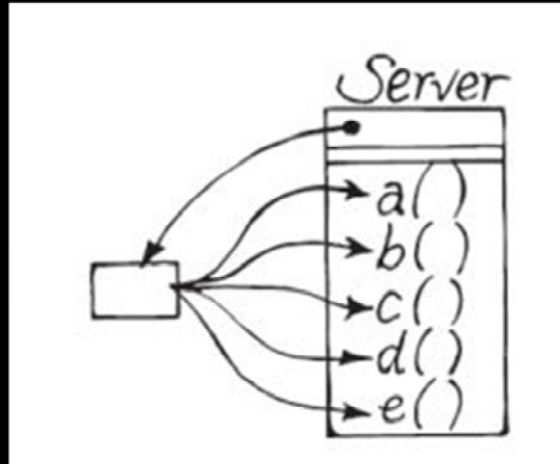
```
s.a();
```

```
s.b();
```

```
s.c();
```

```
s.d();
```

```
s.e();
```



4-Feb-20 7:50 PM

12

Better still: use Observer design pattern

Classes in an Application

- Many simple classes means that each class
 - encapsulates less of overall system intelligence
 - is more reusable
 - is easier to implement
- A few complex classes means that each class
 - encapsulates a large portion of system intelligence
 - is less likely to be reusable
 - is more difficult to implement

4-Feb-20 7:50 PM

13

Lots of little pieces

Classes are cohesive

Methods do only one thing.

Guidelines

- A class should have less than 50 lines
- Most functions should be less than or equal to 5 lines.
 - A function taking more than 3 arguments should be rare and justified specially.

4-Feb-20 7:50 PM

14

Note: On a Home PC - 1 Million function calls take 8 milliseconds and 1 Million objects are created in 23 milliseconds

Some Real Examples

Tool	Files	Lines/file (avg)	LOC/file (avg)
JUnit	88	71	39
Hibernate	1063	90	72
Eclipse	14,620	153	106
DomainObjects for .NET	422	164	98
Compiere ERP & CRM	1191	163	114
Hsqldb	290	503	198

?

4-Feb-20 7:50 PM

15

Q56srp – Restaurants

Q59srp - Customers

Single Responsibility Principle

- A class should have only one reason to change.

- Bad

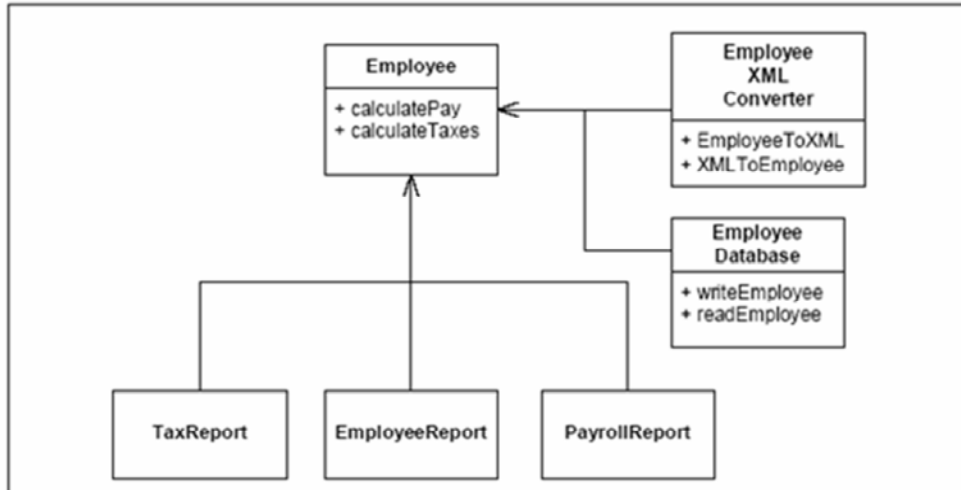
```
public class Employee {  
    public double calculatePay();  
    public double calculateTaxes();  
    public void writeToDisk();  
    public void readFromDisk();  
    public String createXML();  
    public void parseXML(String xml);  
    public void displayOnEmployeeReport(  
        PrintStream stream);  
    public void displayOnPayrollReport(  
        PrintStream stream);  
    public void displayOnTaxReport(  
        PrintStream stream);  
}
```

4-Feb-20 7:50 PM

16

An item such as a class should just have one responsibility and solve that responsibility well. If a class is responsible both for presentation and data access, that's a good example of a class breaking SRP.

SRP implemented



Popular API

- Java
 - `java.xml.datatype.XMLGregorianCalendar` has both date and time
 - `java.util.concurrent.TimeUnit` is an enum for various units and also converts from one form to another.
- C#
 - `DateTime`



4-Feb-20 7:50 PM

18

Q50 – Account

Q51 – Department

Q55 – processReport1

Functional Programming

- Why are functional languages like Clojure, Scala and F# getting popular?
 - Languages like Java can get some functional features by proper design, using libraries like “Functional Java” and “Akka”.

4-Feb-20 7:50 PM

19

Functional language advantages:

functions are first class citizens

functions are side-effect free

declarative. Uses recursion to reduce the size of the problem.

Pure Functions

- A function that computes output based on parameters passed only.
 - No access to instance variables or global variables
- The function does not alter any input parameters.



Pure Function

```
public static int Add(int a, int b)
{
    return a+b;
}
```

No hidden dependencies, No side effects, Thread-safe.

?

4-Feb-20 7:50 PM

21

Also known as side-effect free functions.

Q60.

Q61

Pipe Simple Commands

- Imagine a CSV file book.csv

ISBN,Title,Price

123,Java,500

234,C#,700

345,OOAD,600

- To get sum of all prices

```
sed 1d book.csv | cut -d, -f 3 | awk '{s+=$1} END {print s}'
```

- To get sum of 3 costliest books

```
sed 1d book.csv | cut -d, -f 3 | sort -rn | head -n 3 |  
awk '{s+=$1} END {print s}'
```

?

4-Feb-20 7:50 PM

22

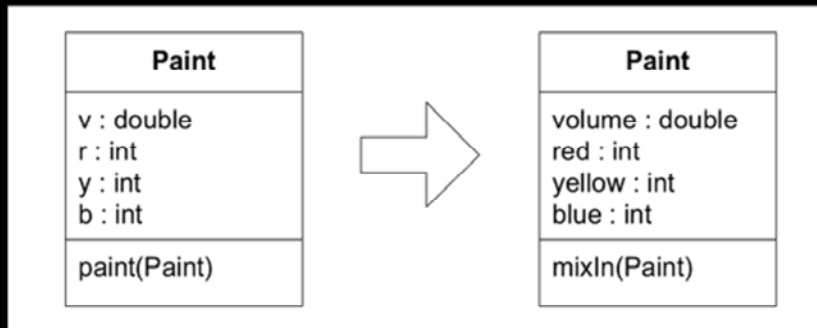
This is Linux philosophy. Do a complex process by performing simple tasks and then piping them.

Instead of building a huge complex class, build simple classes. Pipe output of one class to another. This will result in more flexible coding. More portions will be reusable.

Q62

Rule: Don't abbreviate

- Code should be self documenting.
 - Tools: Eclipse, Codepro, PMD, PMD-CPD, Findbugs, Checkstyle, Cobertura.



4-Feb-20 7:50 PM

23

Public API's have to be documented i.e. every class, function, interface, exceptions.
Mutable objects that can / cannot be modified.

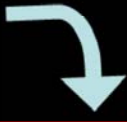
This is possible by choosing the right variable and function names.

Comments are secondary because they tend to lie

C#: FxCop, Simian, Ncover, NDepend for cyclomatic complexity.

C++: Simian or PMD CPD for duplication, coverity for source code analysis


```
public List<int[]> getThem() {  
    List<int[]> list1 =  
        new ArrayList<int[]>();  
    for (int[] x : theList)  
        if (x[0] == 4)  
            list1.add(x);  
    return list1;  
}  
public List<Cell> getFlaggedCells() {  
    List<Cell> flaggedCells =  
        new ArrayList<Cell>();  
    for (Cell cell : gameBoard)  
        if (cell.isFlagged())  
            flaggedCells.add(cell);  
    return flaggedCells;  
}
```



4-Feb-2017 10:01 AM

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
};
```



```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;;  
    private static final String  
        RECORD_ID = "102";  
    /* ... */  
};
```

Guidelines

- Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser.
 - Avoid words like Manager, Processor, Data, or Info in the name of a class.
 - A class name should not be a verb.
- Methods should have verb or verb phrase names like postPayment, deletePage, or save.

Code should be readable

- Any fool can write code that a computer can understand. Good programmers write code that humans can understand. – Martin Fowler

- `Calendar c=Calendar.getInstance();`
 `c.set(2005,Calendar.NOVEMBER, 20);`
 `Date t = c.getTime();` OR
- `Date t = november(20, 2005);`
 `public Date november (`
 `int day, int year) { ... }`

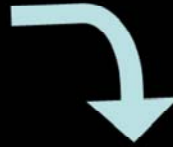
Compare

```
void process() {  
    input();  
    count++;  
    output();  
}
```

```
void process() {  
    input();  
    tally();  
    output();  
}  
private void tally() {  
    count++;  
}
```

Compose Method Pattern

```
public void add(Object element) {  
    if (!readOnly) {  
        int newSize = size + 1;  
        if (newSize > elements.length) {  
            Object[] newElements =  
                new Object[elements.length + 10];  
            for (int i = 0; i < size; i++)  
                newElements[i] = elements[i];  
            elements = newElements;  
        }  
        elements[size++] = element;  
    }  
}
```



```
public void add(Object element) {  
    if (readOnly)  
        return;  
    if (atCapacity())  
        grow();  
    addElement(element);  
}
```

4-Feb-20 7:50 PM

29

Benefits and Liabilities

- +Efficiently communicates what a method does and how it does what it does.
- +Simplifies a method by breaking it up into well-named chunks of behavior at the same level of detail.
- Can lead to an overabundance of small methods.
- Can make debugging difficult because logic is spread out across many small methods.

Improve

```
flags |= LOADED_BIT;
```

- Solution: Extract to a message

```
void setLoadedFlag() {  
    flags |= LOADED_BIT;  
}
```

Improve Code

```
// Check to see if the employee
// is eligible for full benefits
if ((employee.flags & HOURLY_FLAG)
    && (employee.age > 65)) ...

if (employee.
    isEligibleForFullBenefits())
```


Improve

```
public static void endMe() {  
    if (status == DONE) {  
        doSomething();  
        ...  
        return  
    } else {  
        <other code>  
    }  
}
```

Avoid “else”

```
public static void endMe() {  
    if (status == DONE) {  
        doSomething();  
        ...  
        return;  
    }  
    <other code>  
}
```

Improve

```
public Node head() { ...  
    if (isAdvancing())  
        return first;  
    else  
        return last;  
}  
  
public static Node head() { ...  
    return isAdvancing()?first:last;  
}
```

Improve

- Function signature
`void render(boolean isSuite)`
- Remove boolean variables from functions.
Have two functions.

```
void renderForSuite()
```

```
void renderForSingleTest()
```

Avoid Long parameter lists

- Long parameter list means more chances of an error.
 - `CreateWindow` in Win32 has 11 parameters.
- How to solve it?

4-Feb-20 7:50 PM

36

Break the method or

Create Helper class to hold parameters

Improve

```
class Board {  
    ...  
    String board() {  
        StringBuffer buf = new StringBuffer();  
        for(int i = 0; i < 10; i++) {  
            for(int j = 0; j < 10; j++)  
                buf.append(data[i][j]);  
            buf.append("\n" );  
        }  
        return buf.toString();  
    }  
}
```

Only one level of indentation per method

```
Class Board {  
    ...  
    String board() {  
        StringBuffer buf = new StringBuffer();  
        collectRows(buf);  
        return buf.toString();  
    }  
    void collectRows(StringBuffer buf) {  
        for(int i = 0; i < 10; i++)  
            collectRow(buf, i);  
    }  
    void collectRow(StringBuffer buf, int row) {  
        for(int i = 0; i < 10; i++)  
            buf.append(data[row][i]);  
        buf.append("\n");  
    }  
}
```

?

4-Feb-20 7:50 PM

38

Self documenting code

Q20 – inch; Q06 – NO_GROUPING; Q07 – addHoliday; Q21 – full name in English;

Q22 – complexPassword; Q23 – TokenStream; Q25 - orderItems

Good Comments?

```
String text = ""'bold text'"";
ParentWidget parent = new BoldWidget(
    new MockWidgetRoot(), ""'bold text'"");
AtomicBoolean failFlag = new AtomicBoolean();
failFlag.set(false);
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
    WidgetBuilderThread widgetBuilderThread = new
        WidgetBuilderThread(widgetBuilder, text,
            parent, failFlag);
    Thread thread = new Thread(widgetBuilderThread);
    thread.start();
}
assertEquals(false, failFlag.get());
```

4-Feb-20 7:50 PM

39

Comment to

- explain WHY we are doing something?
- Also for external documentation. Javadocs public API
- To give warnings: e.g. Don't run unless you want to kill this program.
- Todo comments

Find the flaw

```
if (deletePage(page) == E_OK)
    if (registry.deleteReference(page.name) == E_OK)
        if (configKeys.deleteKey(
            page.name.makeKey()) == E_OK)
            logger.log("page deleted");
        else
            logger.log("configKey not deleted");
    else
        logger.log("deleteReference ... failed");
try {
    deletePage(page);
    registry.deleteReference(page.name);
    configKeys.deleteKey(
        page.name.makeKey());
} catch (Exception e) {
    logger.log(e.getMessage()); }
```

Samurai Principle

- Throw exception if any error occurs.



4-Feb-20 7:50 PM

41

Guidelines

- Prefer long to int and double to float to reduce errors.
- Avoid literal constants other than "", null, 0 and 1.

Constants

```
for (int j=0; j<34; j++) {  
    s += (t[j]*4)/5;  
}
```



```
int realHoursPerIdealDay = 4;  
const int WORK_DAYS_PER_WEEK = 5;  
int sum = 0;  
for (int j=0; j < NUMBER_OF_TASKS; j++) {  
    int realTaskDays = taskEstimate[j] *  
        realHoursPerIdealDay;  
    int realTaskWeeks = realTaskDays /  
        WORK_DAYS_PER_WEEK;  
    sum += realTaskWeeks;  
}
```

?

Q32 – FoodSalesReport.

Fail-Fast

- Compile time checking is best

Contrast this signature:

```
void assignCustomerToSalesman (  
    long customerId,  
    long salesmanId);
```

with

```
void assignCustomerToSalesman (  
    Customer c,  
    Salesman s);
```

Fail Fast

- Bad:
In Java a Properties class maps String to String

4-Feb-20 7:50 PM

45

put method does not make this check
save method does this check

Compare

```
void setAmount(int  
    value, String  
    currency) {  
    this.value =  
        value;  
    this.currency =  
        currency;  
}
```

```
void setAmount(  
    Money value) {  
    this.value=value;  
}
```

4-Feb-20 7:50 PM

46

The code on the right

- Is flexible because any runner can be used
- Less prone to error, because the caller does not have to worry about exceptions

Improve

```
setOuterBounds ( x, y,  
                 width, height);  
setInnerBounds ( x+2, y+2,  
                 width-4, height-4);
```

- Solution: Use Parameter Object

```
setOuterBounds (bounds);  
setInnerBounds (bounds.expand  
                (-2));
```


Guideline

- Wrap all primitives and Strings

4-Feb-20 7:50 PM

48

An int on its own is just a scalar with no meaning. When a method takes an int as a parameter, the method name needs to do all the work of expressing the intent. If the same method takes an hour as a parameter, it's much easier to see what's happening. Small objects like this can make programs more maintainable, since it isn't possible to pass a year to a method that takes an hour parameter. With a primitive variable, the compiler can't help you write semantically correct programs. With an object, even a small one, you are giving both the compiler and the programmer additional information about what the value is and why it is being used.

Small objects such as hour or money also give you an obvious place to put behavior that otherwise would have been littered around other classes. This becomes especially true when you apply the rule relating to getters and setters and only the small object can access the value.

Direct access to Variables

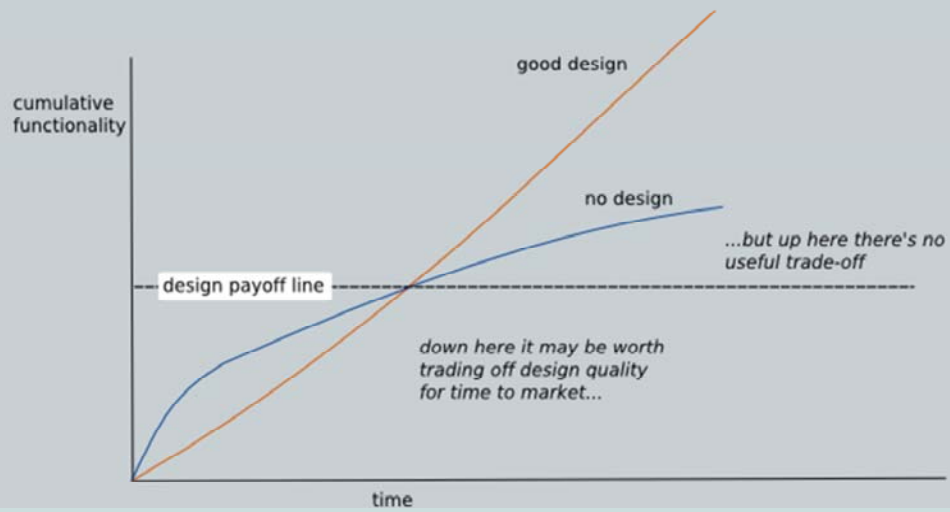
- Compare
`doorRegister=1;`
with
`openDoor ();`
with
`door.open ();`

4-Feb-20 7:50 PM

49

The last one is the best. It uses objects and functions.

Is Good Design worth it?



Let us be Practical

- Not all of a large system will be well designed.
- Our application has
 - Generic Subdomain
 - Supporting Subdomain
 - Core Domain

4-Feb-20 7:50 PM

51

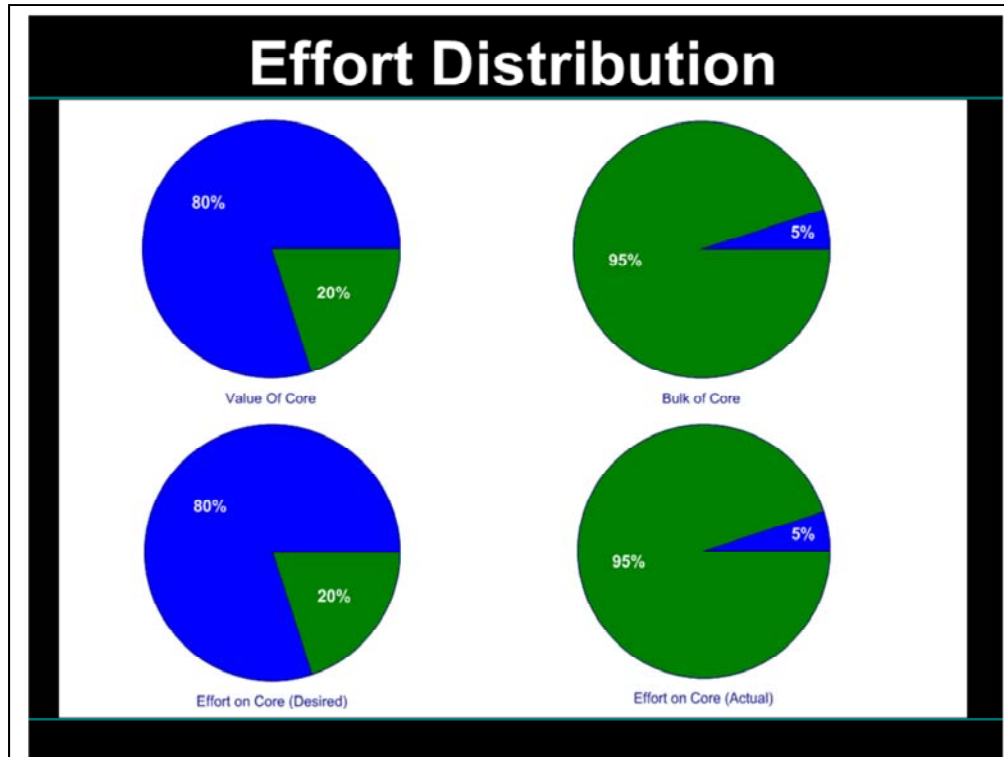
Core domain:

The core domain is the one with business value, the one that makes your business unique. Apply your best talent to the core domain. Spend a lot of time on it and make it as good as you can.

Generic Subdomain: Identify cohesive subdomains that are not the motivation for your project. Factor out generic models of these subdomains and place them in separate modules. These domains are not as important as the core domains. Don't assign your best developers to them. Consider off-the-shelf solutions or a published model. E.g. calendar, Invoicing, Accounting

Supporting Subdomain: Related to our domain but not worth spending millions of dollars every year. E.g. Advt's on bank statements.

User ratings on Amazon and Ebay. On Amazon, this feature is supporting domain because customers usually buy the book even if others have given it low rating and they really want it. For ebay it is core domain because people will hesitate to make a deal with a person with low rating.



From Domain Driven Design.

Domain Vision Statement

- Write a short description (about one page) of the core domain and the value it will bring.
- Ignore aspects that are the same as other domain models.
- Write this statement early and revise it as you gain new insight.

53

Highlighted Core: Write a very brief document (3-7 pages) that describes the core domain and the primary interactions between core elements.

Design Smells

- Duplication
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Primitive Obsession
- Switch statements
- Parallel Inheritance Hierarchy
- Lazy class
- Speculative Generality
- Data Class
- Refused Bequest
- Comments
- Data Clumps

4-Feb-20 7:50 PM

54

Divergent change occurs when one class is commonly changed in different ways for different reasons. If you look at a class and say, "Well, I will have to change these three methods every time I get a new database; I have to change these four methods every time there is a new financial instrument," you likely have a situation in which two objects are better than one. That way each object is changed only as a result of one kind of change.

Shotgun surgery is similar to divergent change but is the opposite. You whiff this when every time you make a kind of change, you have to make a lot of little changes to a lot of different classes. When the changes are all over the place, they are hard to find, and it's easy to miss an important change.

Feature Envy: The whole point of objects is that they are a technique to package data with the processes used on that data. A classic smell is a method that seems more interested in a class other than the one it actually is in.

Data Clumps: Data items tend to be like children; they enjoy hanging around in groups together. Often you'll see the same three or four data items together in lots of places: fields in a couple of classes, parameters in many method signatures. Bunches of data that hang around together really ought to be made into their own object.

Data classes are like children. They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility.

Refused Bequest: Subclasses get to inherit the methods and data of their parents. But what if they don't want or need what they are given? They are given all these great gifts and pick just a few to play with.

Summary

- Keep it DRY, shy and tell the other guy.
- Prefer composition over inheritance
- Self-documenting code
- Not all parts of a large system will be well-designed.

4-Feb-20 7:50 PM

55

No Golden Bullet.