

OOAD

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

It's different



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

2

Let there be no doubt that object-oriented design is fundamentally different than traditional structured design approaches. It requires a different way of thinking about decomposition, and it produces software architectures that are largely outside the realm of structured design culture. – Grady Booch

Many people tell the story of the CEO of a software company who claimed that his product would be object oriented because it was written in C++. Some tell the story without knowing that it is a joke.

Does OO benefit?

- Lesser Code
- Faster
- Easier Maintenance

2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

3

The product of object thinking is software that manifests simplicity and composability, which lead, in turn, to adaptability, flexibility, and evolvability.

A 1 million-line program, written in with non-OO concepts can be duplicated in OO with 100K LOC or fewer.

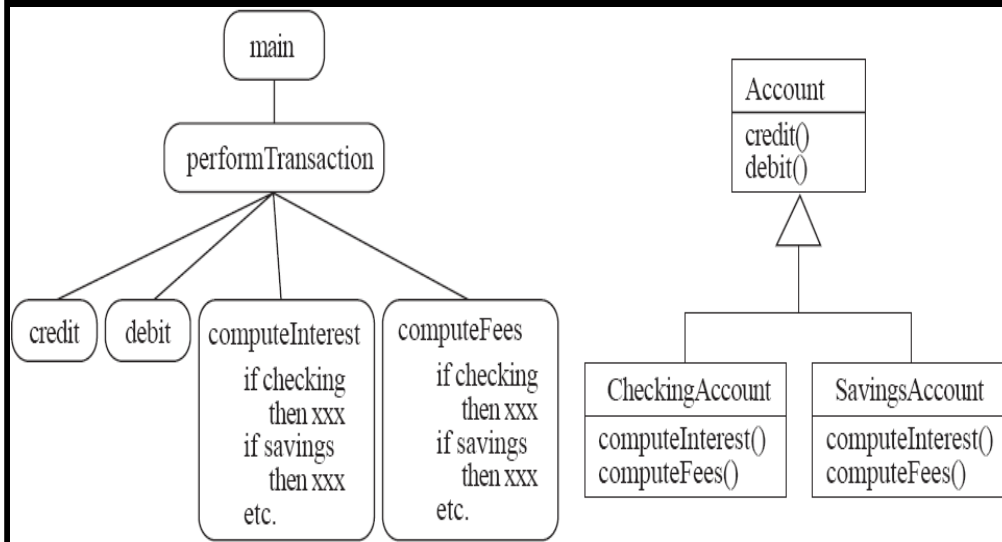
Time to delivery is reduced by at least 50% and usually 70%.

Project that took 2 years can complete in 8 to 12 months.

Exceptions:

- Device driver written in 100 lines of assembly
- Internet search engine needs “database thinking”
- Network Router (I/O performance)
- Mobile phone (memory footprint)
- Embedded Sensor (power consumption)

How is OO different?



Think like an Object and not like a Computer.

2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

4

How is object thinking different from thinking like a computer?

Object thinking involves a very different means of solution – a cooperating community of virtual persons

Object thinking focuses our attention on the problem space rather than the solution space

OO Design is an Art



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

5

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



2-Feb-23 5:55 PM



<https://vijaynathani.github.io/>

6

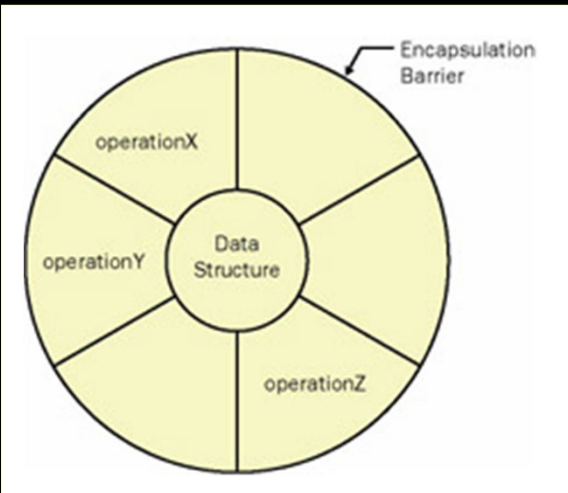
Collaborative rather than imperial management style

OOD -


Commitment to design based on coordination and cooperation rather than control

Driven by internal capabilities instead of conforming to external procedures.

Representation of an Object



or



Objects are not something that we do; objects are a way that we think.

2-Feb-23 5:55 PM
<https://vijaynathani.github.io/>
7

In a data-driven approach, the attributes of an object are discovered first and then responsibilities are meted out as a function of which object holds which data.

A behavioral approach mandates the assignment of responsibilities first. Only when you are satisfied with the distribution of responsibilities among your objects are you ready to make a decision about what they need to know to fulfill those responsibilities and which parts of that knowledge they need to keep as part of their structure—in instance variables or attributes.

This is the single biggest difference in definition between data-driven and behavior-driven or responsibility-driven approaches to objects.

Responsibilities are not functions, although there is a superficial resemblance. The former reflects expectations in the domain—the problem space—while the latter reflects an implementation detail in the solution space—the computer program.

Our goal during discovery is object definition, not object specification. Definition means we want to capture how the object is defined by those using it, what they think of that object, what they expect of it when they use it, and the extent to which it is similar to and different from other objects in the domain. Specification will come later (maybe 30 seconds later if you are doing XP and working on a single object), when we allow ourselves to consider how this object might be implemented (simulated) in software.

It's a major mistake, from an object thinking perspective, to define objects in terms of an application instead of a domain.

Decide Classes

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



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

8

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.

2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

9

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

How to find classes?

- Object thinking emphasizes the need to *understand the domain first*.



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

10

Simulation of a problem domain drives object discovery and definition

A truism of software development is that the most costly mistakes are made the first day. Why?

Simply for a lack of knowledge.

Developers anticipate how the computer is going to implement your software before trying to understand how the software should simulate some part of the domain in which it is going to be used.

It's never appropriate to tell yourself, "This is what the code will look like, so I need an object to hold these parts of the code, and another to hold these parts, and another to make sure these two do what they are told to do when they are told to do it," which is precisely what structured development tempts you to do.

Perhaps the greatest benefit of object thinking is that of helping you start off in the right direction. Object thinking does this by emphasizing the need to *understand the domain first*.

Software expertise does not trump domain expertise. The longer a software developer works in a domain, the more effective her software work will be.

Most books and methods addressing how to do object development recommend that the object discovery process begin with underlining the nouns (names) in a domain or problem description. While it is true that many of those nouns will indeed turn out to be viable objects, it is unlikely that any written description will be

sufficiently complete or accurate to meet the needs of domain

Metaphor

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

2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

11

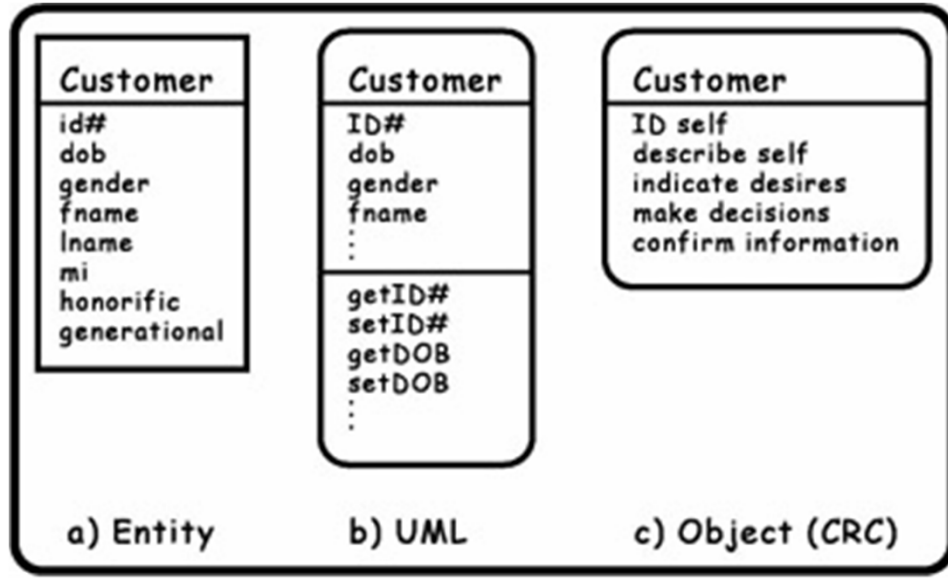
Distributed cooperation and communication must replace hierarchical centralized control as an organizational paradigm. E.g. A traffic signal and cars.

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?

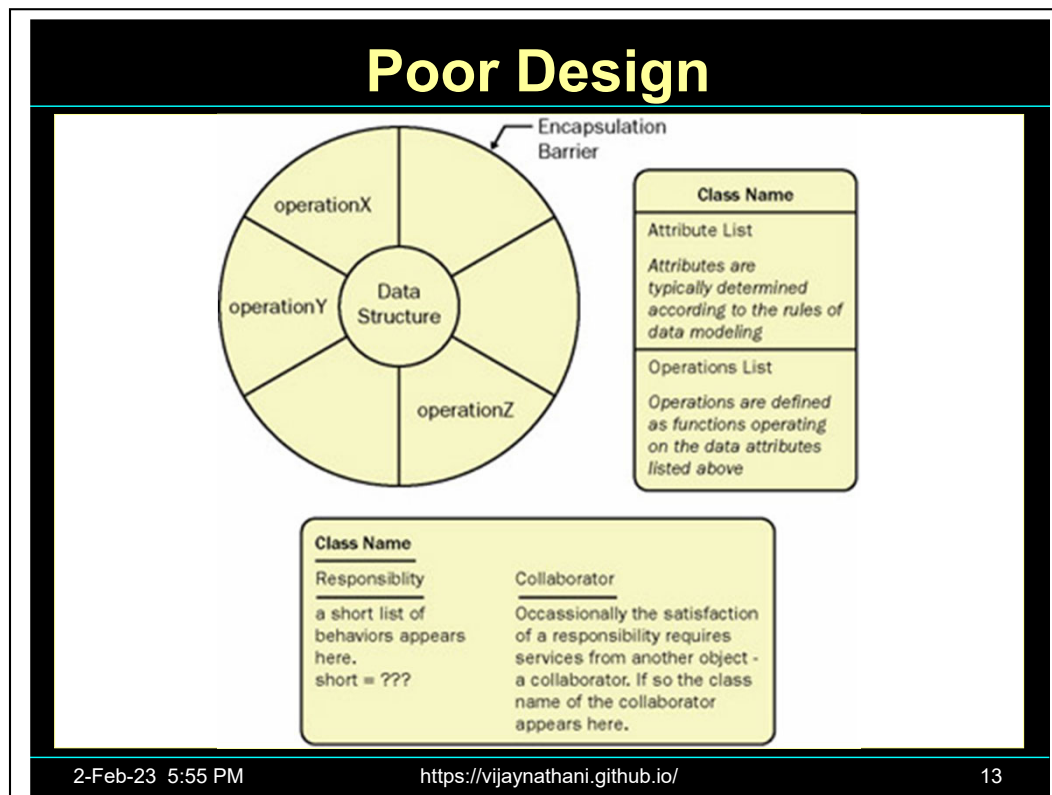
How should our Class be?



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

12



Shown above are "soccer bar", animated data entry and CRC card. All three are not very useful.

They depict what is to be built instead of what is to be modeled.

Object discovery and specification must be at domain level.

Metaphors

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



2-Feb-23 5:55 PM

<https://vijaynathani.github.io/>

14

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.