Module 01: Object-Oriented Analysis & Design

Challenges in Software Engineering

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

This presentation uses diagrams, examples and selected texts from *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007) with kind permission of the author

Module Objectives

- Understand why Software is still *developed* and **not** *constructed*
- Understand why Software projects fail
- Take a glimpse of the remedial measures
- KYC: Know Your Course

Module Outline

- Is Software *Engineering*?
  - Evolution of Engineering Domains
- Why Software Projects Fail?
- Software: Development → Construction
- KYC: Know Your Course
  - Course Outline
  - Course Text Book
  - Course Examples
  - Course TAs

Software Engineering

The order of the day is to refer to Software Development as Software Engineering

Is it fair?

Can we really Construct Software as Civil Engineers build bridges?

Engineering: Skills of Construction

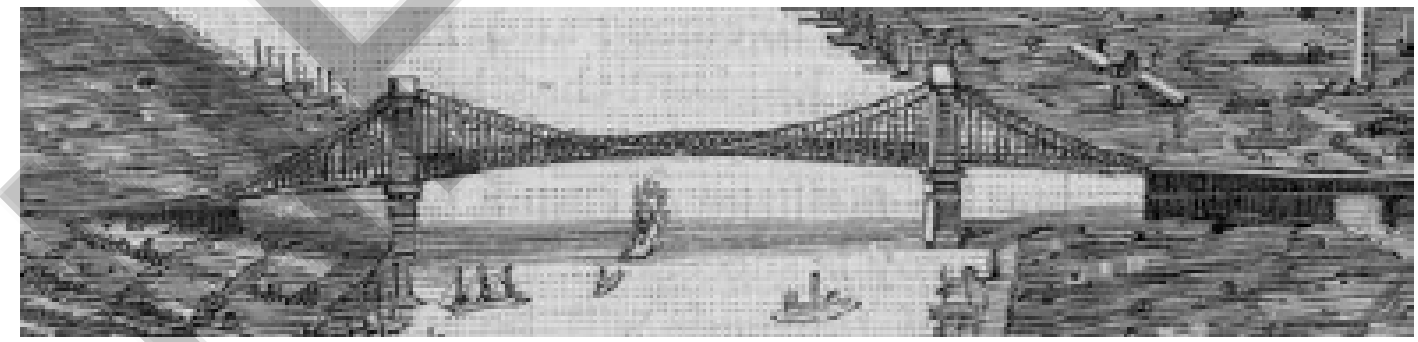
- **Civil Engineering**
  - Construction of Buildings
- **Mechanical Engineering**
  - Construction of Automobiles
- **Electrical Engineering**
  - Construction of Power Plants
- **Software Engineering**
  - *Development of Software*

Evolution of Domains

- Bridges
- Surgery
- Airplanes
- Software

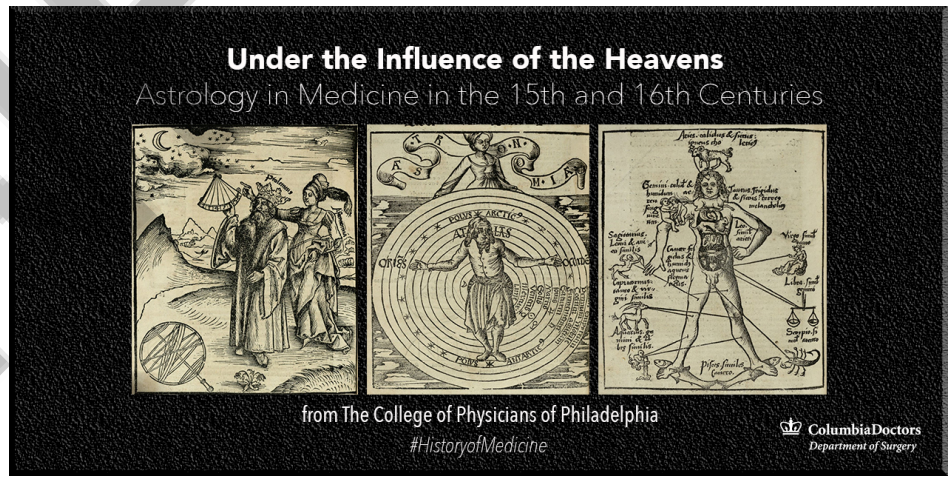
Bridge Construction – Art of Connecting

- Early Wood, Stone – *time immemorial*
- Then Iron, Steel
- Concrete Bridges
- *Constructing a bridge* is different from *innovating a bridge* (with new material, for instance) for the first time
- *Engineers use well established metrics, standards to design bridges* - they do not innovate at this stage



Medical Surgery – Art of Curing

- *Health was thought to be restored by purging, starving, vomiting or bloodletting*
  - Surgeons and barbers specialized in this practice
  - Widely practiced in 18^{th} and 19^{th} century
  - Declared quackery by 1900
- Infection control
  - Survived surgery, died out of infection
  - Germ theory and sterility came only in late 1800's
  - *Strict protocols for Surgery*. Rate of infection $< 2.5\%$



Airplanes – Art of Flying

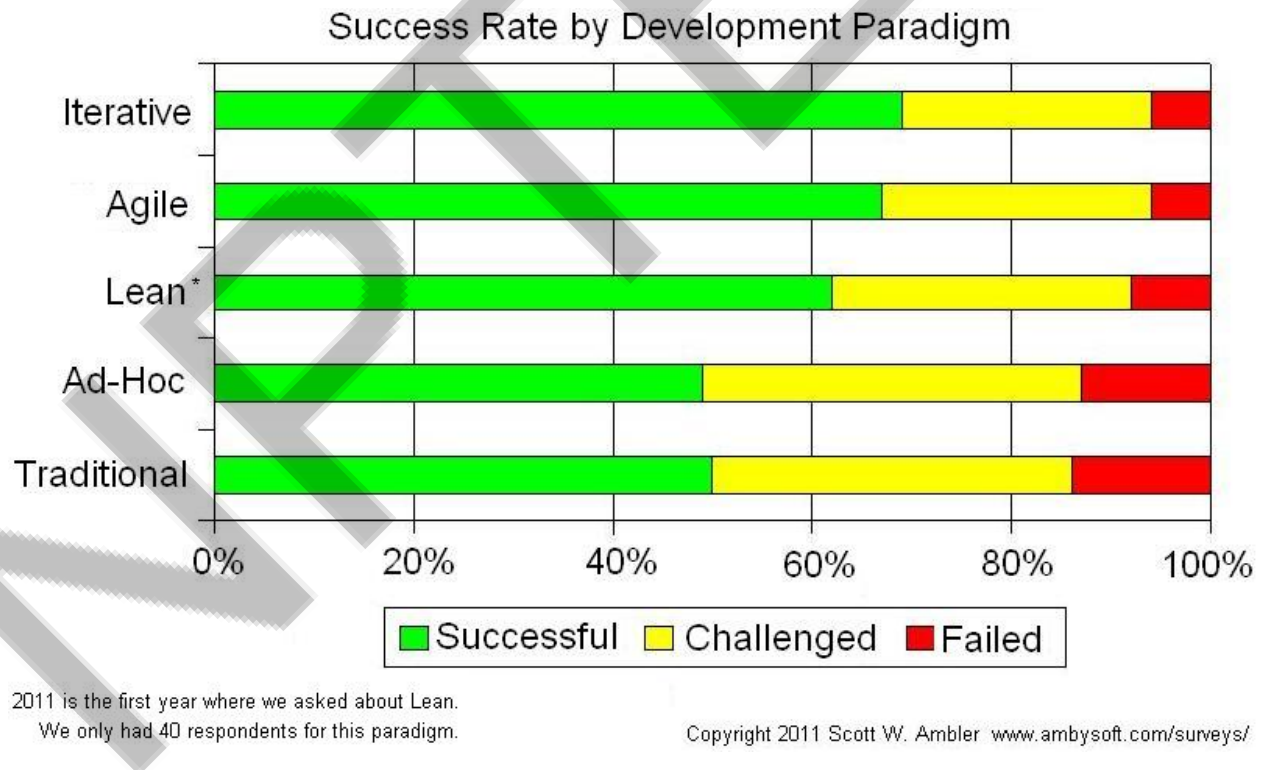
- 400 BC Chinese fly kite aspiring humans to fly
- For centuries, we tried to fly like birds ... disastrous
- Steam powered, hot air
- Gliders, single man
- Engine powered
- 1903 Wright brothers' first flight - 12 seconds, 120 feet, 10 feet altitude
- *Manufacture of airplanes and flying them are strictly regulated by proven practices, standards*

Software Development – Art of Problem Solving

- Vision of Mechanical Computing by Babbage (1810's)
- Relatively nascent field in comparison
  - Incompleteness Theorem (1931) by Kurt Godel
  - Turing machines (1936) by Alan Turing
- Today, machines are getting faster or more powerful
- Yet, is software delivered *successfully*? That is,
  - On time
  - Within budget
  - Feature complete
  - Working (failure free)

Success of Software Projects – Or the Lack of It

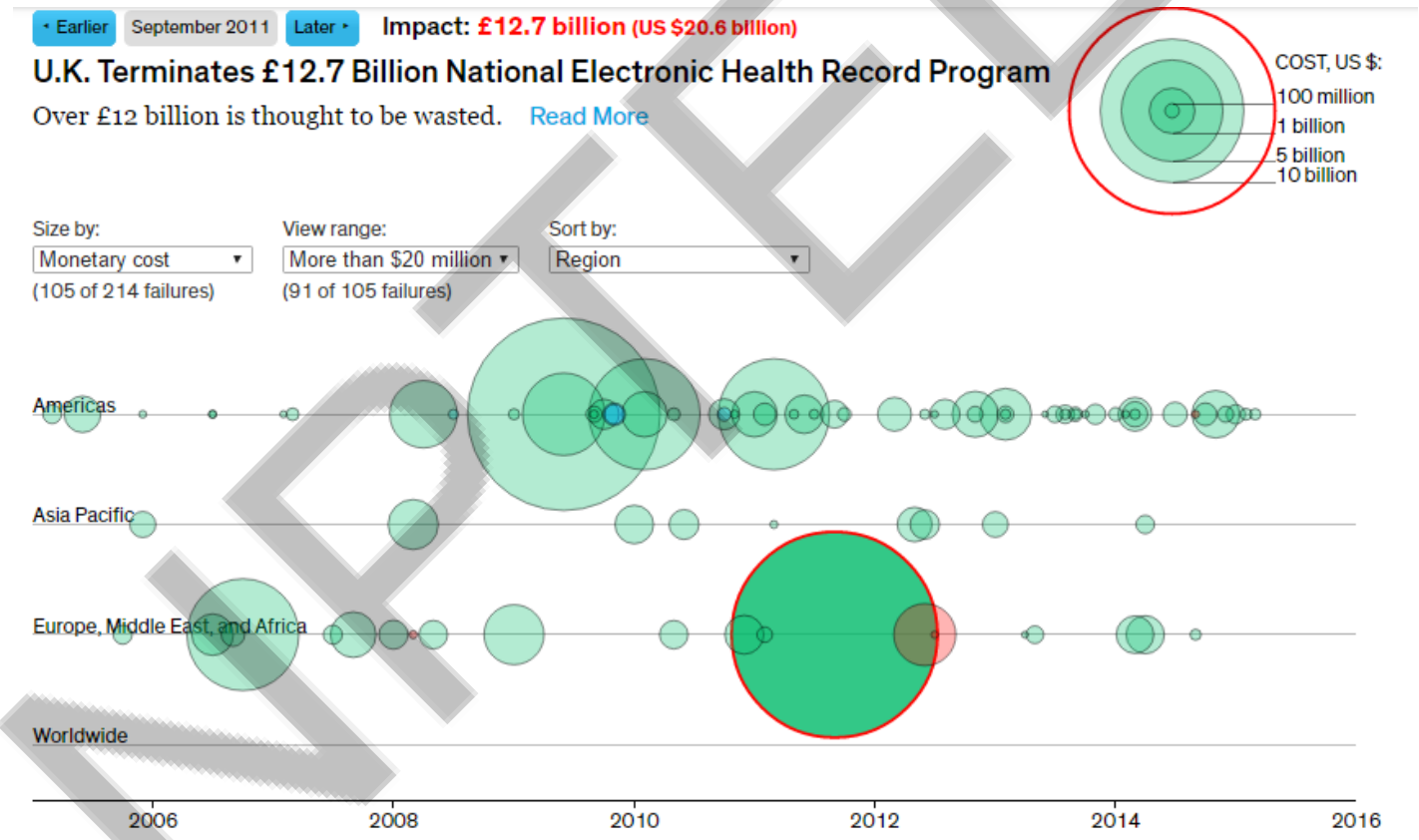
- How successful are we in developing software?
- More than 30% of software projects fail - irrespective of development paradigm



Source: *http://www.ambysoft.com/surveys/success2011.html*

Success of Software Projects – Or the Lack of It

- A decade of failures – 2006 to 2016



Source: *http://spectrum.ieee.org/static/the-staggering-impact-of-it-systems-gone-wrong*

Why Projects Fail? – Complexity is the culprit!

- **Complexity: Changes From Requirements**
  - Customers Learn from the Solution
  - Business Environment and Conditions Change
  - Business Processes are Re-engineered
- **Complexity: Changes From Technology**
  - Tools/Platform Release New Versions
  - Actual Tool/Platform Capabilities may Vary from Plans
- **Complexity: Changes From People**
  - Interactions are Complex
  - Individual Behavior is Unpredictable
- **Complexity: ...**

Software Engineering

- If software engineering like manufacturing or like designing a manufacturing plant?
  - Is it like making another cell phone? Or,
  - Is it like the making of cell phones?
    - It took 37 years for commercialization
- **Manufacturing** must be *predictive*
  - Measure and control quality
  - Measure and control quantity
- **Designing a manufacturing plant** is *creative / innovative*
  - Most software development is innovative process rather than predictive manufacturing
  - Requires great deal of innovation, interaction / communication

Software Development → Software Construction

- Structured Analysis, Modeling, Design, and Implementation
- Adherence to *Good Practices*

Object-Oriented Analysis & Design is a precursor to both

Object-Oriented Analysis and Design: Course Outline

- **Software Complexity**: Understanding the challenges OOAD can address
- **Object Model**: Defining the primitives of the OO paradigm
- **Classes and Objects**: Bringing in the broader perspectives
- **Classes and Objects**: Identification approaches using OOAD
- **Unified Modeling Language**: Understanding UML Diagrams (3 weeks)
- **OOAD Case Studies**: Applying OOAD in different contexts

Text Book

Object-Oriented Analysis and Design – With Applications
by *Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston*, 3rd Ed., 2007

Example Systems for OOAD

- Running Example in Lectures
  - **Leave Management System (LMS)**
  - Examples from Booch's OOAD Book
- Running Example in Assignments
  - (Students') **Assignment Management System (AMS)**
  - Examples from Booch's OOAD Book
- Examples for Complete Workout
  - (Indian) **Postal Management System (PMS)**:
  - (Newspaper) **Story Management System (SMS)**
  - (Rental) **Car Management Systems (CMS)**
  - Examples from Booch's OOAD Book

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655

Module Summary

- *Software Engineering* is closer to *Development* than *Construction*
- Software Projects fails for Complexity, Change
- **OOAD** can take software from *Development* to *Construction*

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655

Module 02: Object-Oriented Analysis & Design

Complexity of Software

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

This presentation uses diagrams, examples and selected texts from *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007) with kind permission of the author

Module Objectives

- Understand why Software is Complex
- Study the elements of Complexity

Module Outline

- Defining Software Complexity
- Four Elements of Complexity
  - The Complexity of the Problem Domain
  - The Difficulty of Managing the Development Process
  - The Flexibility Possible through Software
  - The Problems of Characterizing the Behavior of Discrete Systems

Module 02: Lecture 02

- Defining Software Complexity
- Four Elements of Complexity
  - The Complexity of the Problem Domain
  - The Difficulty of Managing the Development Process

What do we refer to as *Software*?

- Personal or Limited-use Software
- Industrial-Strength Software

Personal or Limited-use Software

- Limited set of behaviors
- Not very complex
- Specified, constructed, maintained, and used by the same person or a small group
  - Amateur programmer, or
  - Professional developer working in isolation
- Tend to have short life span
- Can be thrown away and replaced with entirely new software rather than attempt to reuse them, repair them, or extend their functionality
- Generally more tedious than difficult to develop
- Techniques to develop them is of little interest here

Industrial-Strength Software

- Exhibit a very rich set of behaviors – *reactive systems that drive or are driven by events in the physical world*
- Works with scarce resources – *time, space, power, ...*
- Maintains the integrity of millions of records while allowing concurrent updates and queries – *airline booking*
- Commands and controls of real-world entities – *routing of air or railway traffic*
- Tend to have a long life span
- Depended by many users over time on proper functioning
- Usually based on frameworks that simplify the creation of domain-specific applications

Complexity of Industrial-Strength Software systems exceeds the human intellectual capacity

Software is Inherently Complex

Inherent complexity derives from four elements:

- The Complexity of the Problem Domain
- The Difficulty of Managing the Development Process
- The Flexibility Possible through Software
- The Problems of Characterizing the Behavior of Discrete Systems

The Complexity of the Problem Domain

- **Domains are difficult to understand**. For example:
  - Electronic system of a multi-engine aircraft
  - Merchant Shipping
  - Online Trading and Reconciliation
- **Functional Requirements** are
  - *Complex to master*
  - Often are competing, even contradictory
- **Non-Functional Requirements** (*usability, performance, cost, survivability, and reliability*) are
  - *Often Implicit*
  - Difficult to justify in budget

The Complexity of the Problem Domain

- **Communication Gap** between *Users* and *Developers*
  - Users *cannot express*; developers *cannot understand*
  - *Lack of expertise* across domains
  - *Different perspectives* on the nature of the problem leading to *different assumptions* regarding the nature of the solution
  - *Few instruments* to precisely capture requirements – large texts with some drawings is all we have
  - Leads to *External Complexity*
- **Changing / Evolving Requirements** during Development
  - Early products lead to *better understanding of needs* by the users
  - Developers get *enlightened through the process of development*

The Complexity of the Problem Domain

- **Large Capital Investments**
  - Make *scrapping of projects unfeasible* with changing requirements
  - Lead to *inordinate percentage of software preservation*

Maintenance	*Correct errors*
Evolution	*Respond to changing requirements*
Preservation	*Use extraordinary means to keep an ancient and decaying piece of software in operation*

The Difficulty of Managing the Development Process

- Create **Illusion of Simplicity** to hide external complexity. Hence code bases get large
- Code bases are **large in size** (LOC: 10^5 to 10^6 & more)
  - In spite of the *use of smart languages*
  - In spite of the *reuse of designs and codes*
  - Goes well beyond the comprehension of a single (or small group of) individual/s – even with meaningful decomposition. Hence, we need teams
- Challenges associated with **team development**
  - Large code bases mean more teams with *more developers in more geographies*
  - More developers means *more complex communication* and hence *more difficult coordination*
  - Key management challenge is to maintain a *unity and integrity of design*

Module 02: End of Lecture 02

- Defining Software Complexity
- Four Elements of Complexity
  - The Complexity of the Problem Domain
  - The Difficulty of Managing the Development Process

Module 02: Lecture 03

- Four Elements of Complexity
  - The Flexibility Possible through Software
  - The Problems of Characterizing the Behavior of Discrete Systems

The Flexibility Possible through Software

Civil Construction	Software Development
• **Sources its material** (timber, steel, cement, glass, etc.) from other vendors	• Often **builds the components** within team – *because components are also software and can be built*
• Has **uniform building codes** and standards for the quality of raw materials	• **Few standards** exist for reusable components
• Rarely would a builder think about **adding a new sub-basement** to an existing 100-story building	• Amazingly, users of software systems rarely think twice about **asking for equivalent changes**

The Problems of Characterizing the Behavior of Discrete Systems

- Software is built and executed on digital computers – Hence, they are **Discrete Systems**
- A large application would have
  - hundreds or even thousands of variables
  - more than one (often several) thread of control
- A state is:
  - Entire collection of variables
  - The current values of variables
  - The current address of each process
  - The calling stack of each process

A Software has finite number of states. Yet, many of these the states are often intractable and influenced by external factors

Module Summary

- Understood the difference between Limited-use and Industry-strength software
- Studied the four elements of Software Complexity

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655

Module 03: Object-Oriented Analysis & Design

Structure and Attributes of a Complex System

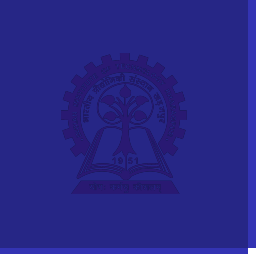
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

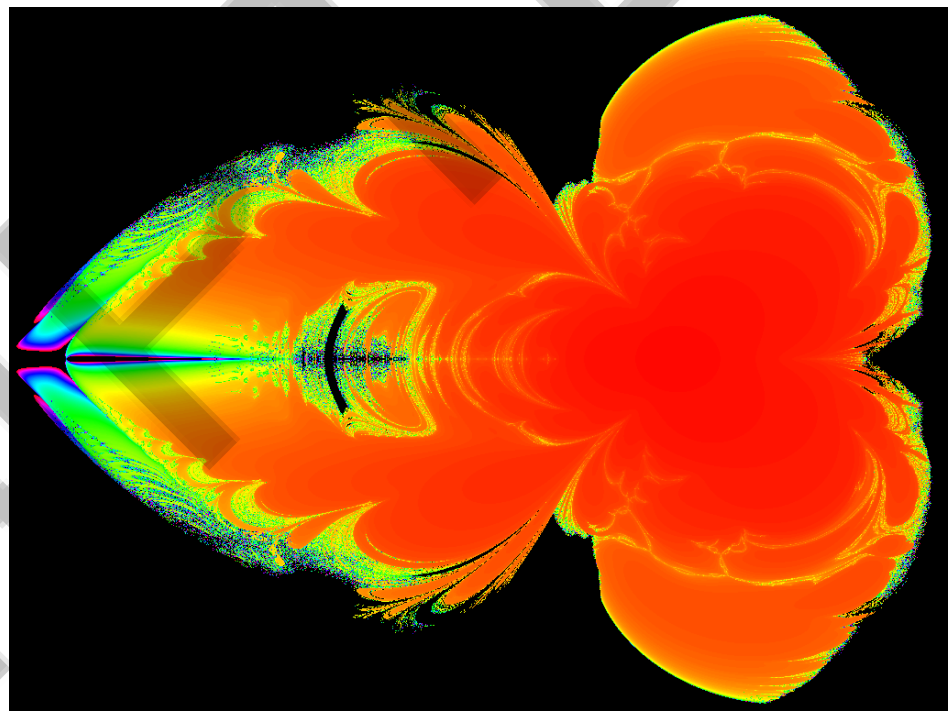
Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

This presentation uses diagrams, examples and selected texts from *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007) with kind permission of the author



Module Objectives

- Understand the Structure of Complex Systems with examples of Man-made, Natural, and Social Systems
- Understand the Attributes of Complex Systems



Module Outline

- Structure of Complex Systems – Examples
  - Personal Computer
  - Plants
  - Education System in India
- Attributes of a Complex System
  - Hierarchic Structure
  - Relative Primitives
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms

Module 03: Lecture 04

- Structure of Complex Systems – Examples
  - Personal Computer
  - Plants
  - Education System in India
- Attributes of a Complex System
  - Hierarchic Structure

The Structure of a Personal Computer

- Central Processing Unit (CPU) – *Executes programs*
  - Arithmetic & Logic Unit (ALU)
    - Registers
      - NAND gate
        - CMOS Gate
        - Interconnect
      - Inverter
    - Control Logic
  - Primary Memory
  - Bus – peripheral devices
- Hard disk drive – *Persistent storage for data*
- Monitor – *Outputs data*
- Keyboard – *Inputs data*
- Secondary storage device (DVD drive / USB) – *Removable storage for data*

Building Blocks

- Personal Computer
  - Gates – NAND, Inverter, etc.
  - Interconnections

Complex Systems are Hierarchic

- Each Level of Hierarchy represents a Layer of Abstraction
- Each Layer
  - Is built on top of other layers: CMOS Gates → NAND Gates→ Registers
  - (in turn) Supports other layers: CMOS Gates → NAND Gates→ Registers
  - Is independently understandable: CPU
  - Works independently with clear separation of concerns: ALU, Memory
- Common services / properties are shared across Layers: Same power-tree feeds the components of CPU
- Layers together show **Emergent Behavior**

  Behavior of the whole is greater than the sum of its parts
- Systems demonstrate cross-domain commonality: Cars have processors, memory, display

The Structure of Plants

- Roots – *Absorb water and minerals from the soil*
  - Branch Roots
    - Collection of Cells
      - Chloroplasts
      - Nucleus
  - Root Hairs
    - Collection of Cells ...
  - Root Apex
  - Root Cap
- Stems – *Transport raw materials from roots up to leaves*
- Leaves – *Use the water and minerals for photosynthesis*
  - Epidermis
  - Mesophyll
  - Vascular Tissue

Building Blocks

- Plants
  - Cells – different types
  - Vessels

Complex Systems are Hierarchic

- Each Level of Hierarchy represents a Layer of Abstraction
- Each Layer
  - Is built on top of other layers: Cells → Branch Roots → Roots
  - (in turn) Supports other layers: Cells → Branch Roots → Roots
  - Is independently understandable: Leaf
  - Works independently with clear separation of concerns: Roots, Leaves
- Common services / properties are shared across Layers: Oxygen supply to Roots, Stems, & Leaves
- Layers together show **Emergent Behavior**

  Behavior of the whole is greater than the sum of its parts
- Systems demonstrate cross-domain commonality: Cells are constituents of plants & animals

The Structure of Education System in India

- CFTIs – IIT, IIM, NIT, IISER, etc.
  - Departments
    - Students
    - Teachers
    - Staffs
  - Library
- UGC / AICTE – *Regulates higher education*
  - Universities
    - Colleges
      - Departments
        - Students ...
- CBSE / ICSE – *Regulates school education*
  - Schools
    - Students ...
    - Play Ground

Building Blocks

- Education System
  - Knowledge Delivery – Contact, Remote, MOOCs
  - Examination – Written, Oral, Practical
  - Admission
  - Graduation

Complex Systems are Hierarchic

- Each Level of Hierarchy represents a Layer of Abstraction
- Each Layer
  - Is built on top of other layers: Departments → Colleges → Universities
  - (in turn) Supports other layers: Departments → Colleges → Universities
  - Is independently understandable: College
  - Works independently with clear separation of concerns: Departments, Library
- Common services / properties are shared across Layers: NKN is shared between institutes; Departments use the same telephone system
- Layers together show **Emergent Behavior**

  Behavior of the whole is greater than the sum of its parts
- Systems demonstrate cross-domain commonality: Similar Leave policy for Teachers & Staff of PSUs

Building Blocks – RECAP

- Personal Computer
  - Gates – NAND, Inverter, etc.
  - Interconnections
- Plants
  - Cells – different types
  - Vessels
- Education System
  - Knowledge Delivery – Contact, Remote, MOOCs
  - Examination – Written, Oral, Practical
  - Admission
  - Graduation

Complex Systems are Hierarchic – RECAP

- Each Level of Hierarchy represents a Layer of Abstraction
- Each Layer
  - Is built on top of other layers: CMOS Gates → NAND Gates→ Registers; Cells → Branch Roots → Roots; Departments → Colleges → Universities
  - (in turn) Supports other layers: CMOS Gates → NAND Gates→ Registers; Cells → Branch Roots → Roots; Departments → Colleges → Universities
  - Is independently understandable: CPU, Leaf, College
  - Works independently with clear separation of concerns: ALU, Memory; Roots, Leaves; Departments, Library
- Common services / properties are shared across Layers: Same power-tree feeds the components of CPU; Oxygen supply to Roots, Stems, & Leaves; NKN is shared between institutes; Departments use the same telephone system
- Layers together show **Emergent Behavior**

  Behavior of the whole is greater than the sum of its parts
- Systems demonstrate cross-domain commonality: Cars have processors, memory, display; Cells are constituents of plants & animals; Similar Leave policy for Teachers & Staff of PSUs; Laws of conservation of momentum apply equally in astronomy (macro) and nuclear physics (micro)

Complex Systems are Hierarchic – RECAP

- Each Level of Hierarchy represents a Layer of Abstraction
- Each Layer
  - Is built on top of other layers
  - (in turn) Supports other layers
  - Is independently understandable
  - Works independently with clear separation of concerns
- Common services / properties are shared across Layers
- Layers together show **Emergent Behavior**

  Behavior of the whole is greater than the sum of its parts
- Systems demonstrate cross-domain commonality

The Five Attributes of a Complex System

- Hierarchic Structure
- Relative Primitives
- Separation of Concerns
- Common Patterns
- Stable Intermediate Forms

Hierarchic Structure

- All **systems** are composed of **interrelated sub-systems**
- Sub-systems are composed of sub-sub-systems, and so on
- Lowest level sub-systems are composed of **elementary components**
- All systems are parts of larger systems
- The value added by a system must come from the relationships between the parts, not from the parts per se

We can understand only those systems that have a hierarchic structure

Module 03: End of Lecture 04

- Structure of Complex Systems – Examples
  - Personal Computer
  - Plants
  - Education System in India
- Attributes of a Complex System
  - Hierarchic Structure

Module 03: Lecture 05

- Attributes of a Complex System
  - Hierarchic Structure
  - Relative Primitives
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms

Relative Primitives

- **Subjective Choice** – strongly dependent on the experience and expertise of the designer
- What is primitive for one observer may be at a much higher level of abstraction for another.

The choice of what components in a system are primitive is relatively arbitrary and is largely up to the discretion of the observer of the system

Separation of Concerns

- Hierarchic systems are:
  - **decomposable** – *can be divided into identifiable parts*
  - **nearly decomposable** – *the parts are not completely independent*

Intracomponent linkages	**Intercomponent linkages**
• Involves the *internal structure* of the components	• Involves *interaction among components*
• *Stronger*	• *Weaker*
• *High-frequency* dynamics	• *Low frequency* dynamics

Difference between intra- and intercomponent interactions provides a clear Separation of Concerns among the various parts of a system – helps the analysis and design in isolation

Common Patterns

- Complex systems have **Common Patterns**
- Complex systems are composed of only a few different kinds of subsystems in various combinations and arrangements (cells found in both plants and animals etc.)

Common Patterns are a major source of reuse in OOAD.

Examples include Design Patterns, STL in C++, Data Structures in Python etc.

Stable Intermediate Forms

- It is extremely difficult to design a complex system correctly in one go
- Start with a simple system and then refine (*Iterative Refinement*)
- Objects, once considered complex, become the primitive objects on which more complex systems are built
- The system matures from one intermediate form to the next

A complex system that works is invariably found to have evolved from a simple system that worked

A complex system designed from scratch never works and cannot be patched up to make it work

Module Summary

- Analyzed the structure of man-made, natural and social complex systems to understand their generic principles by elucidation
- Summarized the attributes of a complex system:
  - Hierarchic Structure
  - Relative Primitives
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655

Module 04: Object-Oriented Analysis & Design

Handling of Complexity

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

This presentation uses diagrams, examples and selected texts from *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007) with kind permission of the author

Module Objectives

- Understand how to handle complexity

Module Outline

- **Organized Complexity**: Canonical Form
  - Decomposition Hierarchy
  - Abstraction Hierarchy
  - Class and Object Structure
  - Canonical Form
- **Disorganized Complexity**: Limited Human Capacity

Handling Complexity

- **Organized Complexity**: The Canonical Form of a Complex System
- **Disorganized Complexity**: The Limitations of the Human Capacity for Dealing with Complexity

Hierarchies of a Complex System

- Hierarchies are of multiple types
  - Decomposition – **part–of** or **HAS–A**
  - Abstraction – **IS–A**

Decomposition (**HAS–A**) Hierarchy

- Personal Computer is composed of
  - CPU
  - Memory
  - Keyboard
  - HDD
  - ...
- This is **Decomposition Hierarchy**

Abstraction (**IS–A**) Hierarchy

- A CPU can be one of
  - Pentium
  - Pentium + MMX
  - Pentium II
  - Celeron
  - Pentium III
  - ...
- All processors are different in details, but share the same functionality of a CPU. We say:
  - Pentium IS–A CPU
  - Pentium + MMX IS–A Pentium
  - Celeron IS–A Pentium II
  - ...
- This is **Abstraction Hierarchy**

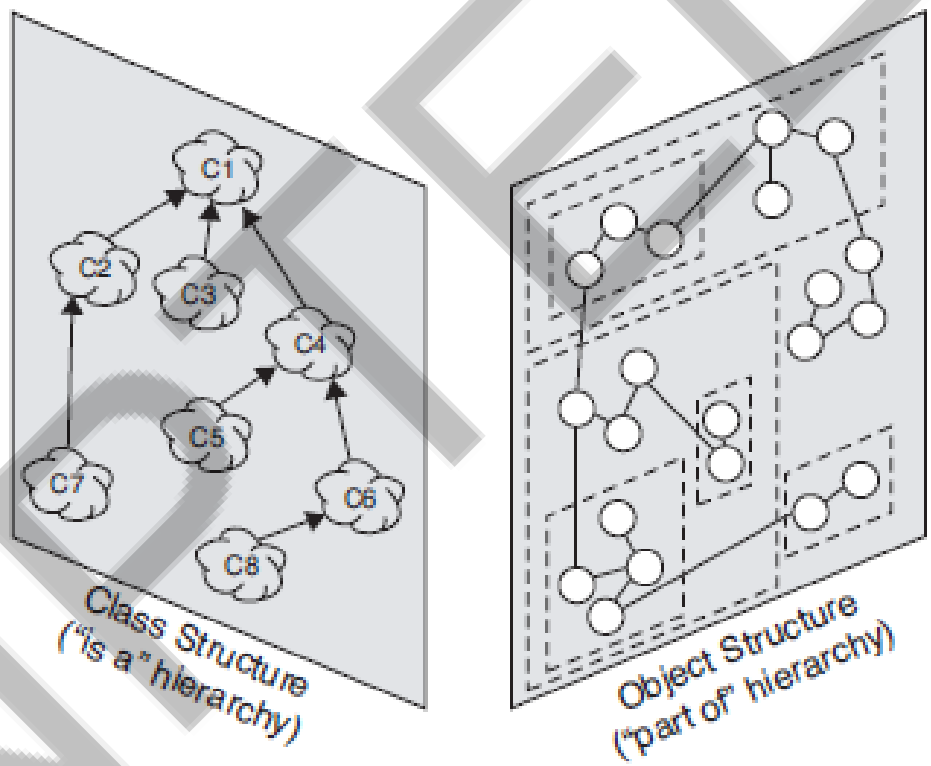
Class and Object Structures

- Hierarchies are referred to as:
  - **Class Structure**: Abstraction (IS–A)
  - **Object Structure** Decomposition (HAS–A)

Note:

- *These are class structure and object structure at a higher level of abstraction*
- *These make up complex systems, for example, a jet engine, an airframe, the various types of seats, an autopilot subsystem, and so forth*
- *These are not classes and objects that we create when coding in software*

Key Hierarchies of Complex Systems



The Key Hierarchies of Complex Systems

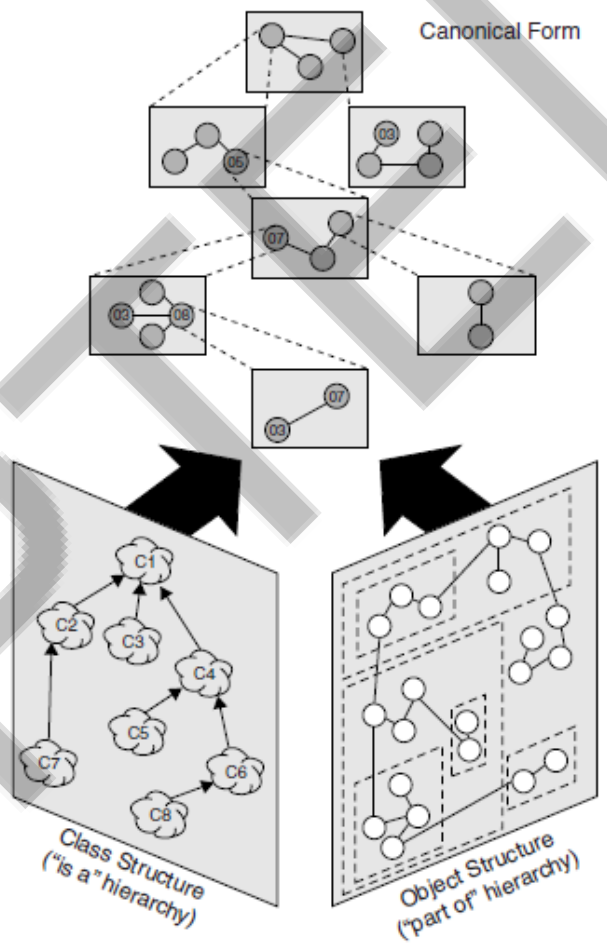
Source: *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007)

The Canonical Form of a Complex System

- System Architecture
  - **Class Structure**: Abstraction (IS–A)
  - **Object Structure** Decomposition (HAS–A)
- Five Attributes of a Complex System
  - Hierarchic Structure
  - Relative Primitives (multiple levels of abstraction)
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms
- This is **Organized Complexity**

Virtually, all complex systems take on the above canonical form

The Canonical Form of a Complex System



The Canonical Form of a Complex System

Source: *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007)

The Limitations of the Human Capacity for Dealing with Complexity

- Maximum number of chunks of information that an individual can simultaneously comprehend is on the order of seven, plus or minus two
- Human channel capacity is related to the capacity of short-term memory
- Processing speed is a limiting factor – it takes the mind about five seconds to accept a new chunk of information
- This leads to **Disorganized Complexity**

Module Summary

- Outlined approaches to handling Complexity
  - **Organized Complexity**: Canonical Form
  - **Disorganized Complexity**: Limited Human Capacity

Fundamental dilemma:

The complexity of the software systems we are asked to develop is increasing, yet there are basic limits on our ability to cope with this complexity

How then do we resolve this predicament?

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655

Module 05: Object-Oriented Analysis & Design

Bringing Order to Chaos

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

This presentation uses diagrams, examples and selected texts from *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007) with kind permission of the author

Recap

- Module 01: Why Software Projects Fail?
- Module 02: Four Elements of Complexity
  - The Complexity of the Problem Domain
  - The Difficulty of Managing the Development Process
  - The Flexibility Possible through Software
  - The Problems of Characterizing the Behavior of Discrete Systems
- Module 03: Structure and Attributes of a Complex System
  - Hierarchic Structure
  - Relative Primitives
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms
- Module 04: Handling of Complexity
  - Decomposition Hierarchy
  - Abstraction Hierarchy
  - Class and Object Structure
  - Organized Complexity: Canonical Form
  - Disorganized Complexity: Limited Human Capacity

Module Objectives

- Build strategies for design of complex systems
- Understand design and model

Module Outline

- Bringing Order to Chaos
  - The Role of Decomposition
  - The Role of Abstraction
  - The Role of Hierarchy
- Designs and Models
  - What is Design?
  - Model Building

Module 05: Lecture 07

- Bringing Order to Chaos
  - The Role of Decomposition

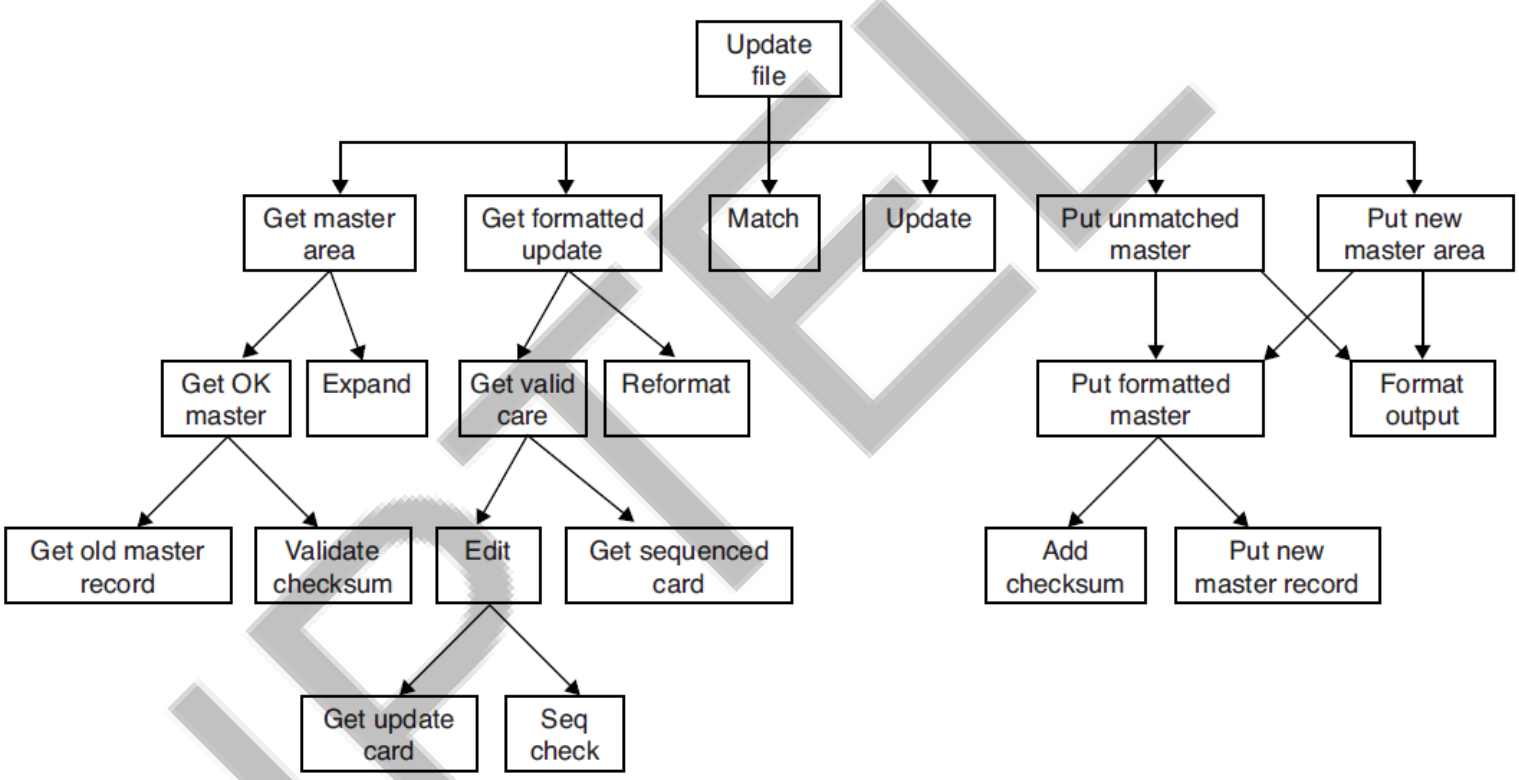
The Role of Decomposition

- Algorithmic Decomposition
- Object-Oriented Decomposition
- Algorithmic versus Object-Oriented Decomposition

Algorithmic Decomposition

- Top-down structured design
- Divide-and-Conquer
- Decompose the problem into key processing steps

Algorithmic Decomposition



Algorithmic Decomposition: Structure Chart

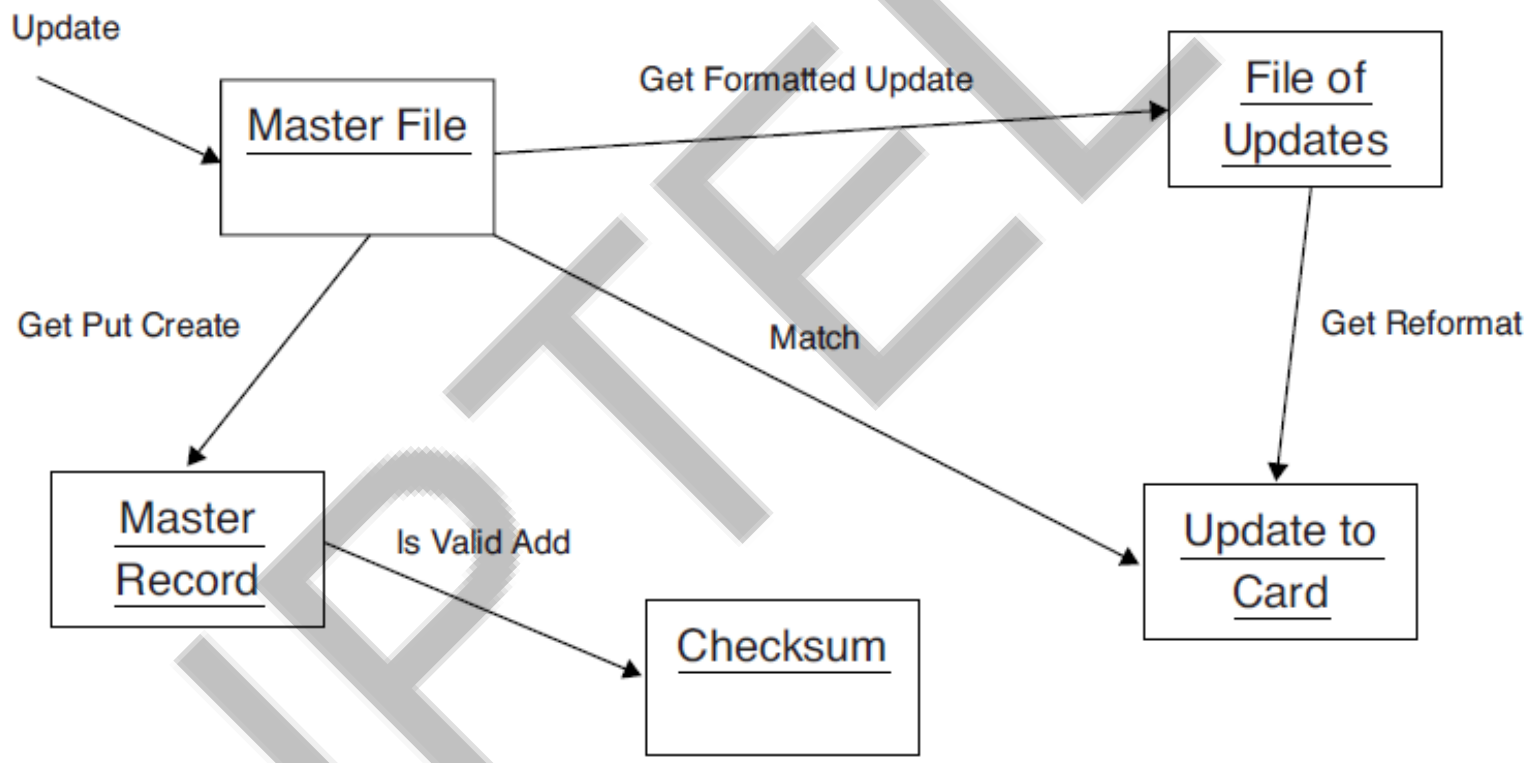
- Design of a program that updates the content of a master file
- Automatically generated from a data flow diagram by an expert system tool
- Decompose the problem into steps like *Get formatted update* & *Add checksum*

Source: *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007)

Object-Oriented Decomposition

- Decompose the system according to the key abstractions in the problem domain
- Objects identified directly from the vocabulary of the problem domain

Object-Oriented Decomposition



Object-Oriented Decomposition

- Decompose according to the key abstractions
- Identify objects like *Master File* and *Checksum* directly from the problem

Source: *Object-Oriented Analysis and Design – With Applications* by Grady Booch et. al. (3rd Ed, 2007)

Object-Oriented Decomposition

- Distinct approach compared to Algorithmic Decomposition
- We view the world as a set of autonomous agents that collaborate to perform some higher-level behavior
- `Get Formatted Update`
  - Removed as an independent algorithm
  - Becomes an operation associated with the object `File of Updates`
  - Calling `Get Formatted Update` on `File of Updates` the operation creates another object, `Update to Card`
- Each object embodies its own unique behavior, and each one models some object in the real world

Object-Oriented Decomposition

- An object is simply a tangible entity that exhibits some well-defined behavior
- Objects do things, and we ask them to perform what they do by sending them messages
- Because our decomposition is based on objects and not algorithms, we call this an **object-oriented decomposition**

Client-Server Computing Model

- Behavior is
  - Services provided by an object
- Services are invoked by
  - Sending Messages
- In Client-Server View
  - Clients request for Services
  - Servers provide Services
  - Contract between client and server ensures correctness

Module 05: End of Lecture 07

- Bringing Order to Chaos
  - The Role of Decomposition

Module 05: Lecture 08

- Bringing Order to Chaos
  - The Role of Decomposition – *continued*
  - The Role of Abstraction
  - The Role of Hierarchy
- Designs and Models
  - What is Design?
  - Model Building

Algorithmic versus Object-Oriented Decomposition: Evolution of Design Methods

- **Top-down structured design**
  - Oldest and most traditional
  - Does not address the issues of data abstraction and information hiding
  - Does not provide an adequate means of dealing with concurrency
  - Does not scale up well for extremely complex systems
  - Is largely inappropriate for use with object-based and object-oriented programming languages.
- **Data-driven design**
  - Mapping system inputs to outputs derives the structure of a software system
  - Has been successfully applied to a number of complex domains (*information management systems*)
  - Less effective for time-critical events
- **Object-oriented design**
  - Models software systems as collections of cooperating objects
  - Treats individual objects as instances of a class within a hierarchy of classes
  - OOAD directly reflects the topology of high-order programming languages (C++ / Java)

Algorithmic versus Object-Oriented Decomposition

Algorithmic	Object-Oriented
• Highlights the ordering of events	• Emphasizes the agents that either (1) Cause action (*Clients*) or (2) Are the subjects on which these operations act (*Servers*)
• Typically larger systems	• Yields smaller systems through the reuse of common mechanisms
• Exposes lower reuse	• Leverages high reuse
• Less resilient and more risky to build complex systems with	• More resilient to change due to underlying stable intermediate forms • Reduces the risk of building complex software systems because they evolve incrementally (iterative refinement)
• Typically flat and unable to reduce complexity	• Directly addresses the inherent complexity of software by using the separation of concerns in a large state space
• Computing Model is typically von Neumann	• Computing Model is Client-Server
• Low concurrency	• Inherently distributed; High concurrency

The Role of Abstraction

- *Disorganized Complexity* results from
  - *Storage (STM) limitations* of human brain – an individual can simultaneously comprehend of the order of seven, plus or minus two chunks of information
  - *Speed limitations* of human brain – it takes the mind about five seconds to accept a new chunk of information
- **Abstraction** provides the major tool to handle Disorganized Complexity by *chunking information*
- Ignore its inessential details, dealing only with the generalized, idealized model of the object

Consider: A binary number 110010101001

Hard to remembers. Right?

Try the octal form: (110)(010)(101)(001) $\Rightarrow$ **6251**

Or the hex form: (1100)(1010)(1001) $\Rightarrow$ **CA9**

The Role of Hierarchy

- Increase the semantic content of individual chunks of information by explicitly recognizing the class and object hierarchies
- *Object structure* $\Rightarrow$ Illustrates how different objects collaborate with one another through patterns of interaction that we call mechanisms
- *Class structure* $\Rightarrow$ Highlights common structure and behavior within a system

What is Design?

Before forging ahead and plunging into Object-Oriented Design Paradigm, let us provide a crisp view of what we consider a *Design*. A Design:

- 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

Model Building

- Physics
  - Time-Distance Equation
- Chemistry
  - Valency-Bond Structures
- Geography
  - Maps
  - Projections
- Electrical Circuits
  - Kirchoff's Loop Equations
  - Time Series Signals and FFT
  - Transistor Models
  - Schematic Diagram
  - Interconnect Routing
- Building & Bridges
  - Drawings – Plan, Elevation, and Side view
  - Finite Element Models

- Models are common in all engineering disciplines
- Model building follows principles of decomposition, abstraction, and hierarchy
- Each model describes a specific aspect of the system
- Build new models upon old proven models

Module Summary

- Decomposition, Abstraction and Hierarchy can help in containing the complexity of a complex software system
- The products of design are models that enable us to reason about our structures, make trade-offs when requirements conflict, and in general, provide a blueprint for implementation
- Models help us immensely to reason with, design, build and debug systems
- We next look into **Object Models**

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655