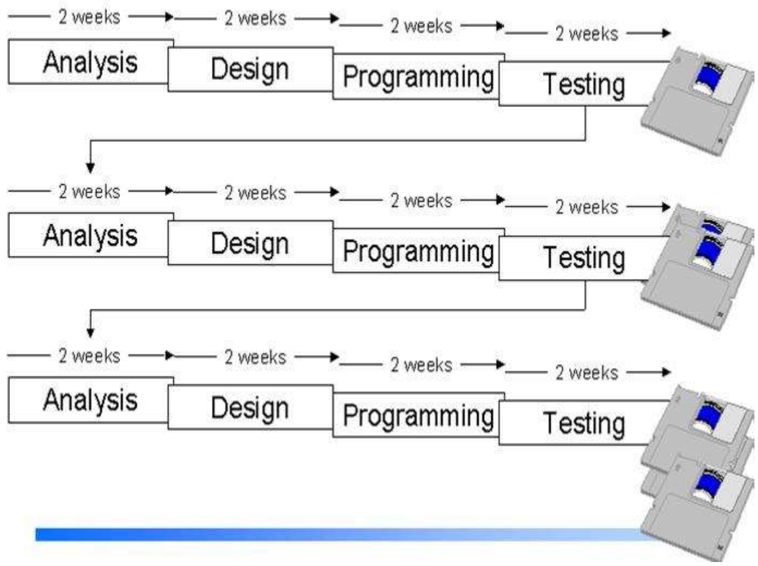
Object-Oriented
Software Development Process

Lecture note #1

Jungsun Kim

College of Computing
Hanyang University

Fall 2024



Dealing with Changing Requirements

Let's say that you are an instructor at a conference.

People in your class had another class to attend following yours, but don't know where it is located.

One of your responsibilities is to make sure everyone knows how to get to their next class.

Structured Approach

1. **Get the list of people in the class.**

2. **For each person on the list:**
   ① Find the next class they are taking.
   ② Find the location of that class.
   ③ Find the way to get from your classroom to the person's next class.
   ④ Tell the person how to get to their next class.

Structured Approach: Code Skeleton

```
main() {
   …
   get the list of student
   for (each student in the list) {
      schedule = getSchedule(student);
      nextClass = getNextClass(currentClass, schedule);
      source = findLocation(currentClass);
      destination = findLocation(nextClass);
      showDirection(source, destination);
   }
}

getSchedule(…) { … }
getNextClass(…) { … }
findLocation(…) { … }
showDirection(…) { … }
```

Other Approach (i.e., OO Approach)

You post directions to go from this classroom to the other classrooms and then tell everyone in the class, *"I have posted the locations of the classes following this in the back of the room, as well as the locations of other classrooms. Please use them to go to your next classroom."*

You

You will expect everyone

- know what their next class is
- can find the classroom they are to go from the list
- can follow the directions for going to the classrooms themselves

Objects and Responsibilities

Instructor

- Telling people to go to next classroom

Student

- Knowing which classroom they are in
- Knowing which classroom they are to go to next
- Going from one classroom to the next

Classroom

- Having a location

Direction Giver

- Given two classrooms, giving directions from one classroom to the other

OO Approach

1. Start the control program.
2. Instantiate the collection of students in the class.
3. Tell the collection to have the students go to their next class.
4. The collection tells each students to go to their next class.
5. Each student
   ① Find where his next class is.
   ② Determines how to get there.
   ③ Goes there.
6. Done.

What's the difference?

The biggest difference is the

"Shift of Responsibility"

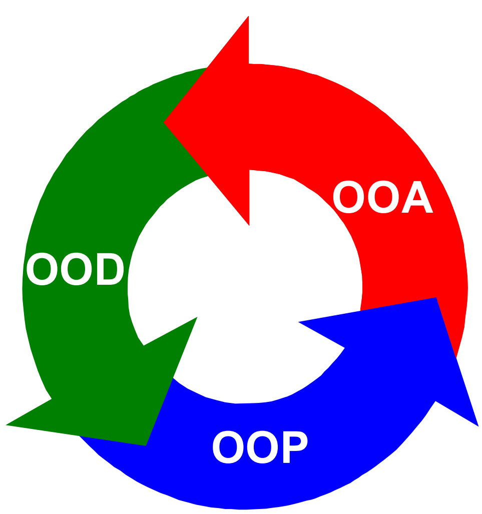
OO Approach: Code Skeleton

```
main() {
   …
   for (each student in the list) {  student.goNextClass();   }
}
```

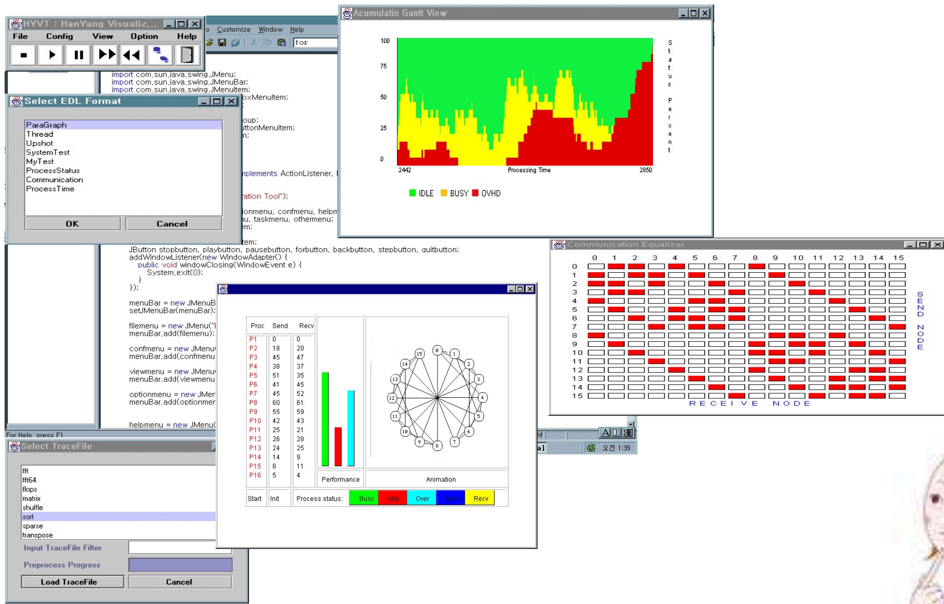
```
class ClassRoom { … }

class DirectionGiver {
   …
   void showDirection(…) {
      find locations for current
      and next class
      …
   }
}
```

```
class Student  {
   …
   void goNextClass() {
      directionGiver.showDirection(
         currentClass, nextClass);
   }
}
```


Motivation for Iterative Development Processes and Object Technologies

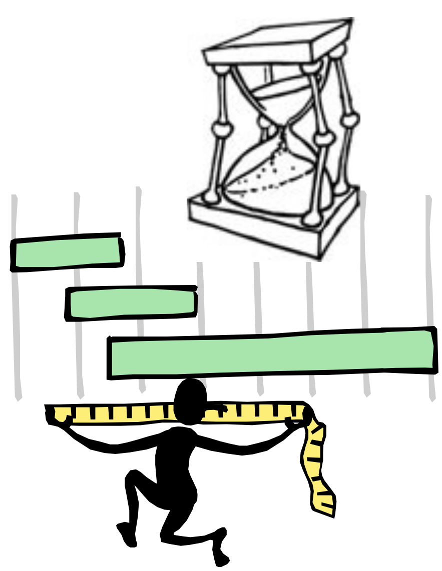


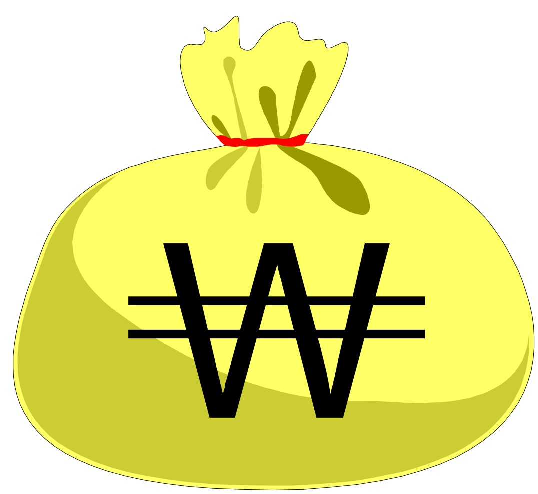
A successful software product must *satisfy* the *stakeholder's* ______________________



Both functional & non-functional requirements!

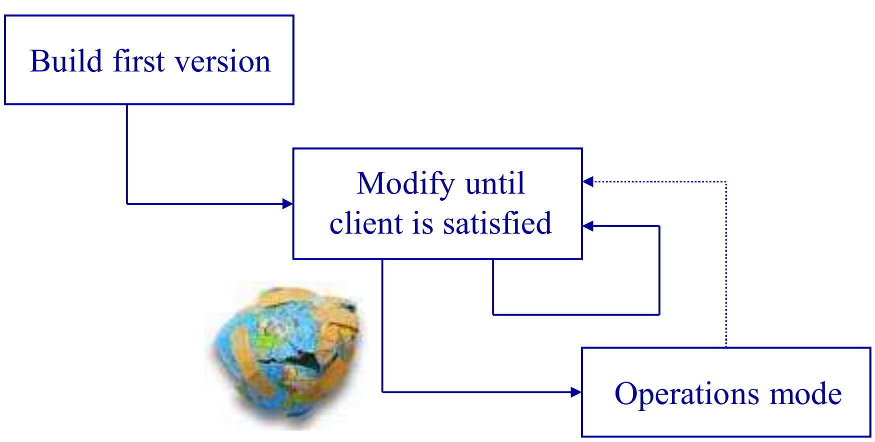
A successful software product must be developed on time and on budget





A successful software product must be *resilient to* change

Software systems were like cathedrals; first we build them and then we pray - *S. Redwine*

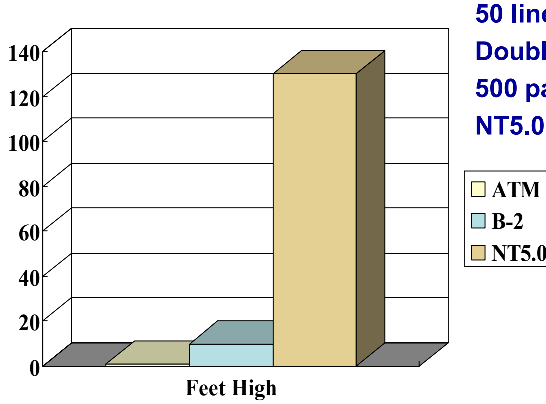


Works well for short programming exercises
Unsatisfactory for any reasonable size

The term software engineering was invented around middle to late 1960's to cope with the software crisis

Cannot produce or maintain high-quality software at reasonable price and on schedule

Why?
- Increasing Size
- Increasing Complexity



50 lines per page
Double sided
500 pages/ream (2inches)
NT5.0 = Statue of Liberty

Engineering Analogy: Building a piece of D.I.Y *vs.* Building a Bridge

Up to now, the programs you've written are probably more like D.I.Y.

Software engineering is more like constructing bridges

Software engineering is an engineering discipline which is concerned with all aspects of software production

"The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints"

- Lethbridge (2001)

"Field of computer science that deals with the building of software systems which are so large or so complex that they are built by a team or teams of engineers"

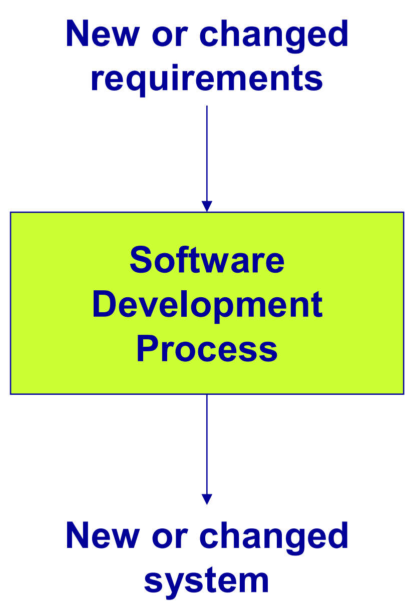
- Ghezzi et. al (1991)

"The technical and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates"

- Fairley (1985)

A development process defines Who is doing what, when and How to reach a certain goal

In software engineering, the goal is to build a software product or to enhance an existing one



Basic workflows (or disciplines) in a process

Analysis

What are the (functional/non-functional) requirements?
-Domain analysis
-Requirements gathering/analysis/spec.

Design

How to devise a logical solution to fulfill the requirements?
-System, Architecture, Internal Designs
-UI, Database designs etc.

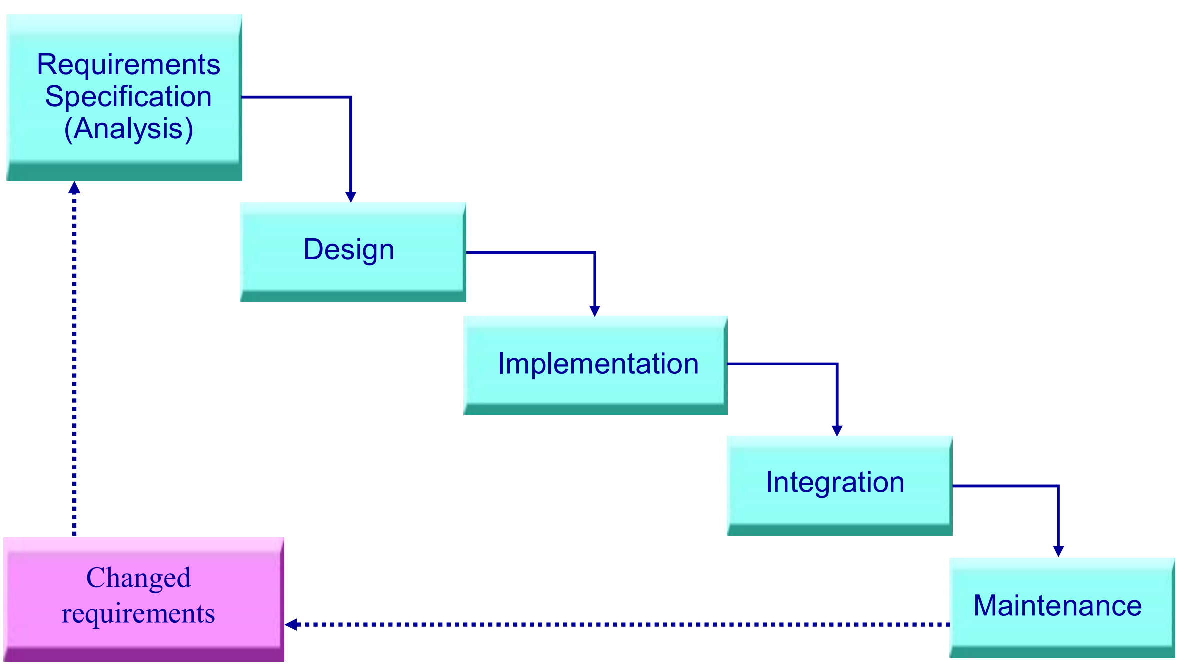
Implementation

Coding of the logical solution

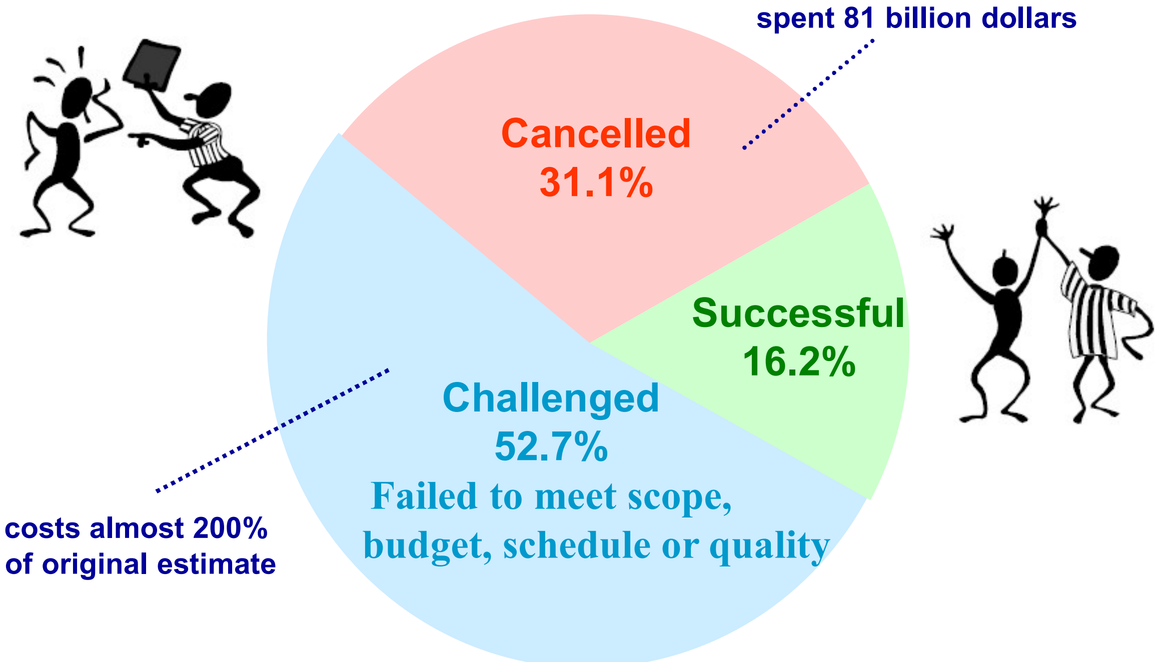
Testing

- Unit test, Integration test, Regression Test
- Acceptance Test
  Does the system do what it was meant to do?

Conventional Waterfall Model



Unpleasant Facts
(All 8380 projects adopted Water Fall Model)



Survey conducted by The Standish Group in 1995

Problems in Conventional Software Development

Long delays

High development cost

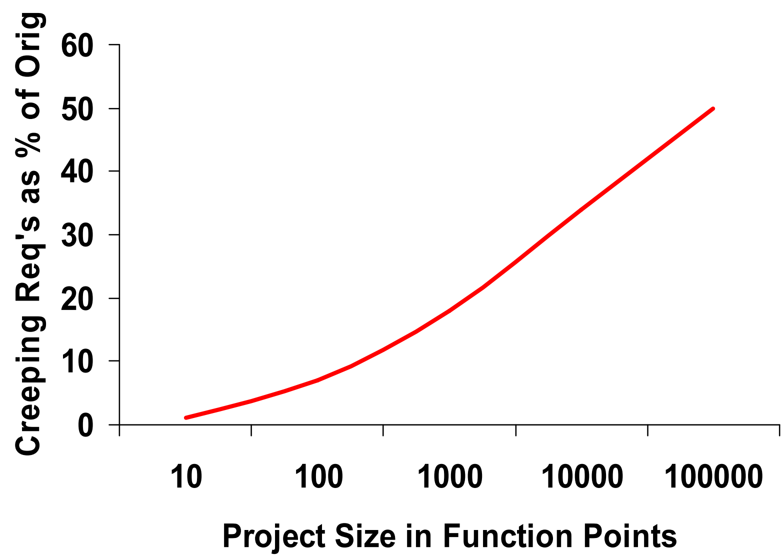
High cancellation rate

Low quality (in *reliability*, *extensibility*, *maintainability* etc.)

High maintenance cost

Faulty Assumption 1:
Requirements can be Fairly Accurate

Applied Software Measurement, Capers Jones, 1997.
Based on 6,700 systems.

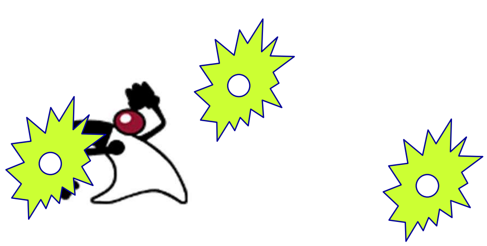


Faulty Assumption 2:
Requirements are Stable

The market changes—constantly

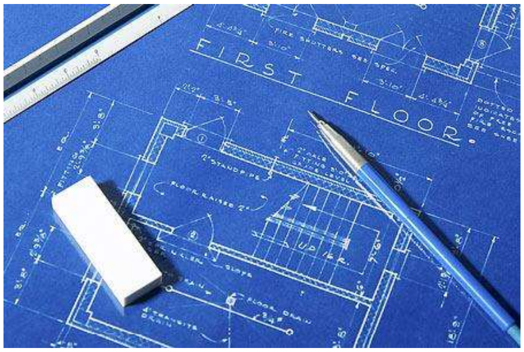
The technology changes

The goals of the stakeholders change



Requirements change during its development and after its deployment ==> moving target problem

Faulty Assumption 3:
Design can be Done, before Programming



Ask a programmer!

Requirements are incomplete & changing

Too many variables, unknowns

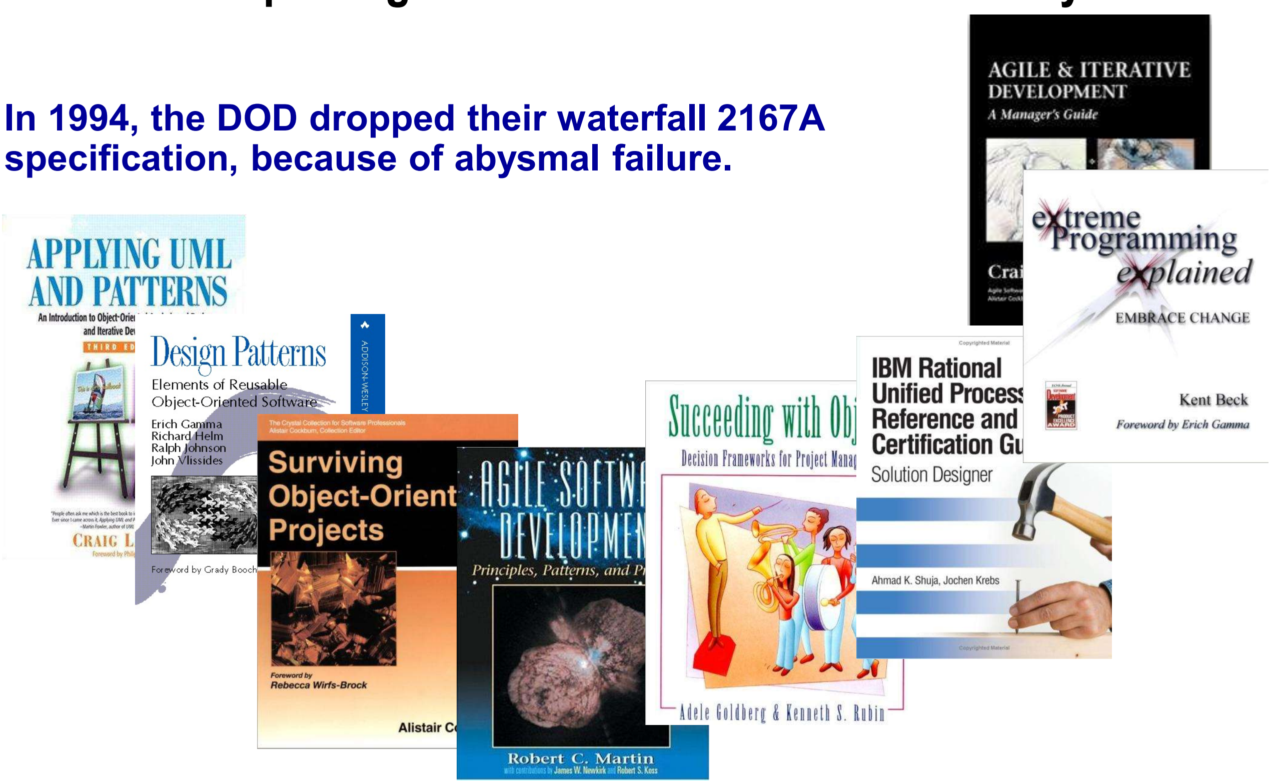
A complete specification must be as detailed as code itself

Software is very "hard"

Discover Magazine, 1999: Software characterized as the most complex "machine" humankind builds.

Recent publications on software development process advocates replacing a waterfall with an iterative lifecycle

In 1994, the DOD dropped their waterfall 2167A specification, because of abysmal failure.



Therefore...

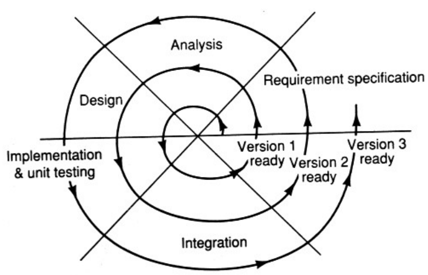
Iterative development embraces changes

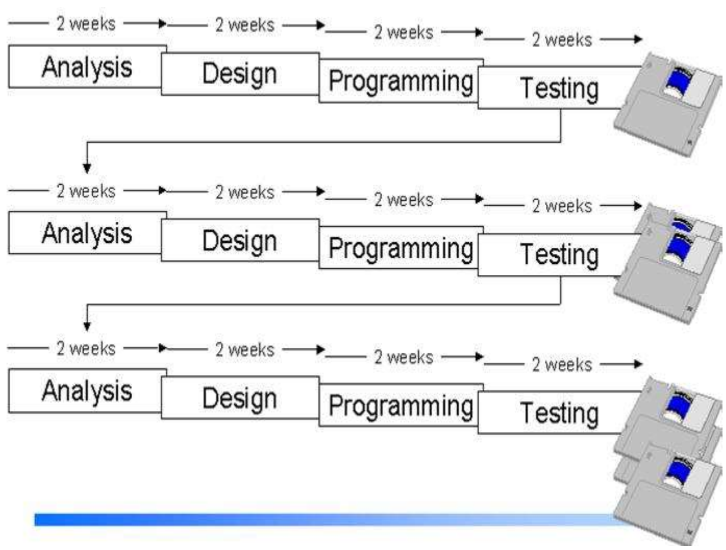
Small steps, feedback and refinement and adaptation

Iterative & incremental

Time-boxed

Aka, evolutionary or spiral





Benefits of Iterative Development

(Page 22 in Textbook)

Early mitigation of high risks (technical, requirements, etc)

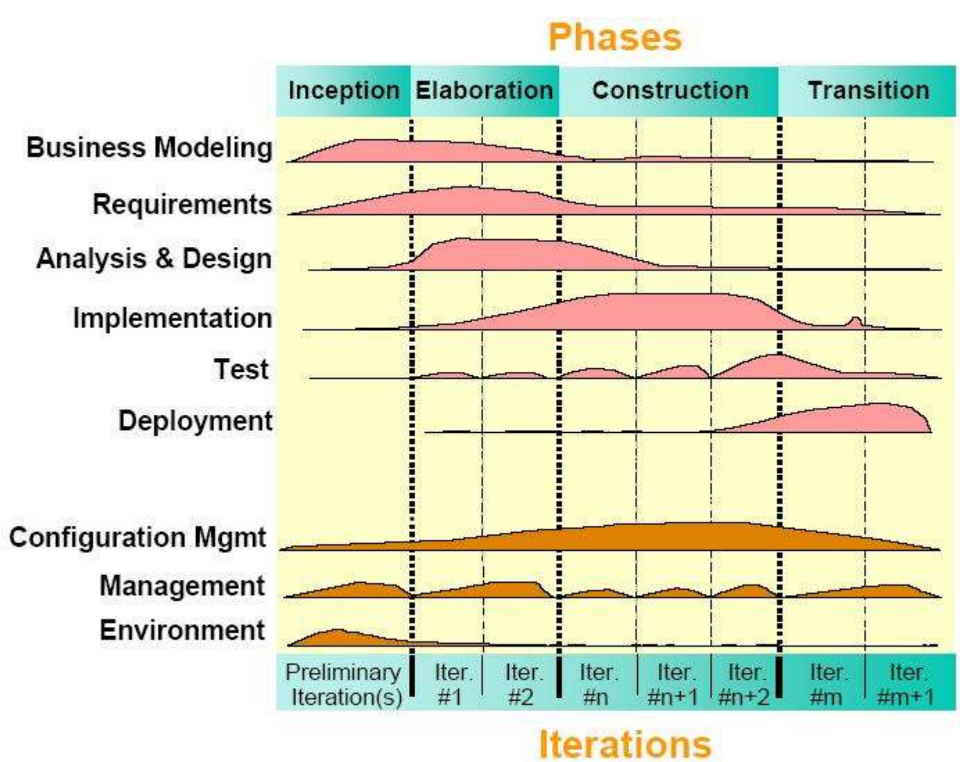
Early visible progress

Early feedback, user engagement, and adaptation

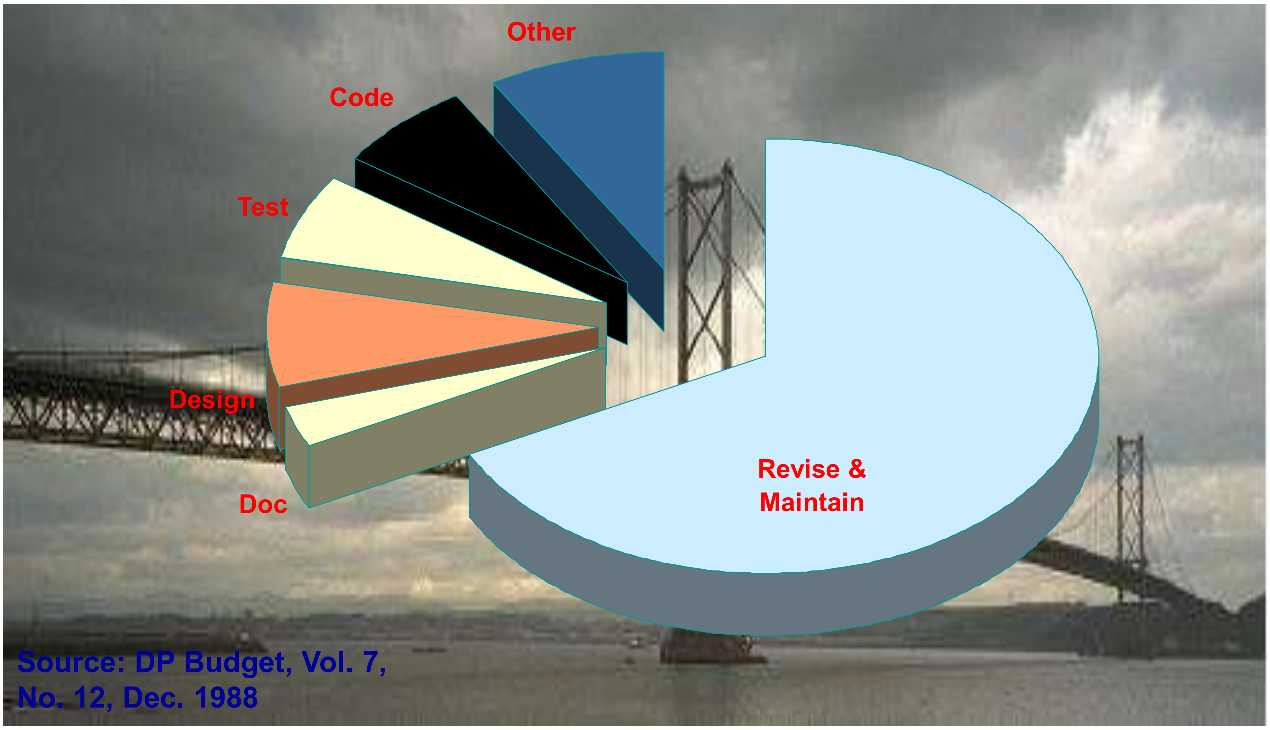
Managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complexity steps

Can improve the process itself, iteration by iteration

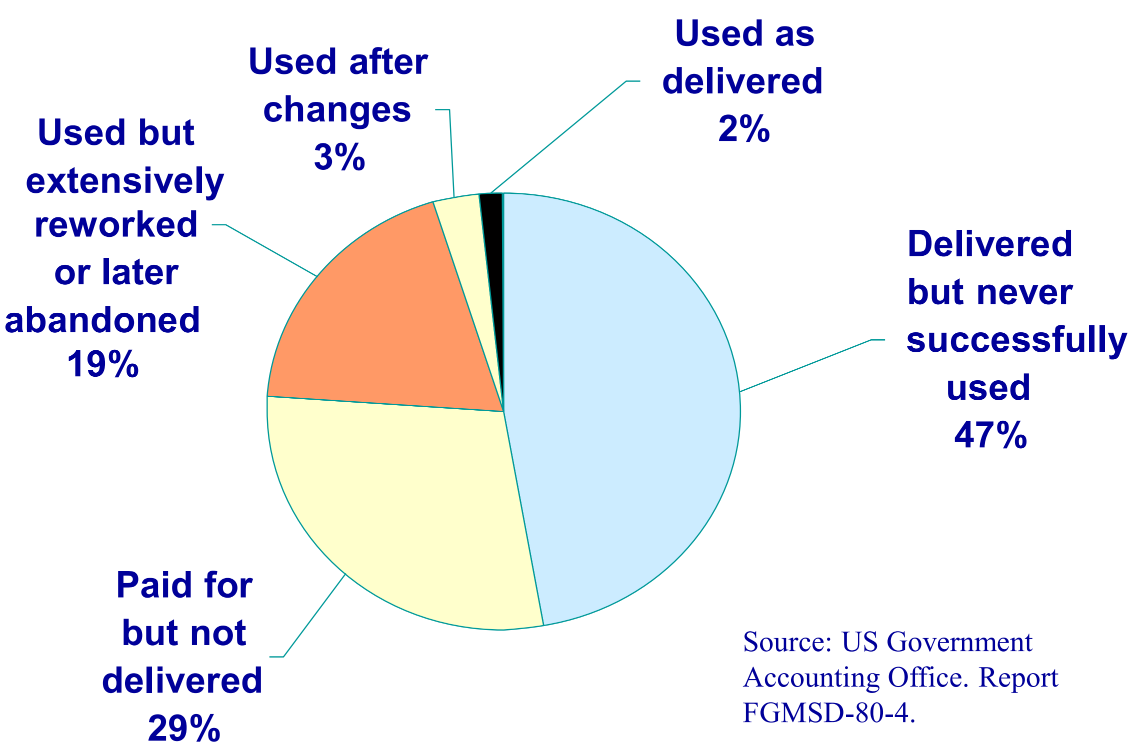
A Popular Iterative Process: The Unified Process



Strategic rational system development plans are based on the complete cost of a system, not solely on development costs



Randomly selected US government software projects: An AT&T study indicates that, on average, business rules change at the rate of 8% per month



Source: US Government Accounting Office. Report FGMSD-80-4.

The real power and advantage of OT is its capacity to tackle complex systems and to support easily adaptable systems, lowering the cost and time of change

***The Corporate Use of OT*, Dec 1997, Cutter Group.**
Prioritized reasons for adopting OT:

1. **Ability to take advantage of new operating systems and tools**
2. **Elegantly tackle complexity & create easy adaptability**
3. **Cost savings**
4. **Development of revenue-producing applications**
5. **Encapsulation of existing applications**
6. **Improved interfaces**
7. **Increased productivity**
8. **Participation in "the future of computing"**
9. **Proof of ability to do OO development**
10. **Quick development of strategic applications**
11. **Software reuse**

Reuse is not usually achieved or worthwhile at the object-level

Research shows no relationship between increased reuse and collecting a library of reusable components from prior projects.

-- Communications of the ACM, pp 75-87 June, 1995

Focus on:

A culture of *framework* creation and use.
Reuse of architecture, analysis and *design patterns*

Design pattern is a description of a problem / solution pair in a certain context

"Each pattern describes a *problem* which occurs over and over again in our environment, and then describes the core of the *solution* to that problem, in such a way that you can use this solution a million times over, without ever doing it the same time twice."

-- Christopher Alexander

Design Pattern Example: Adapter

Name: *Adapter*

Also known as: Wrapper

Context: Client objects call methods of a Supplier object

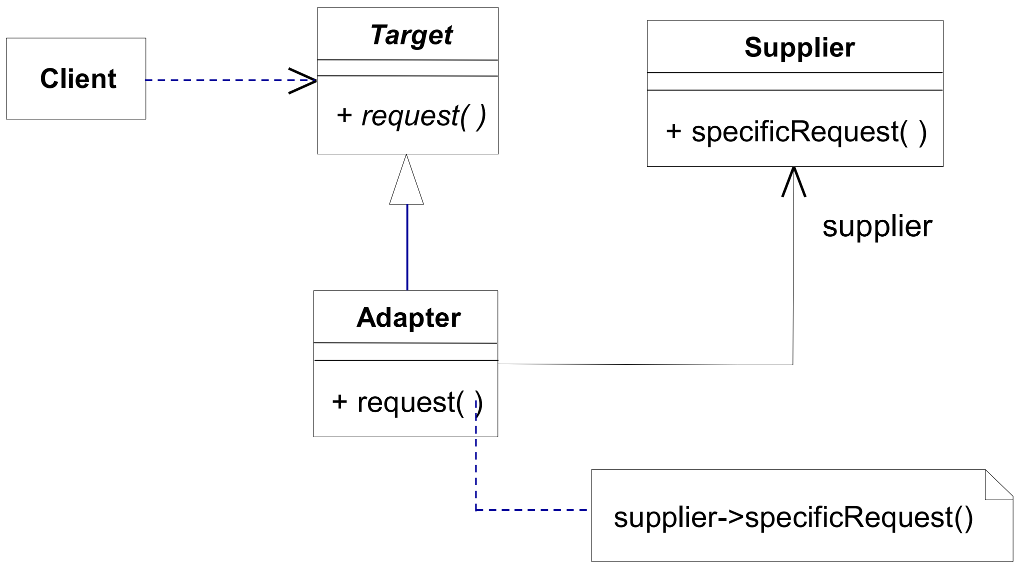
Problem: Client objects expect another interface than the Supplier provides

Solution: Use an Adapter object which adapts one interface to the other

Solution alternatives:

- *Class adapter* (relies on multiple inheritance like C++ or Eiffel, or interface inheritance like Objective C or Java)
- *Object adapter* (relies on single inheritance and delegation)

Object Adapter Solution



A framework is a class library, but more than an ordinary toolkit (such as math, file i/o, data structures …)

An integrated set of cooperating classes

A *semi-complete application*: abstract framework classes are specialized in the application

Inversion of control

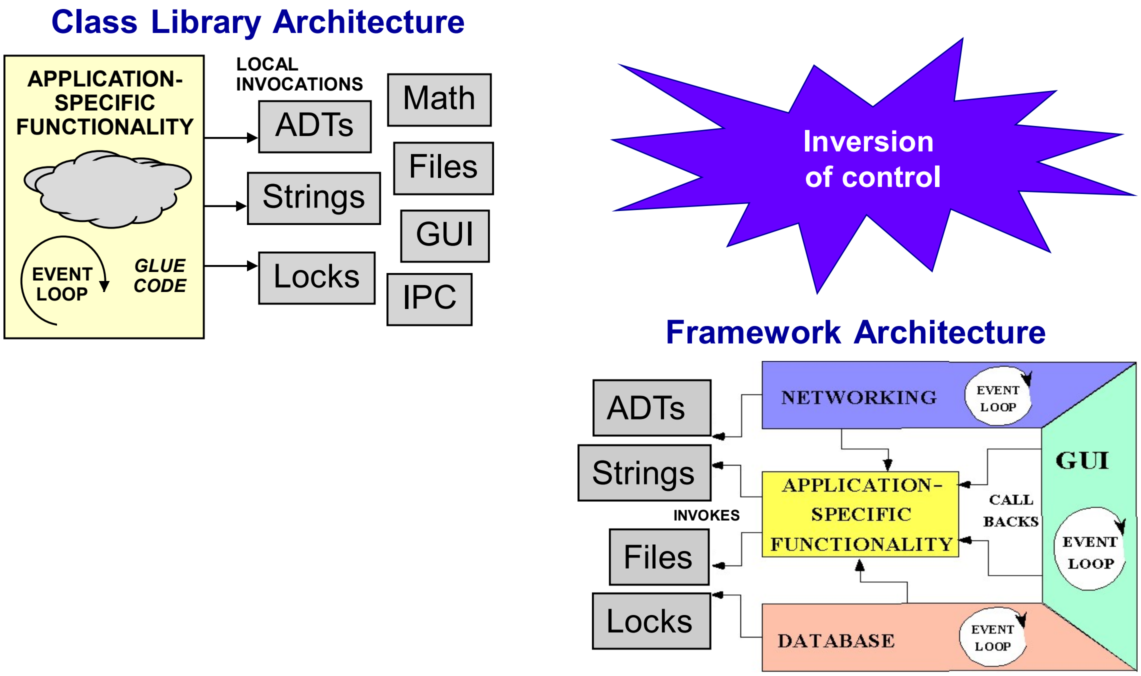
The "main event loop" is often in the framework, rather than in the application code

Code in the framework can invoke code in the application by dynamic binding

Domain-specific

**business,
telecommunications,
windows system,
databases,
etc.**

Hollywood Principle: Don't call me, we'll call you!

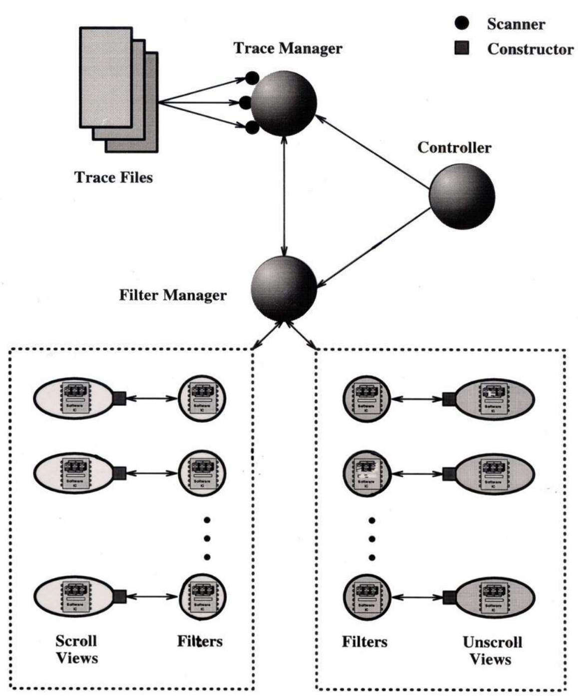


Frameworks allow us to reuse design and code

Framework: family of similar applications

Very fast application development

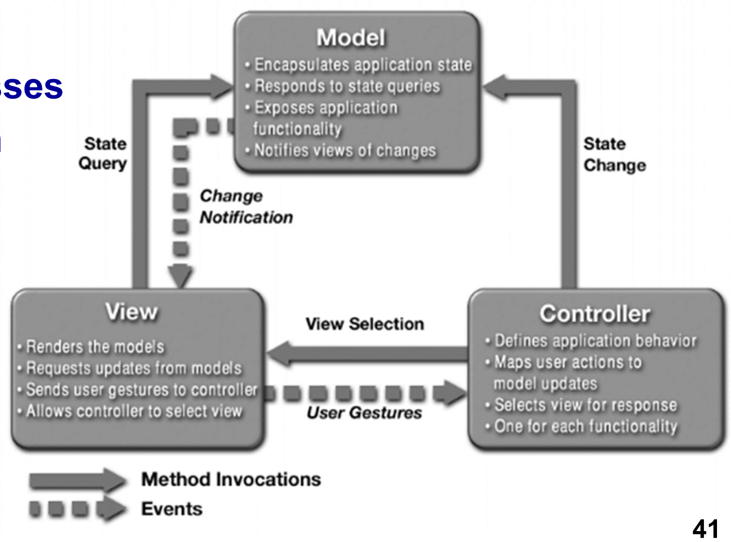
Powerful parameterization mechanism (subclassing and dynamic binding)



Framework Examples

Framework concept origin - use interface frameworks
- Model-View-Controller (MVC) : for smalltalk GUI (1980)
- MacAPP : for Mac applications (1986)
  the first commercially successful framework

Other examples
- Microsoft Foundation Classes
- Choices Operating system framework
- Unidraw Graphical editors
- Android
...



To obtain flexible and reusable systems, it is better to base the structure of software on the __objects__ rather than on the __actions__

Rationale behind OO paradigm:

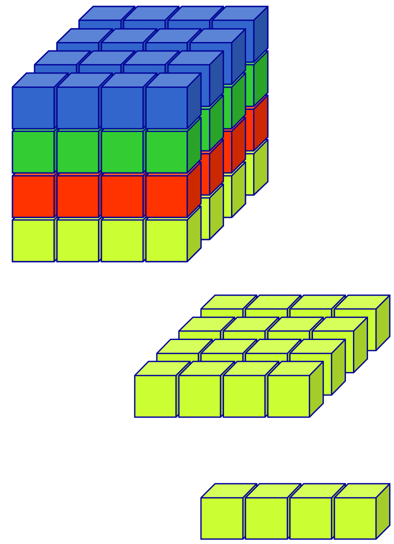
In general, systems evolve, functionality changes, but data objects, interfaces, and components relations tend to remain relatively stable over time

Use it for large systems &
for systems that change often

It is essential to decompose the complex software system into smaller and smaller parts, each of which may then refine independently (i.e., *stepwise refinement*)

Maximum number of chunks of information an individual can simultaneous comprehend is the order of 7 ± 2

--- Miller (1956)



The technique of mastering complexity has been known since ancient times:

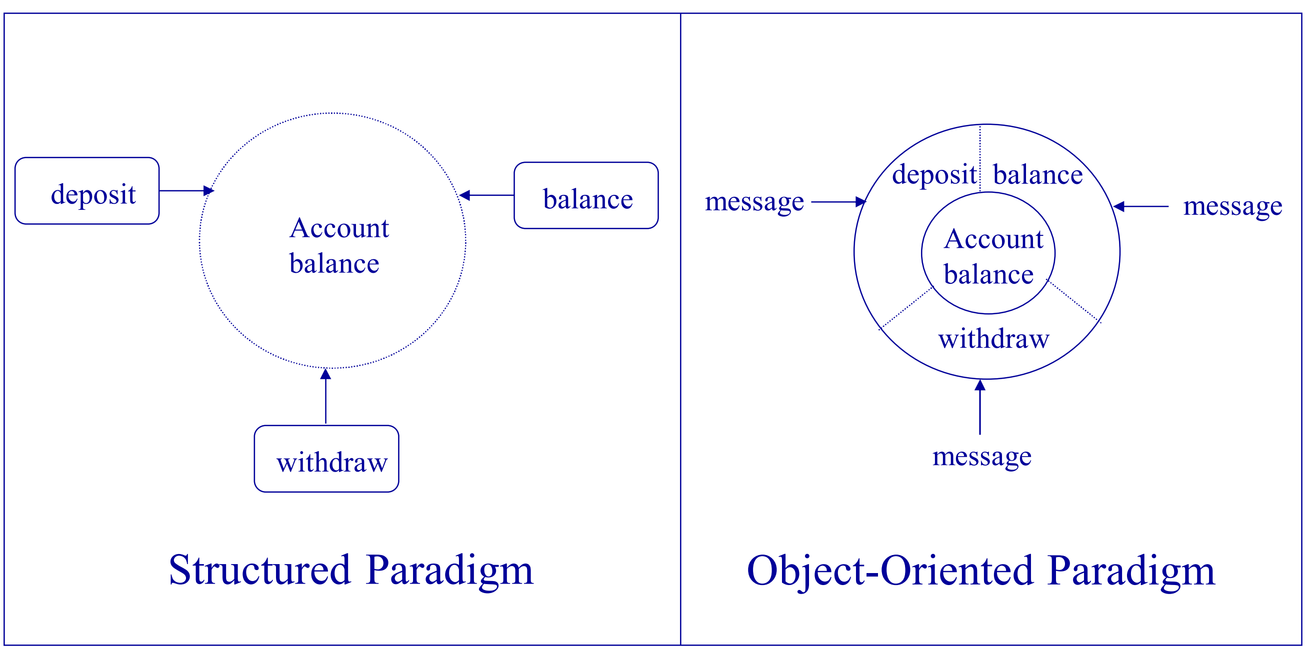
Divide and Conquer

-- Dijkstra (1979)

Structured Paradigm vs. OO Paradigm

Organize a system around procedures/functions

Organize a system around objects



Structured Paradigm

Object-Oriented Paradigm

Structured vs. Object-Oriented Decompositions

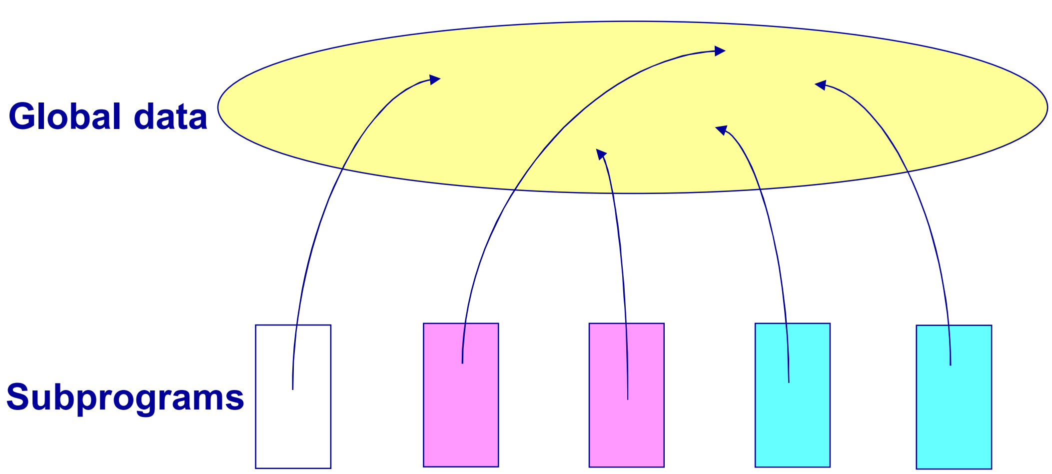
Structured Decomposition

- **Organize a system around procedures/functions**
- ***Program = (Algorithms + Data Structures)***
- **SA/SD/SP**

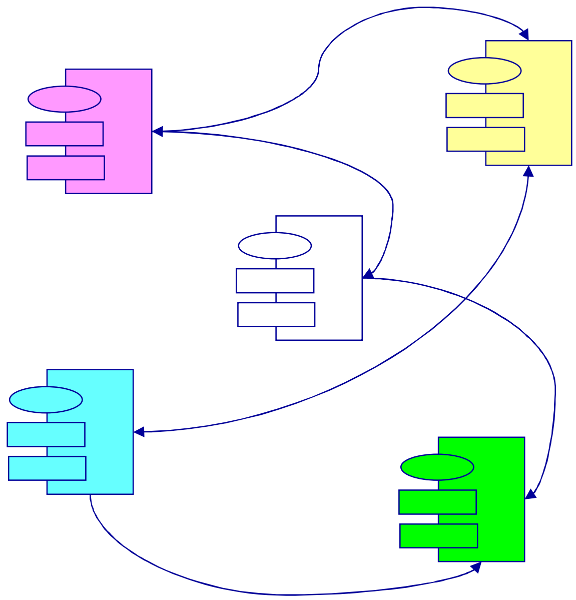
Object-Oriented Decomposition

- **Organize a system around objects**
- ***Object = (Algorithm + Data Structures)***
- ***Program = (Object + Object + ...)***
- **OOA/OOD/OOP**

Design Structure Based on Subprograms



Design Structure Based on Objects (Small Scale)



Design Structure Based on Objects (Large Scale)