Complexity

Object Orientated Analysis and Design

Benjamin Kenwright

Outline

- Review Object Orientated Programming Concepts (e.g., encapsulation, data abstraction, ..)
- What do we mean by Complexity?
- How do we manage Complexity?
- Design and Manage Complex Systems
- Summary/Discussion

Last Week

- Review Concepts around Object Orientated Analysis and Design
- Revise OOP Principles (e.g., Java)
- Read Chapter 1

Question

- Which of the following mechanisms is/are provided by Object Oriented Language to implement Object Oriented Model?

- A. Encapsulation
- B. Inheritance
- C. Polymorphism
- D. All of the mentioned

Answer

- D. All of the mentioned

Question

- Which of the these is the *functionality* of ‘Encapsulation’?

- A. Binds together code and data
- B. Using single interface for general class of actions
- C. Reduce Complexity
- D. All of the mentioned

Answer

- A. Binds together code and data
- `Encapsulation' acts as protective wrapper that prevents code and data from being accessed by other code defined outside the wrapper

What is the output of this Program?

```
1. class Test {
2.     int a;
3.     public int b;
4.     private int c;
5. }
6. class AcessTest {
7.     public static void main(String args[])
8.     {
9.         Test ob = new Test();
10.         ob.a = 10;
11.         ob.b = 20;
12.         ob.c = 30;
13.         System.out.println(" Output :a, b, and c" + ob.a + " " + ob.b +
    " " + ob.c);
14.     }
15.}
```

- A. Compilation error
- B. Run time error
- C. Output : a, b and c 10 20 30
- D. None of the mentioned

Answer

- A. Compilation error

Explanation: Private members of a class cannot be accessed directly. In the above program, the variable c is a private member of class ‘Test’ and can only be accessed through its methods.

Question

- Which of the following is a mechanism by which object acquires the properties of another object?

- A. Encapsulation
- B. Abstraction
- C. Inheritance
- D. Polymorphism

Answer

- Answer: C. Inheritance

Explanation: ‘Inheritance’ is the mechanism provided by Object Oriented Language, which helps an object to acquire the properties of another object usually child object from parent object.

Why is Software Inherently Complex?

- List the four elements

Why is Software Inherently Complex?

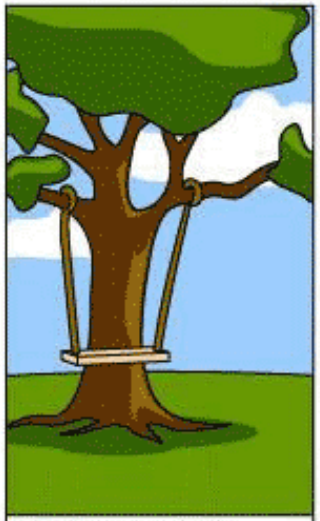
- Inherent Complexity derives from four elements:

1. the complexity of the problem domain
2. the difficulty of managing the development process
3. the flexibility possible through software
4. problems of characterizing the behavior of discrete systems

1. Complexity of the Problem Domain

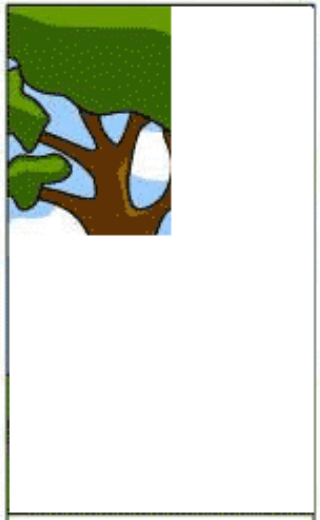
- *Requirements* (functional and non-functional)
  - e.g., electronic system of a multiengine aircraft, a cellular phone switching system, or an autonomous robot
  - Usability, performance, cost, survivability, and reliability
- Users may have only *vague ideas* of what they want in a software system
  - generally lacks expertise in the domain
- Difficult to express requirements
  - e.g., large volumes of texts (*difficult to comprehend, ambiguous ..*)

How the customer explained it



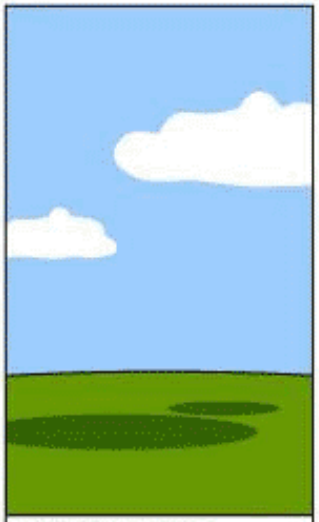
How the Project Leader understood it

How the Analyst designed it



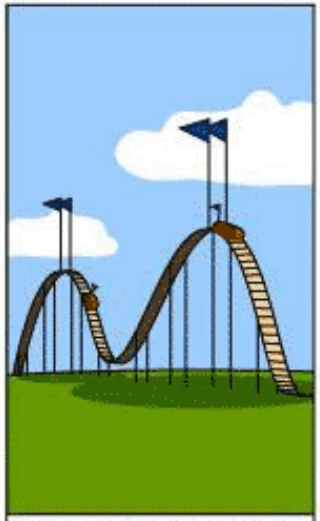
How each developer integrated with others

How QA got the 1st, 2nd, and 3rd build

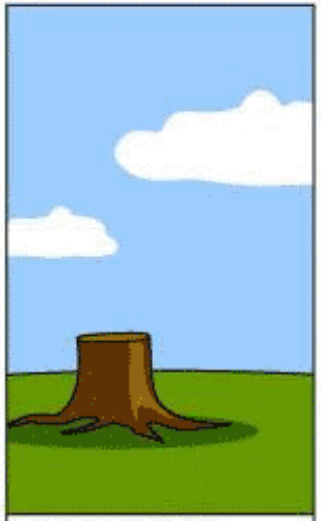


How the project was documented

How the Business Consultant described it



How the customer was billed



How it was supported

What the customer really needed

Complexity of the Problem Domain Cont.

- *Changing* requirements during development
  - early products, such as design documents and prototypes
  - can lead users to better understand and articulate their real needs

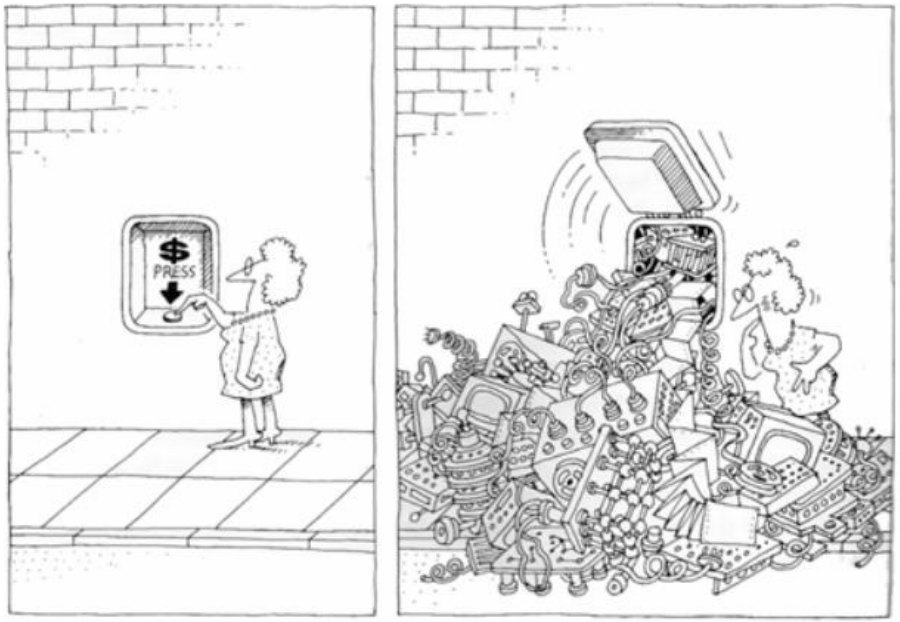
"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

Evolve Over Time

- it is *maintenance* when we correct errors
- it is *evolution* when we respond to changing requirements;
- it is *preservation* when we continue to use extraordinary means to keep an ancient and decaying piece of software in operation

2. Difficulty of Managing the Development Process

- Fundamental task of the software development team is to engineer the *illusion of simplicity*



Cont.

- Strive to write less code
- *Reusing frameworks* of existing designs and code
  - Avoid reinventing wheel
- Sheer *volume of a system's requirements* is sometimes inescapable
- Not unusual today to find systems whose size is measured in hundreds of thousands or even *millions of lines of code*
  - More developers so more complex communication and hence more difficult coordination (location, skillset, attitude, synergy, ..)

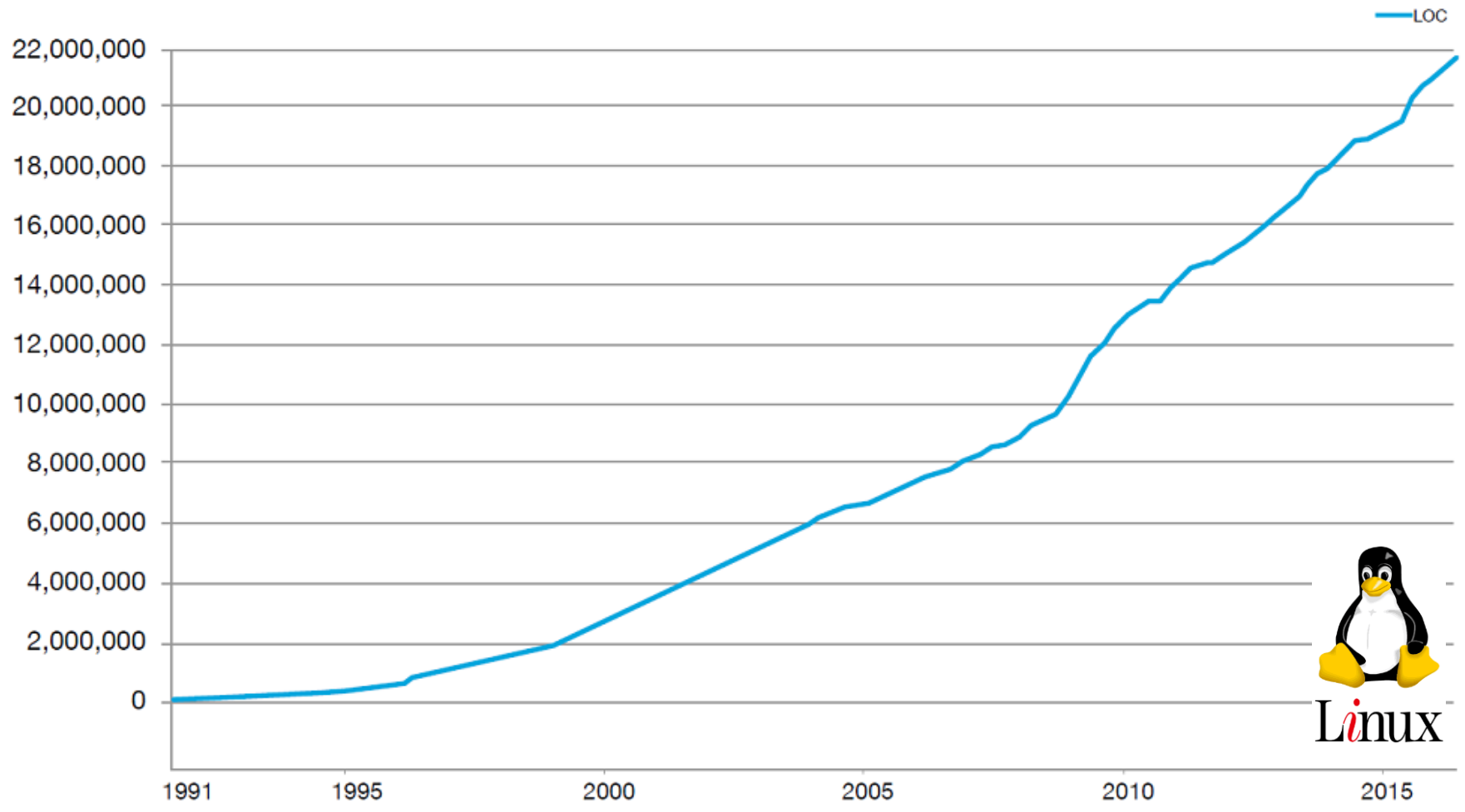
3. Flexibility Possible through Software

- Developer able to express almost any kind of abstraction
- Flexibility incredibly seductive property
- Few uniform rules, codes and standards for quality
  - ▷Makes developing software labor-intensive

4. Problems of Characterizing the Behavior of *Discrete Systems*

- Large application, there may be *hundreds or even thousands of variables and multiple control threads*
- Large systems, there is a combinatorial explosion of *possibilities* (states)
  - Manage this by decomposing the behavior in one part of a system so it has a minimal impact on the behavior in another
- However, worst circumstances, an external event may *corrupt the state of a system* (failed to take into account certain *interactions*)

Total Lines of Code in the Linux Kernel



What are the Five Attributes of a Complex System?

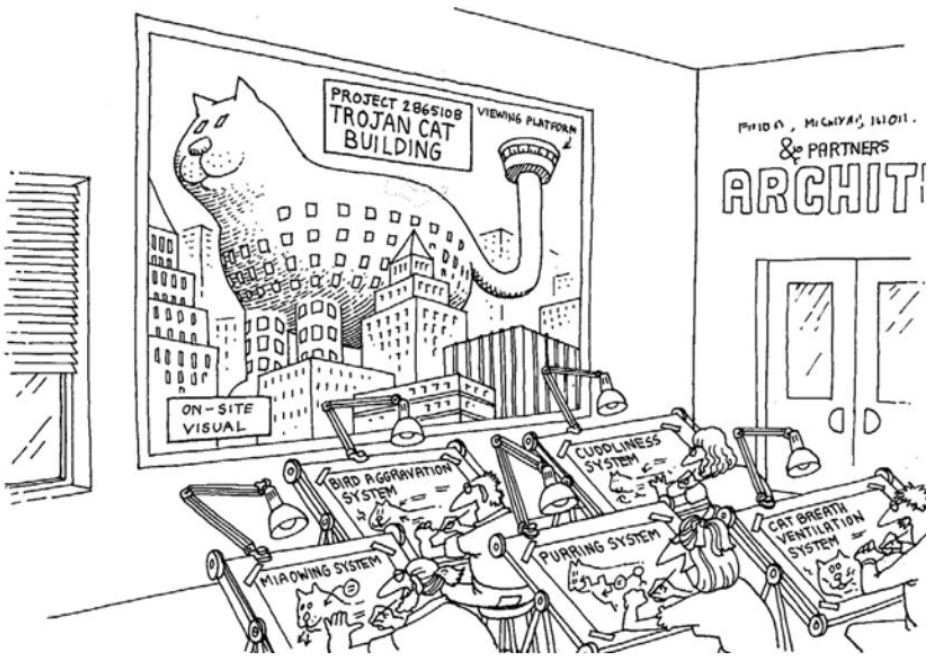
What are the Five Attributes of a Complex System?

- Five attributes common to all complex systems

1. Hierarchic Structure
2. Relative Primitives
3. Separation of Concerns
4. Common Patterns
5. Stable Intermediate Forms

1. Hierarchic Structure

Frequently, complexity takes the form of a hierarchy, whereby a complex system is composed of interrelated *subsystems* that have in turn their *own subsystems*, and so on, until some lowest level of elementary components is reached



2. Relative Primitives

- Components in a system are primitive and relatively arbitrary - largely up to the *discretion of the observer* of the system what they are

3. Separation of Concerns

- Decomposable parts are *not necessarily completely independent*

4. Common Patterns

- Composed of only a *few different kinds* of *subsystems* in various combinations and arrangements

5. Stable Intermediate Forms

- Complex systems will *evolve from simple systems*
- *Never* craft these primitive objects *correctly* the *first time*
- *Improve* them *over time* as we learn more about the real behavior of the system

Question

- Requirements change during development?

- A. True
- B. False

Answer

- A. True

Question

- Complex systems do not evolve from simple systems?

Answer

- A. False

Complex systems will evolve from simple systems

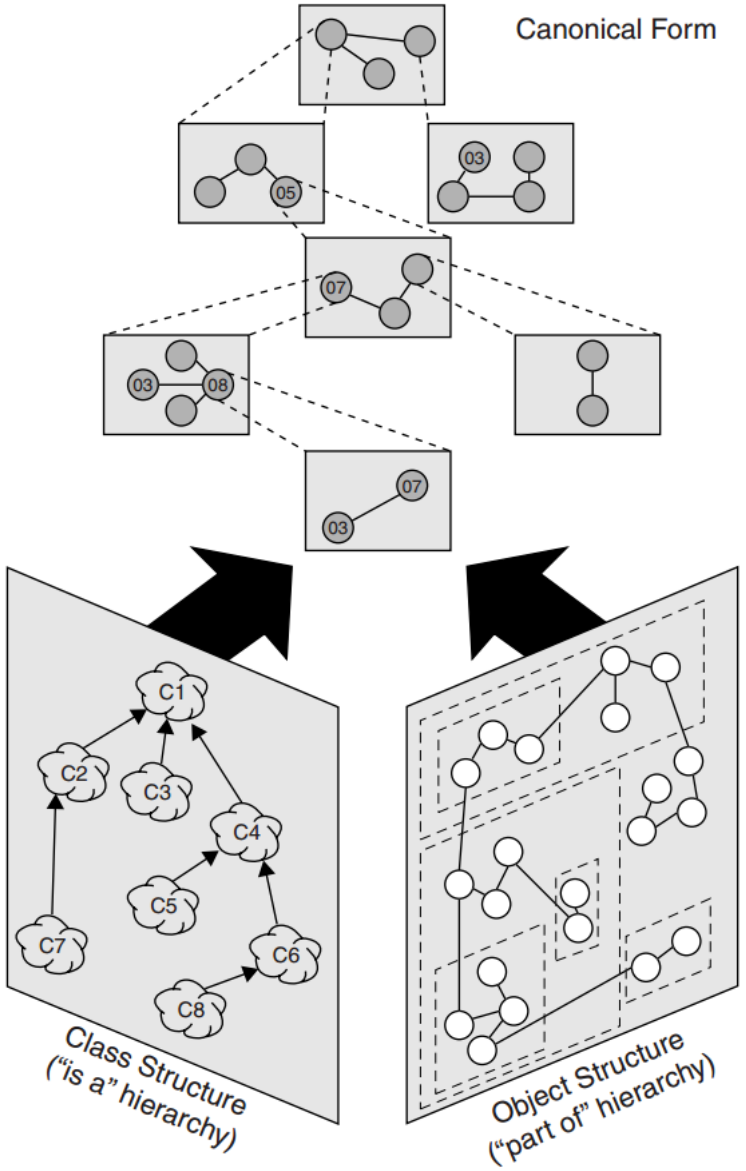
Organized and Disorganized Complexity

- Common abstractions and mechanisms greatly facilitates our understanding of complex systems

Canonical Form of a Complex System

- Canonical Form of a Complex System
  - Different hierarchies are usually present within the same complex system (perspective and requirements)
- Successful complex software systems are those whose designs explicitly encompass well-engineered *class and object structures* and *embody the five attributes* of complex systems

Canonical Form of a Complex System



Limitations of the Human Capacity for Dealing with Complexity

- We must think about many things at once
  - cope with a fairly large, intricate, and sometimes nondeterministic state space
- Maximum number of chunks of information that an individual can simultaneously comprehend is on the order of seven, plus or minus two
- Relates to the capacity of short-term memory and speed to process data (e.g., five seconds to accept a new chunk of information)

Basic Limits on our Ability to Cope with Complexity. How then do we Resolve this Predicament?

Solution

- Order/Logic
- Decomposition
- Structure
- Abstraction/Simplificiations

Objects or Processes

- Complex systems can be viewed by focusing on either *things* or *processes*; there are compelling reasons for applying object-oriented decomposition, in which we view the world as a *meaningful collection of objects* that collaborate to achieve some higher-level behavior

Analysis & Design Methods

- Dozens of design methods have been proposed. However, most methods can be categorized as one of *three kinds*:

Three Design Categories

- Top-down structured design
- Data-driven design
- Object-oriented design

Cont.

- *Top-down structured design* - applies algorithmic decomposition
- *Data-driven design* - mapping system inputs to outputs derives the structure of a software system
- *Object-oriented design* - model software systems as collections of cooperating objects

Question

- What is the maximum number of chunks of information that an individual can simultaneously comprehend?

- A. 5 to 9 Chunks
- B. 3 to 8 Chunks
- C. 1 to 5 Chunks
- D. 8 to 10 Chunks

Answer

- A. 5 to 9 Chunks

Maximum number of chunks of information that an individual can simultaneously comprehend is on the order of seven, plus or minus two

What is the Role of Abstraction?

What is the Role of Abstraction?

- Individual can comprehend only a few chunks of information
- We have an exceptionally powerful technique for dealing with complexity
  - ▷*We abstract from it*
- E.g., Unable to master the entirety of a complex object, we choose to *ignore* its inessential *details*, dealing instead with the *generalized*, idealized *model* of the object

Abstraction (Hierarchical)

- Classifying objects into groups of related abstractions so we can explicitly distinguish common and distinct properties of different objects
  - helps us to master/manage inherent complexity
- Identifying the hierarchies within a complex software system
  - simplify and embody tremendously complicated behavior

What is the Purpose of Design?

What is the purpose of Design?

- Purpose of design is to construct a system that:
  - Satisfies a given (perhaps informal) *functional specification*
  - *Conforms to limitations* of the target medium
  - Meets implicit or explicit *requirements* on performance and resource usage
  - Satisfies implicit or explicit *design criteria* on the form of the artifact
  - Satisfies *restrictions on the design process* itself, such as its length or cost, or the tools available for doing the design

Question

- What are the Five Attributes of a Complex System?

- A. Relative Primitives Separation of Concerns, Hierarchic Structure, Common Patterns, Stable Intermediate Forms
- B. Hierarchic Structure, Common Primitives, Separation of Concerns, Stable Patterns, Dynamic Patterns
- C. Hierarchic Structure, Primitive Structures, Separation Patterns, Stable Forms, Intermediate Attributes

Answer

Five Attributes of a Complex System:

- **A.** Relative Primitives Separation of Concerns, Hierarchic Structure, Common Patterns, Stable Intermediate Forms

Question

- Which of the following is the functionality of ‘Data Abstraction’?

- A. Reduce Complexity
- B. Binds together code and data
- C. Parallelism
- D. None of the mentioned

Answer

- A. Reduce Complexity

Explanation: An essential element of Object Oriented Programming is ‘Data Abstraction’ which means hiding things. Complexity is managed through abstraction.

Summary

- *Complexity* in Object Orientated Analysis and Design
- Why Software is Inherently Complex
- Human Intellectual Capacity
- Engineer the *Illusion* of Simplicity
- Evolve from stable intermediate forms
- Manage complexity through *decomposition, abstraction*, and *hierarchy*
- Focus on either things or processes
  - ▷view the world as a meaningful collection of objects

This Week

- Review Slides
- Read Chapter 2
- Quizzes/Questions

Questions/Discussion