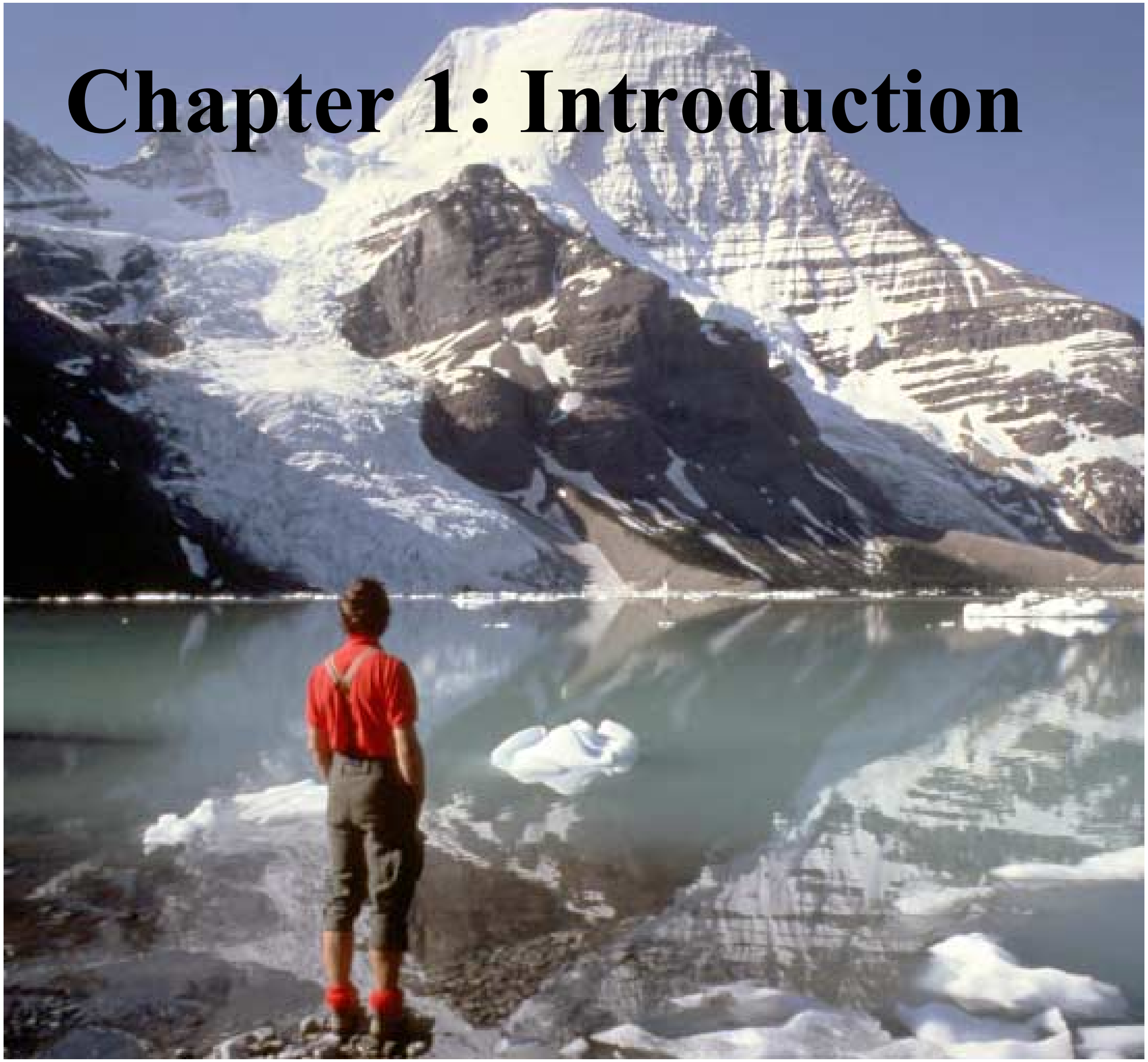
Chapter 1: Introduction



Outline of Today's Lecture

- Software Track Record
- What is Software Engineering
- Software Lifecycle
- Optional stuff for today
  - **Why is software complex?**
  - **Dealing with the complexity**
    - **Abstraction**
    - **Decomposition**
    - **Hierarchy**

Software Production has a Poor Track Record Example: Space Shuttle Software

- Cost: $10 Billion, millions of dollars more than planned
- Time: 3 years late
- Quality: First launch of Columbia was cancelled because of a synchronization problem with the Shuttle's 5 onboard computers.
    - **Error was traced back to a change made 2 years earlier when a programmer changed a delay factor in an interrupt handler from 50 to 80 milliseconds.**
    - **The likelihood of the error was small enough, that the error caused no harm during thousands of hours of testing.**
- Substantial errors still exist.
    - **Astronauts are supplied with a book of known software problems "Program Notes and Waivers".**

Quality of today's software….

- The average software product released on the market is not error free.

…has major impact on Users

Software Engineering: A Problem Solving Activity

- **Analysis**: Understand the nature of the problem and break the problem into pieces
- **Synthesis**: Put the pieces together into a large structure

For problem solving we use

- Techniques (methods):
  - **Formal procedures for producing results using some well-defined notation**
- Methodologies:
  - **Collection of techniques applied across software development and unified by a philosophical approach**
- Tools:
  - **Instrument or automated systems to accomplish a technique**

Software Engineering: Definition

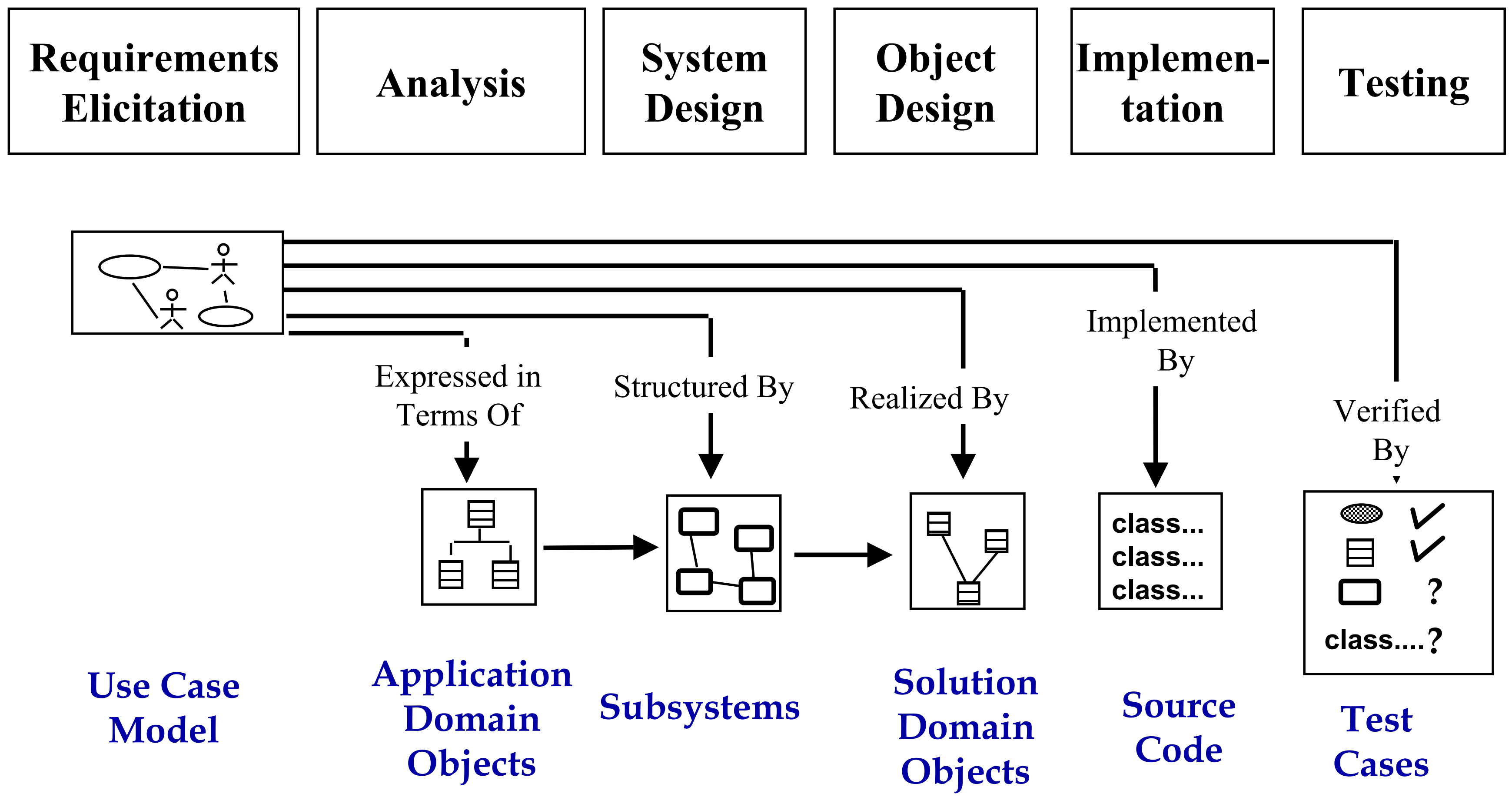
Software Engineering is a collection of techniques, methodologies and tools that help with the production of

- a high quality software system
- with a given budget
- before a given deadline

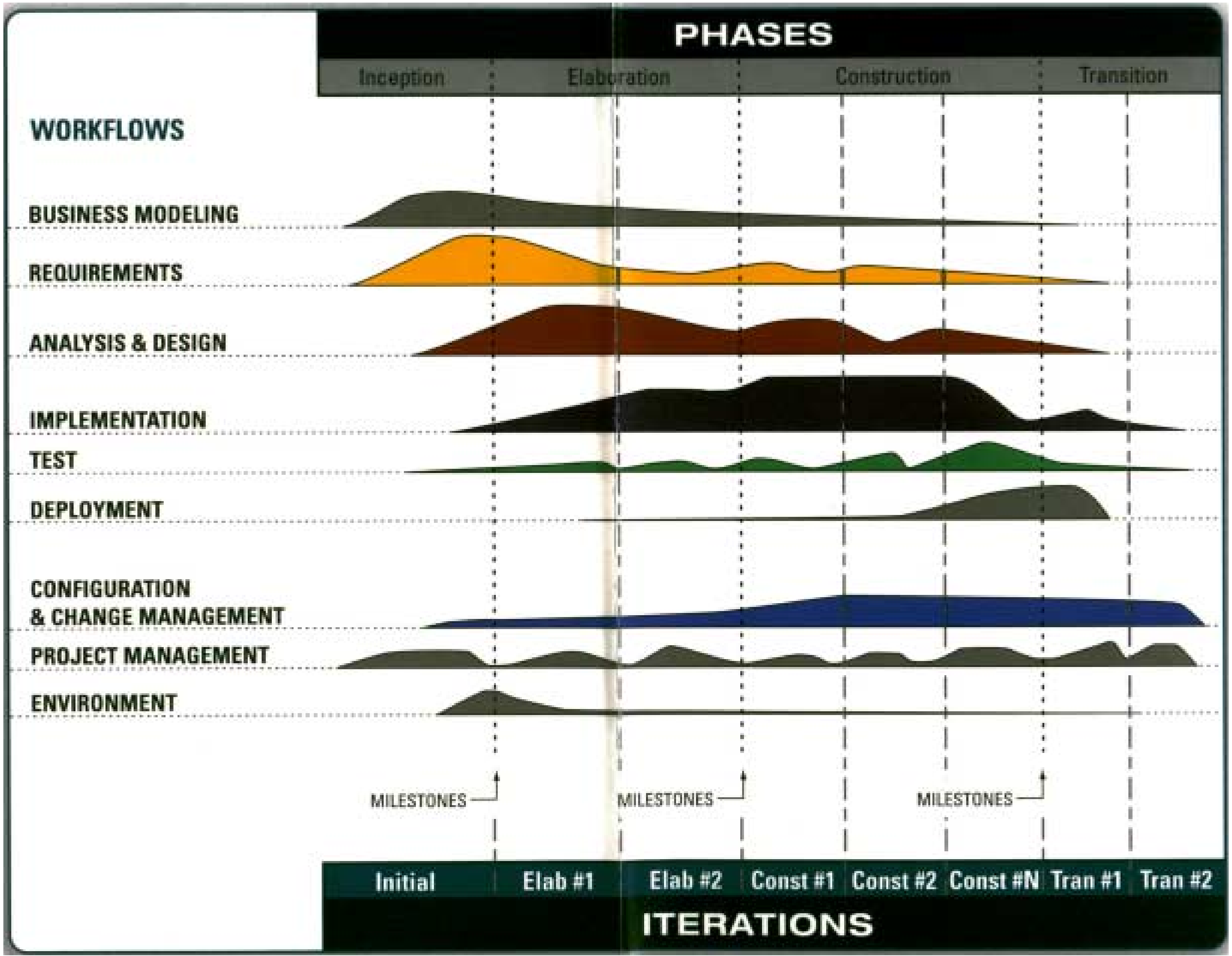
while change occurs.

Software Lifecycle Activities

...and their models



Rational Unified Process (RUP)



Scientist vs Engineer

- Computer Scientist
  - **Proves theorems about algorithms, designs languages, defines knowledge representation schemes**
  - **Has infinite time…**
- Engineer
  - **Develops a solution for an application-specific problem for a client**
  - **Uses computers & languages, tools, techniques and methods**
- Software Engineer
  - **Works in multiple application domains**
  - **Has only 3 months...**
  - **…while changes occurs in requirements and available technology**

Factors affecting the quality of a software system

- **Complexity:**
  - The system is so complex that no single programmer can understand it anymore
  - The introduction of one bug fix causes another bug
- **Change:**
  - The "Entropy" of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive ("Second Law of Software Dynamics").
  - As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

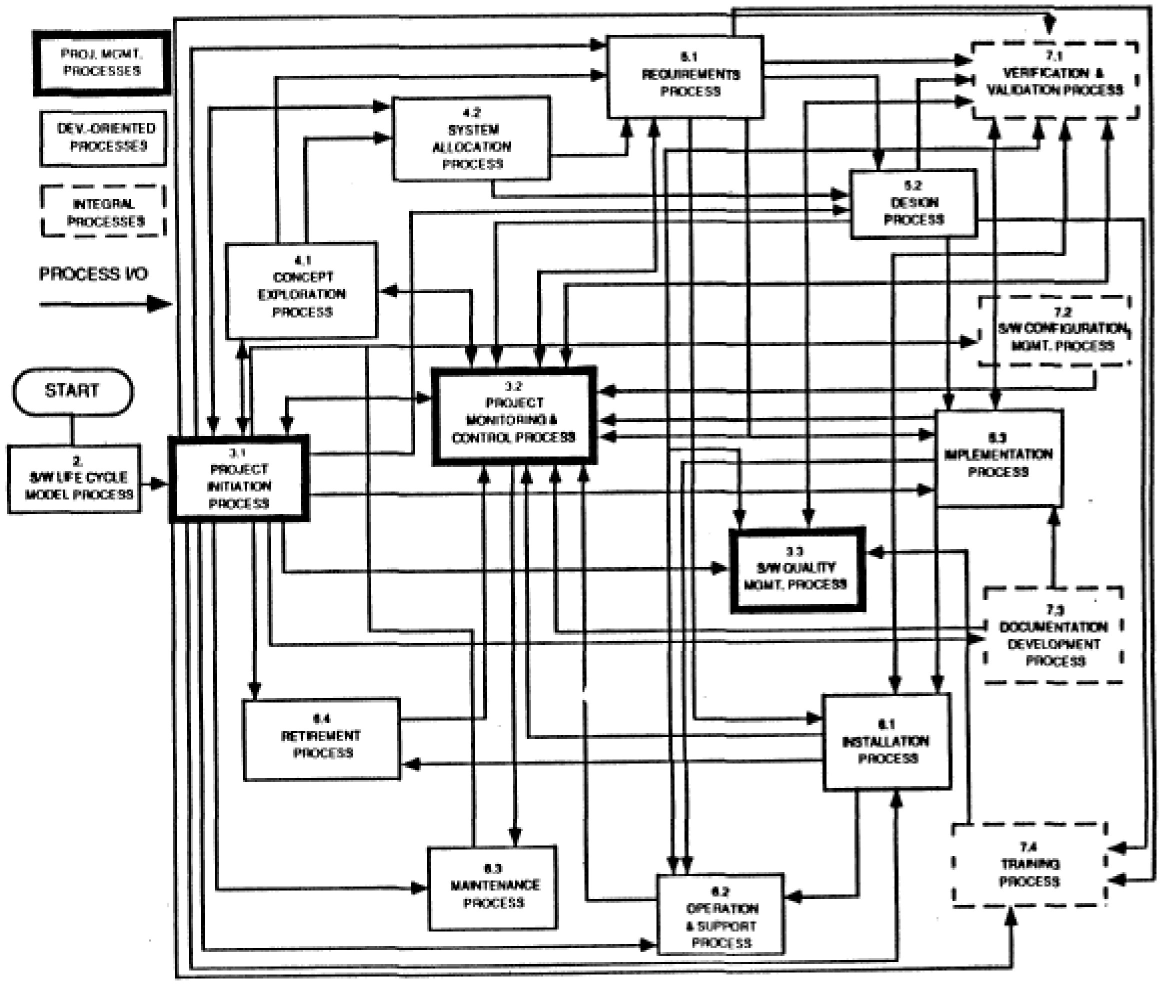
Why are software systems so complex?

- The problem domain is difficult
- The development process is very difficult to manage
- Software offers extreme flexibility
- Software is a discrete system
    - **Continuous systems have no hidden surprises (Parnas)**
    - **Discrete systems have!**

Dealing with Complexity

1. Abstraction
2. Decomposition
3. Hierarchy

What is this?


PROJ. MGMT. PROCESSES
DEV.-ORIENTED PROCESSES
INTEGRAL PROCESSES
PROCESS I/O
START
2. S/W LIFE CYCLE MODEL PROCESS
3.1 PROJECT INITIATION PROCESS
4.1 CONCEPT EXPLORATION PROCESS
4.2 SYSTEM ALLOCATION PROCESS
5.1 REQUIREMENTS PROCESS
7.1 VERIFICATION & VALIDATION PROCESS
5.2 DESIGN PROCESS
3.2 PROJECT MONITORING & CONTROL PROCESS
7.2 S/W CONFIGURATION MGMT. PROCESS
5.3 IMPLEMENTATION PROCESS
3.3 S/W QUALITY MGMT. PROCESS
7.3 DOCUMENTATION DEVELOPMENT PROCESS
6.4 RETIREMENT PROCESS
6.1 INSTALLATION PROCESS
7.4 TRAINING PROCESS
6.3 MAINTENANCE PROCESS
6.2 OPERATION & SUPPORT PROCESS

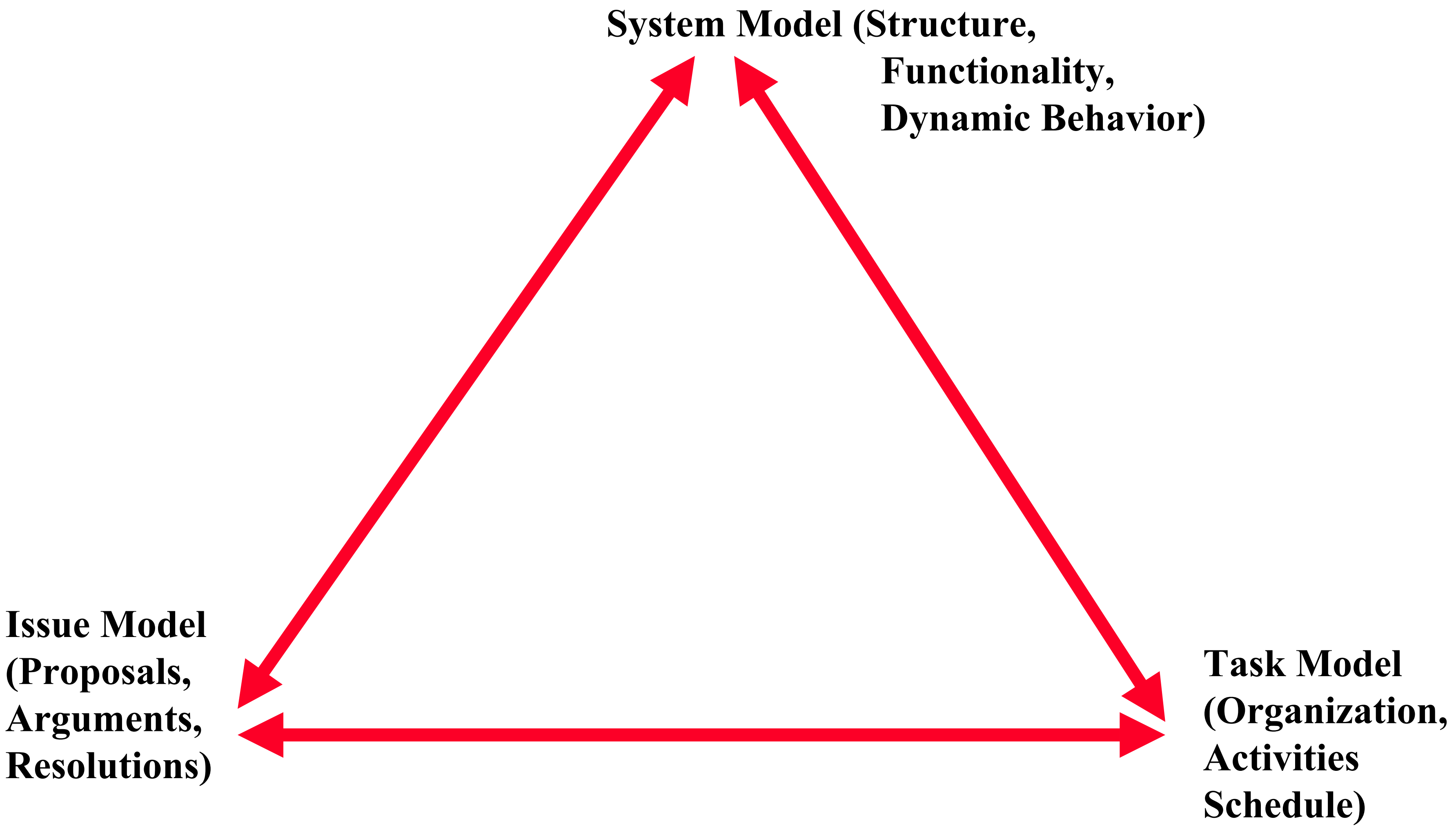
1. Abstraction

- Inherent human limitation to deal with complexity
  - **The 7 +- 2 phenomena**
- Chunking: Group collection of objects
- Ignore unessential details: => Models

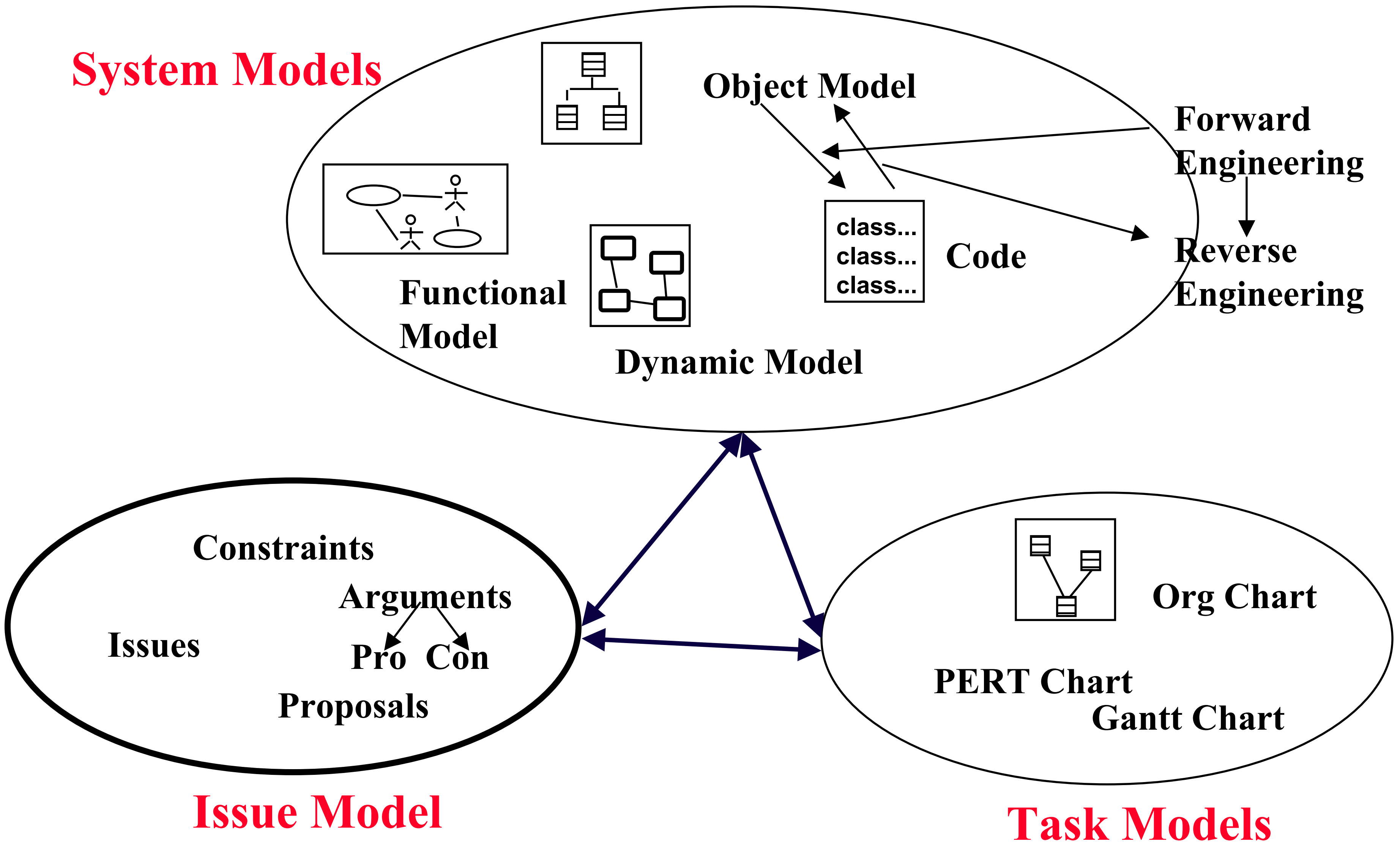
Models are used to provide abstractions

- System Model:
  - **Object Model: What is the structure of the system? What are the objects and how are they related?**
  - **Functional model: What are the functions of the system? How is data flowing through the system?**
  - **Dynamic model: How does the system react to external events? How is the event flow in the system ?**
- Task Model:
  - **PERT Chart: What are the dependencies between the tasks?**
  - **Schedule: How can this be done within the time limit?**
  - **Org Chart: What are the roles in the project or organization?**
- Issues Model:
  - **What are the open and closed issues? What constraints were posed by the client? What resolutions were made?**

Interdependencies of the Models


System Model (Structure, Functionality, Dynamic Behavior)
Issue Model (Proposals, Arguments, Resolutions)
Task Model (Organization, Activities Schedule)

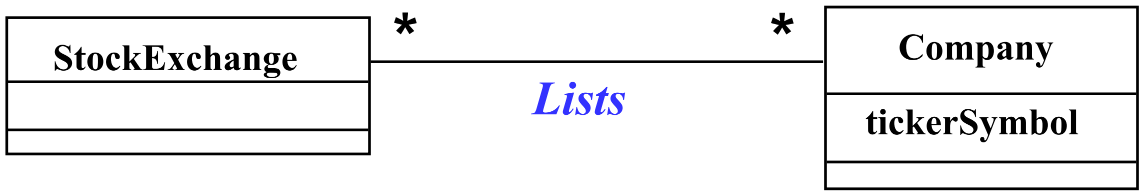
The "Bermuda Triangle" of Modeling



Model-based software Engineering: Code is a derivation of object model

Problem Statement : A stock exchange lists many companies. Each company is identified by a ticker symbol

Analysis phase results in cbject model (UML Class Diagram):



Implementation phase results in code

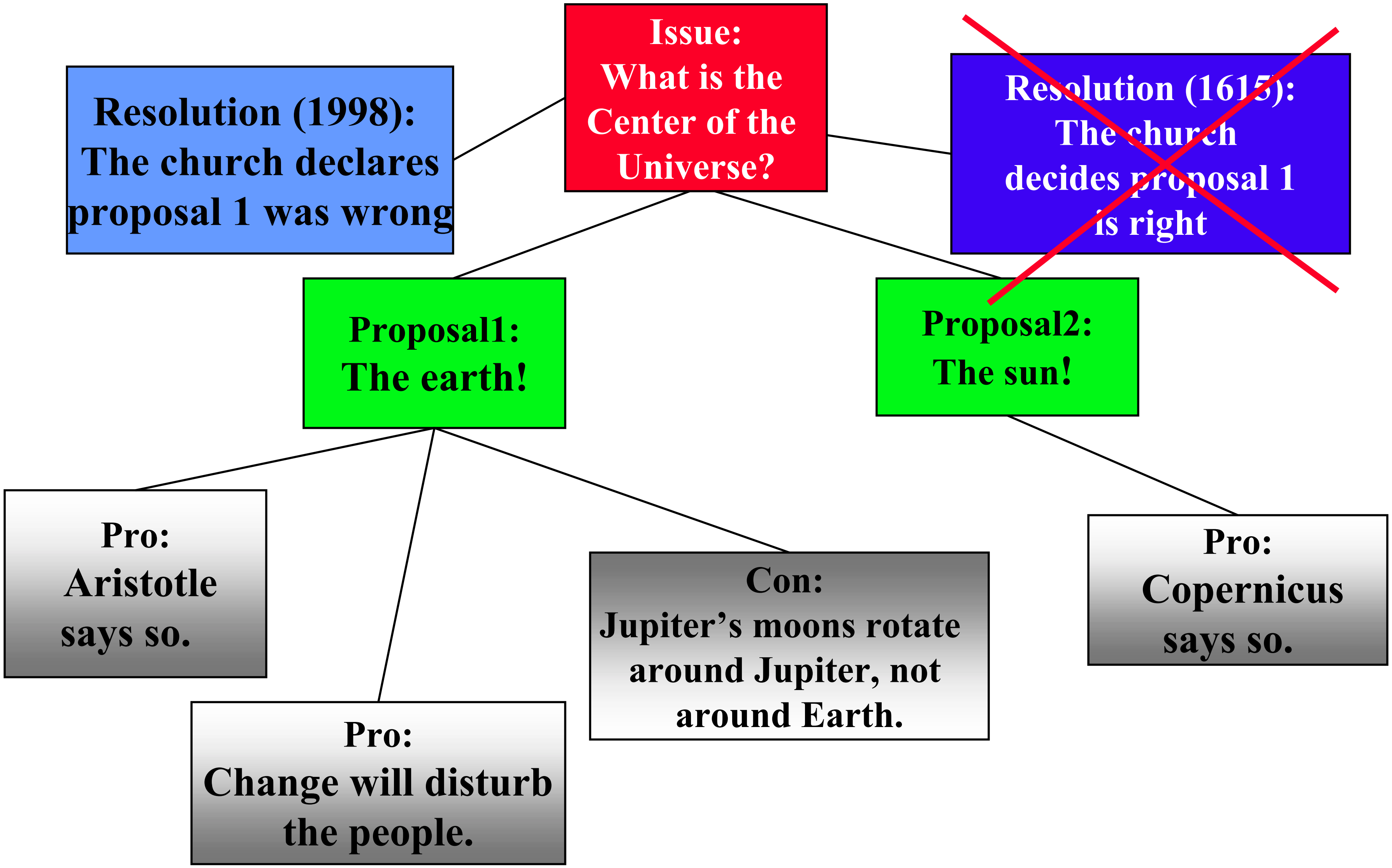
```
public class StockExchange
{

 public Vector m_Company = new Vector();

};


public class Company

{

 public int m_tickerSymbol
 public Vector m_StockExchange = new Vector();

};
```

A good software engineer writes as little code as possible

Example of an Issue: Galileo vs the Church

- What is the center of the Universe?
  - **Church: The earth is the center of the universe. Why? Aristotle says so.**
  - **Galileo: The sun is the center of the universe. Why? Copernicus says so. Also, the Jupiter’s moons rotate round Jupiter, not around Earth.**

Issue-Modeling



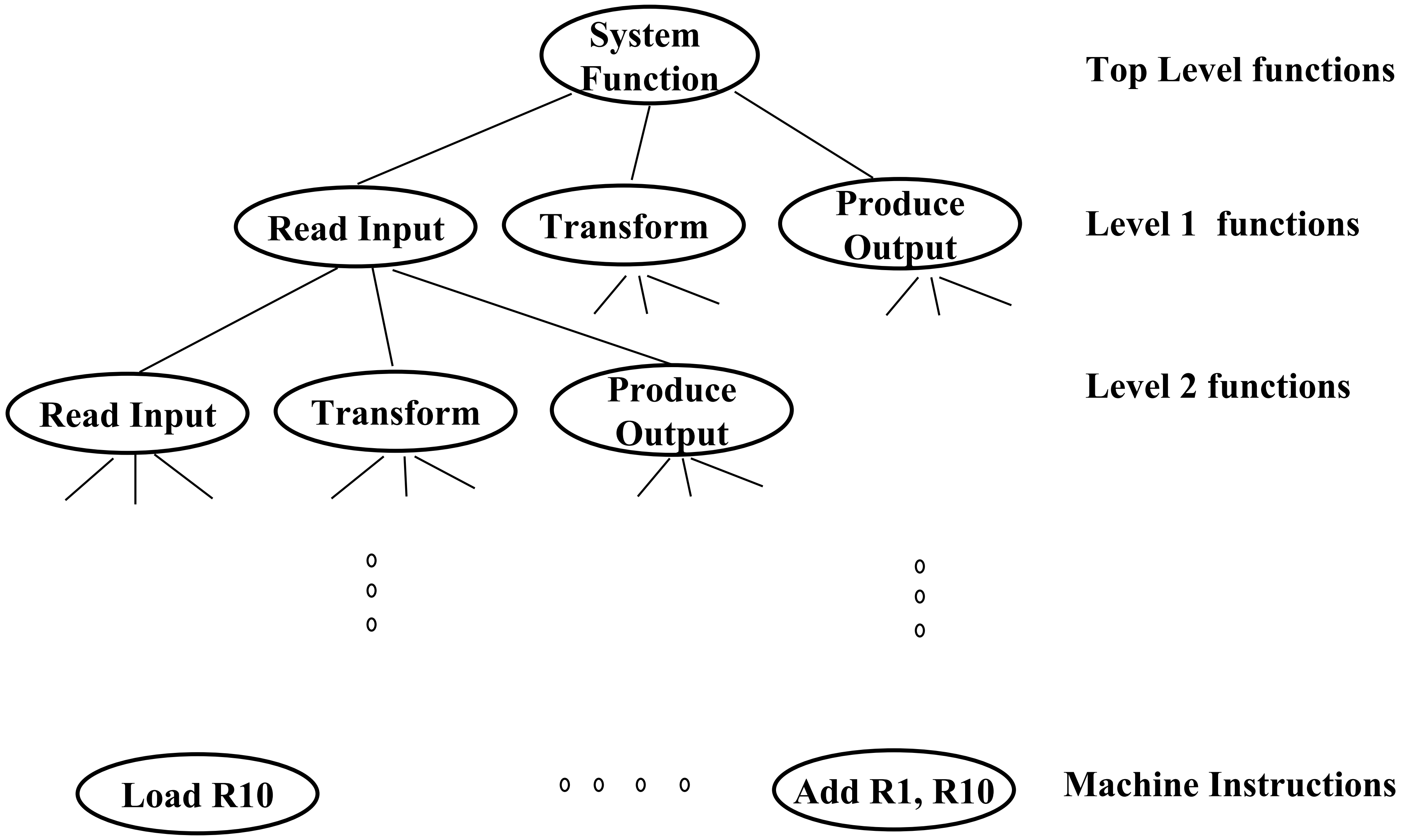
Issue: What is the Center of the Universe?
Resolution (1998): The church declares proposal 1 was wrong
~~Resolution (1615): The church decides proposal 1 is right~~
Proposal1: The earth!
Proposal2: The sun!
Pro: Aristotle says so.
Pro: Change will disturb the people.
Con: Jupiter’s moons rotate around Jupiter, not around Earth.
Pro: Copernicus says so.

2. Decomposition

- A technique used to master complexity ("divide and conquer")
- Functional decomposition
  - **The system is decomposed into modules**
  - **Each module is a major processing step (function) in the application domain**
  - **Modules can be decomposed into smaller modules**
- Object-oriented decomposition
  - **The system is decomposed into classes ("objects")**
  - **Each class is a major abstraction in the application domain**
  - **Classes can be decomposed into smaller classes**

Which decomposition is the right one?

Functional Decomposition

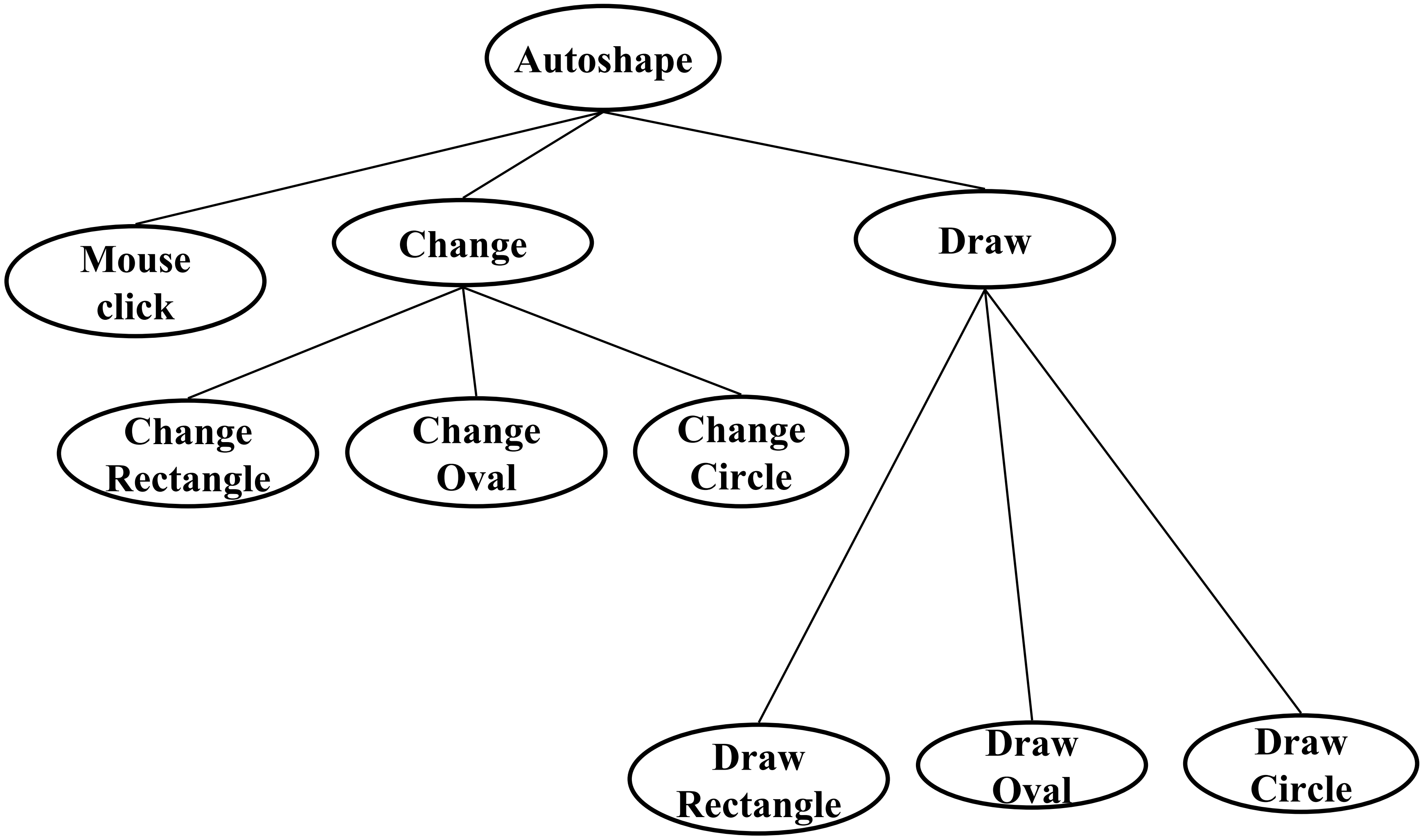


Functional Decomposition

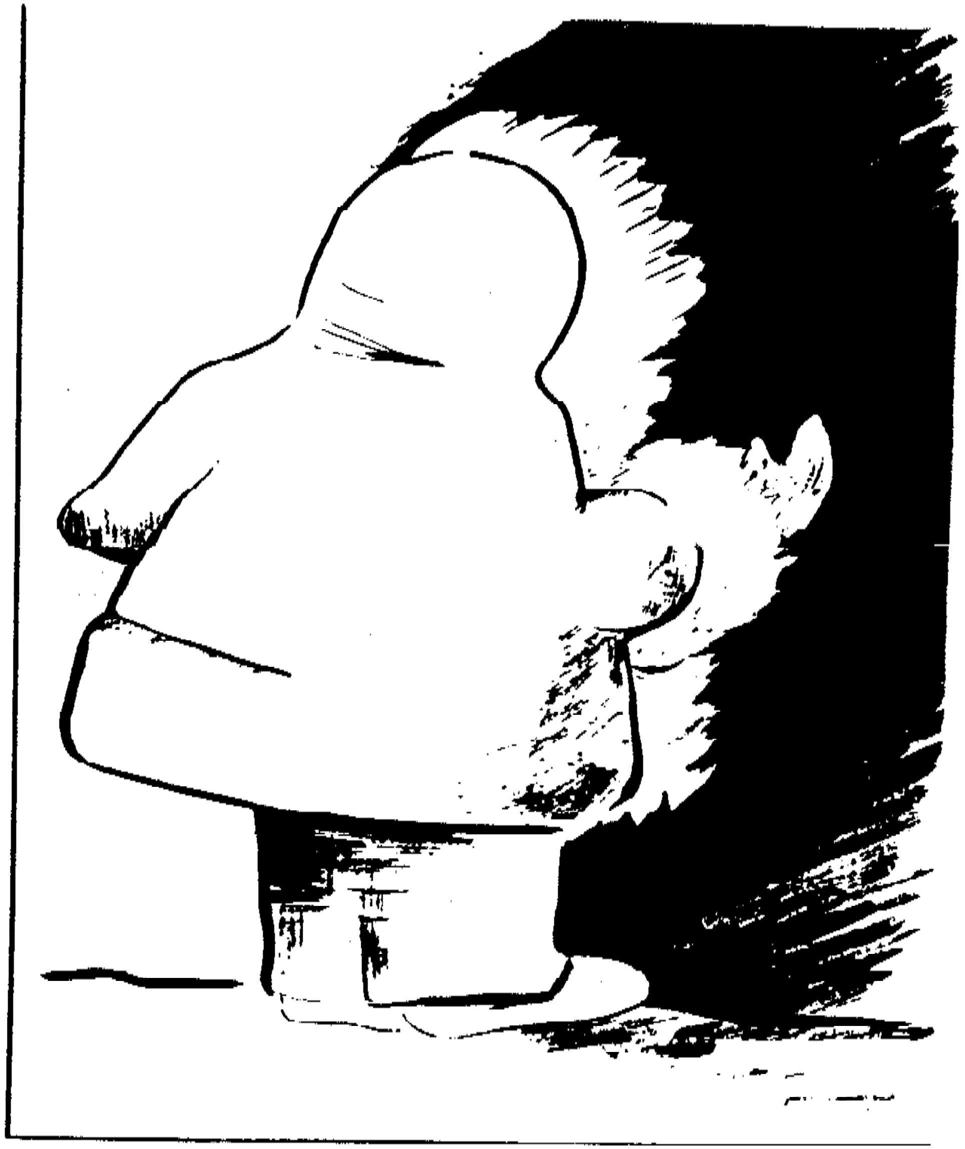
- Functionality is spread all over the system
- Maintainer must understand the whole system to make a single change to the system
- Consequence:
  - **Codes are hard to understand**
  - **Code that is complex and impossible to maintain**
  - **User interface is often awkward and non-intuitive**

- Example: Microsoft Powerpoint's Autoshapes

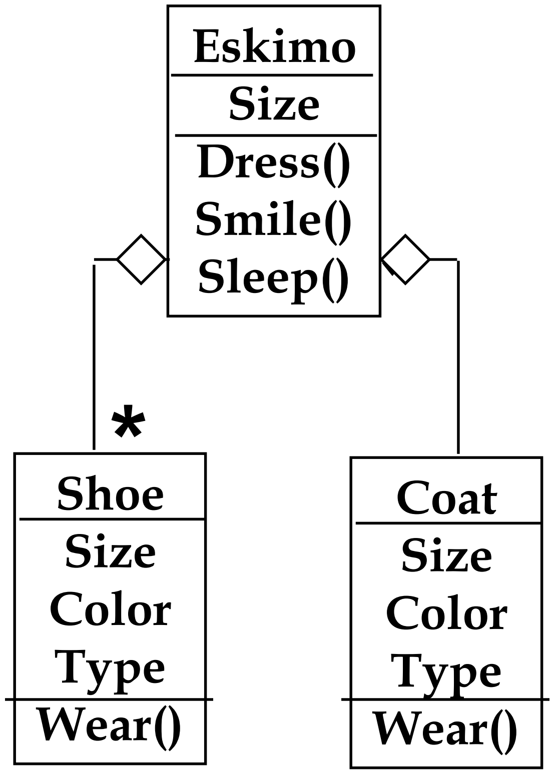
Functional Decomposition: Autoshape



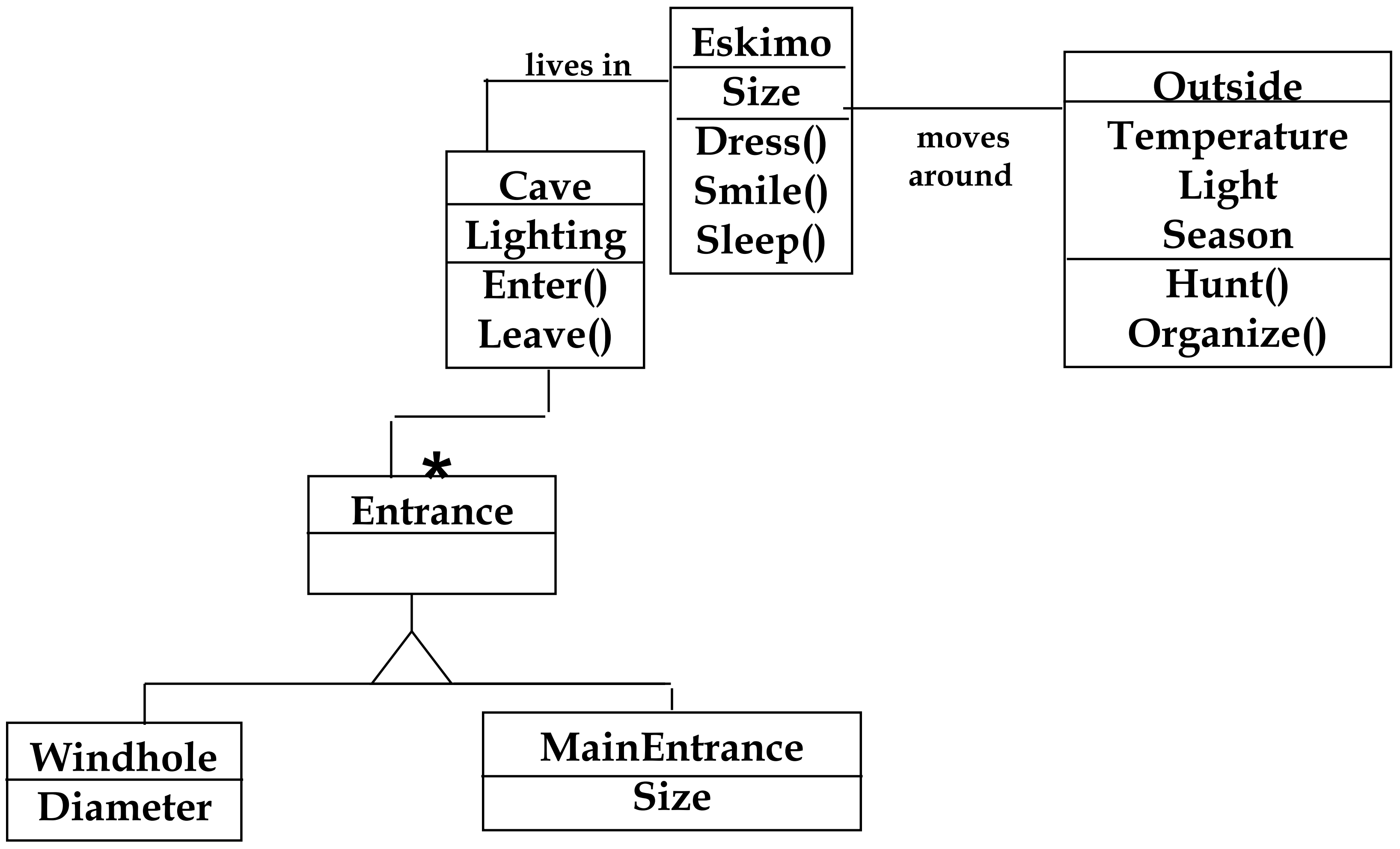
What is This?



Model of an Eskimo

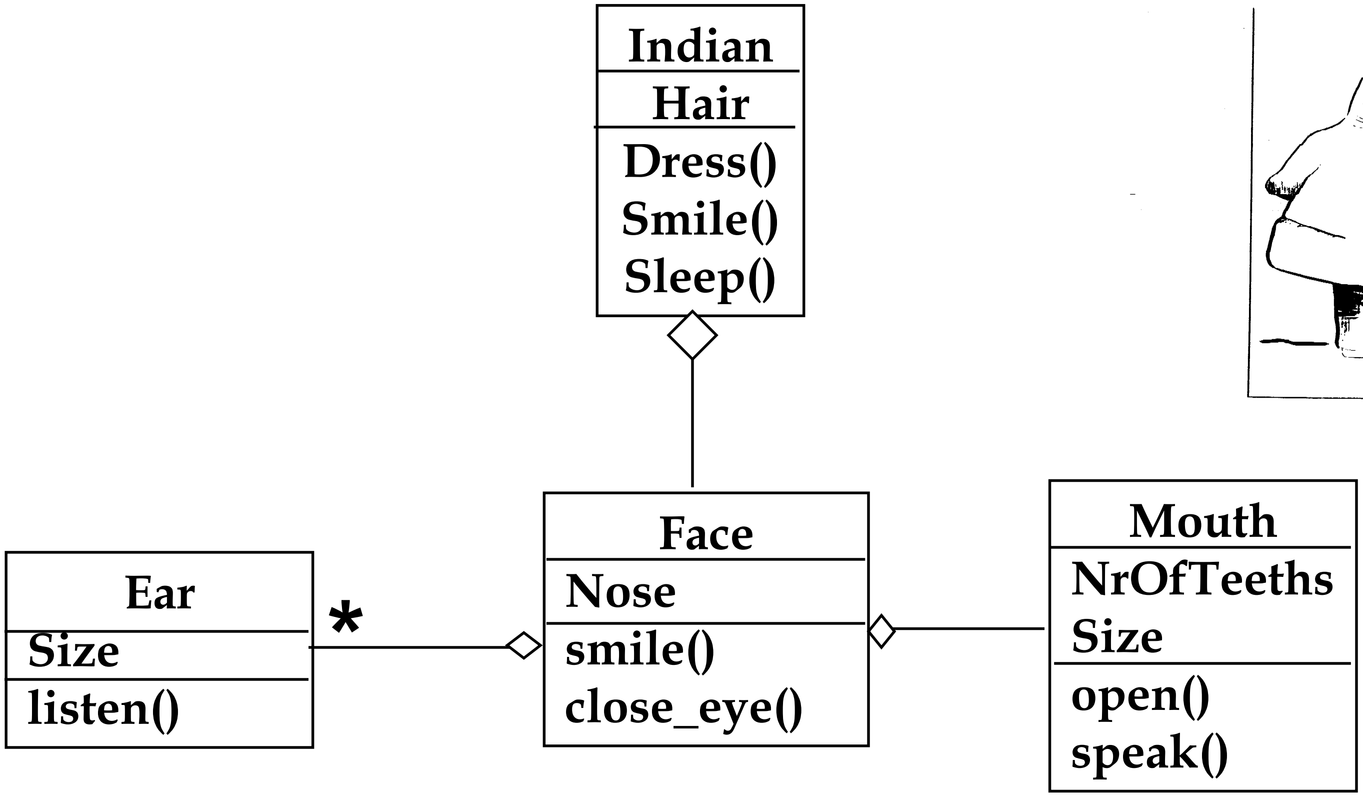


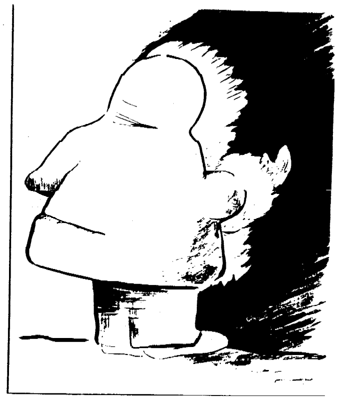
Iterative Modeling then leads to ....



but is it the right model?

Alternative Model: The Head of an Indian

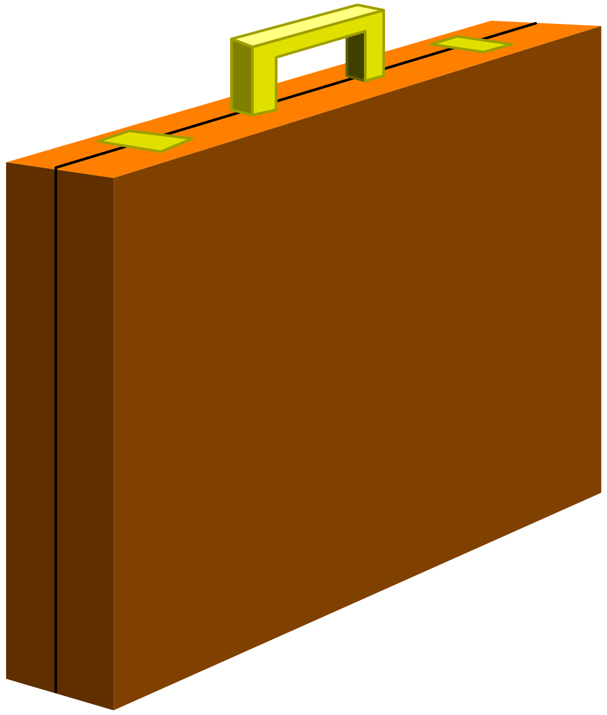




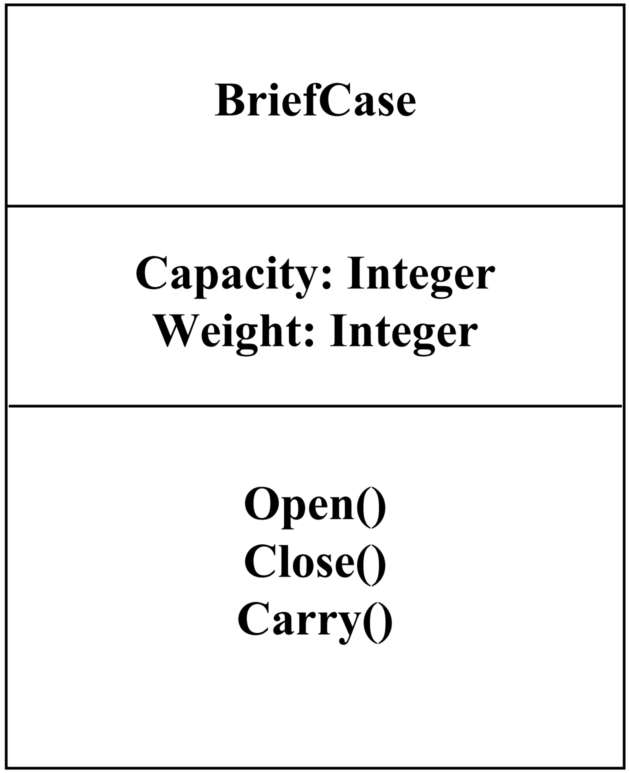
Class Identification

- Class identification is crucial to object-oriented modeling
- Basic assumption:
  1. **We can find the classes for a new software system: We call this Greenfield Engineering**
  2. **We can identify the classes in an existing system: We call this Reengineering**
  3. **We can create a class-based interface to any system: We call this Interface Engineering**
- Why can we do this? Philosophy, science, experimental evidence
- What are the limitations? Depending on the purpose of the system different objects might be found
  - **How can we identify the purpose of a system?**

What is this Thing?



Modeling a Briefcase



A new Use for a Briefcase

BriefCase

Capacity: Integer
Weight: Integer

Open()
Close()
Carry()
SitOnIt()

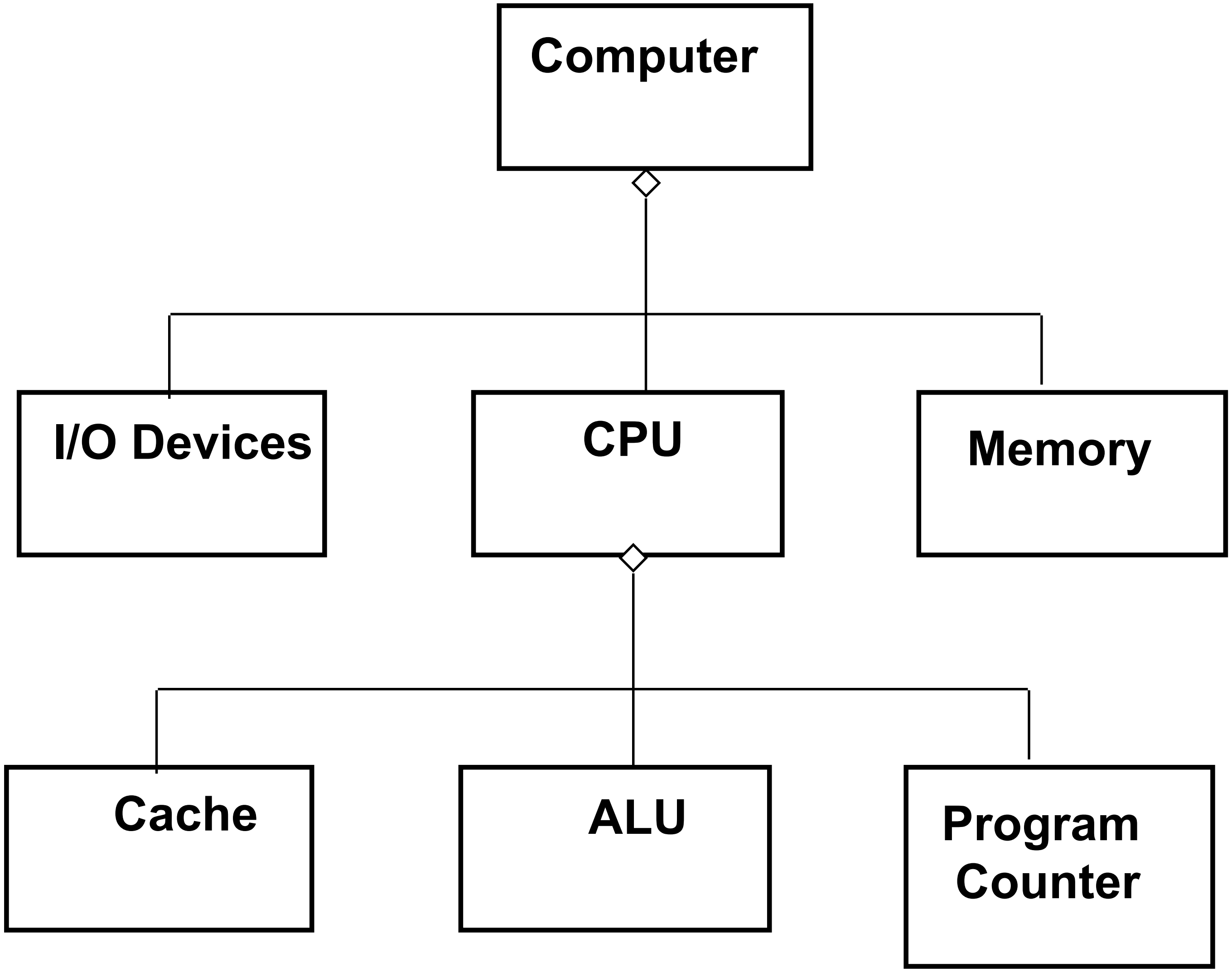
Questions

- Why did we model the thing as “Briefcase”?
- Why did we not model it as a chair?
- What do we do if the SitOnIt() operation is the most frequently used operation?
- The briefcase is only used for sitting on it. It is never opened nor closed.
  - **Is it a “Chair”or a “Briefcase”?**
- How long shall we live with our modeling mistake?

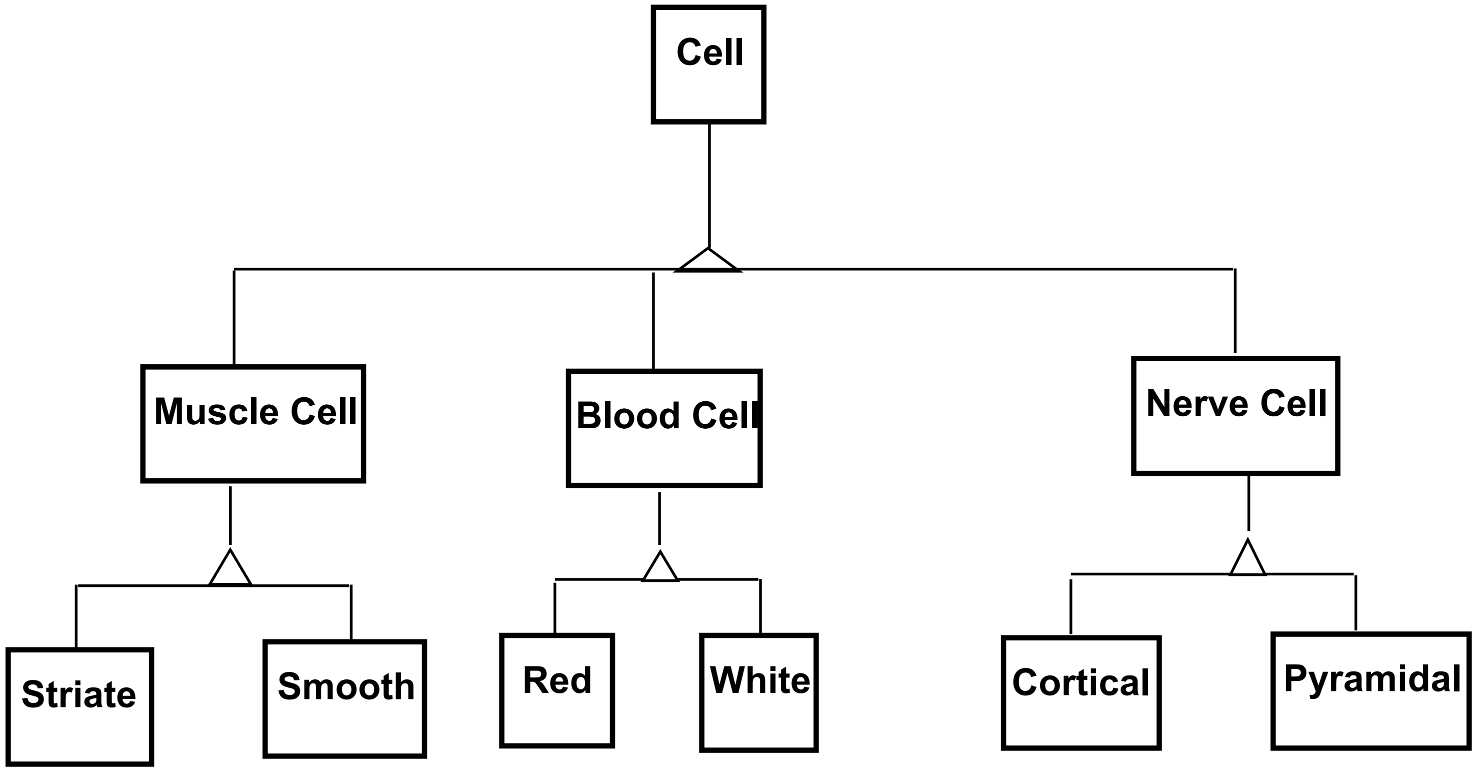
3. Hierarchy

- We got abstractions and decomposition
  - **This leads us to chunks (classes, objects) which we view with object model**
- Another way to deal with complexity is to provide simple relationships between the chunks
- One of the most important relationships is hierarchy
- 2 important hierarchies
  - **"Part of" hierarchy**
  - **"Is-kind-of" hierarchy**

Part of Hierarchy



Is-Kind-of Hierarchy (Taxonomy)



So where are we right now?

- Three ways to deal with complexity:
    - **Abstraction**
    - **Decomposition**
    - **Hierarchy**
- Object-oriented decomposition is a good methodology
    - **Unfortunately, depending on the purpose of the system, different objects can be found**
- How can we do it right?
    - **Many different possibilities**
    - **Our current approach: Start with a description of the functionality (Use case model), then proceed to the object model**
    - **This leads us to the software lifecycle**

Software Lifecycle Definition

- Software lifecycle:
  - **Set of activities and their relationships to each other to support the development of a software system**
- Typical Lifecycle questions:
  - **Which activities should I select for the software project?**
  - **What are the dependencies between activities?**
  - **How should I schedule the activities?**

Reusability

- A good software design solves a specific problem but is general enough to address future problems (for example, changing requirements)
- Experts do not solve every problem from first principles
  - **They reuse solutions that have worked for them in the past**
- Goal for the software engineer:
  - **Design the software to be reusable across application domains and designs**
- How?
  - **Use design patterns and frameworks whenever possible**

Design Patterns and Frameworks

- Design Pattern:
  - **A small set of classes that provide a template solution to a recurring design problem**
  - **Reusable design knowledge on a higher level than datastructures (link lists, binary trees, etc)**
- Framework:
  - **A moderately large set of classes that collaborate to carry out a set of responsibilities in an application domain.**
    - **Examples: User Interface Builder**
- Provide architectural guidance during the design phase
- Provide a foundation for software components industry

Patterns are used by many people

- Chess Master:
  - **Openings**
  - **Middle games**
  - **End games**
- Writer
  - **Tragically Flawed Hero (Macbeth, Hamlet)**
  - **Romantic Novel**
  - **User Manual**
- Architect
  - **Office Building**
  - **Commercial Building**
  - **Private Home**
- Software Engineer
  - **Composite Pattern: A collection of objects needs to be treated like a single object**
  - **Adapter Pattern (Wrapper): Interface to an existing system**
  - **Bridge Pattern: Interface to an existing system, but allow it to be extensible**

Summary

- Software engineering is a problem solving activity
  - **Developing quality software for a complex problem within a limited time while things are changing**
- There are many ways to deal with complexity
  - **Modeling, decomposition, abstraction, hierarchy**
  - **Issue models: Show the negotiation aspects**
  - **System models: Show the technical aspects**
  - **Task models: Show the project management aspects**
  - **Use Patterns: Reduce complexity even further**
- Many ways to do deal with change
  - **Tailor the software lifecycle to deal with changing project conditions**
  - **Use a nonlinear software lifecycle to deal with changing requirements or changing technology**
  - **Provide configuration management to deal with changing entities**