The University of Iowa

CS:2820 (22C:22)

Object-Oriented Software Development

Spring 2015

Software Complexity

by

Cesare Tinelli

Complexity

- Software systems are complex artifacts
- Failure to master this complexity results in projects that
  - are late
  - go over budget
  - do not meet requirements

Complexity

- The physical world is full of complex systems (both natural and man-made)
- Software's complexity is however fundamentally different:
  - Software is unbound by physical constraints
- *Industrial(-strength)* software exhibits a rich set of behaviors

Industrial Software

- Process Control (oil, gas, water, ...)
- Transportation (air traffic control, ...)
- Health Care (patient monitoring, device control, ...)
- Finance (automatic trading, bank security, ...)
- Defense (intelligence, weapons control, ...)
- Manufacturing (precision milling, assembly, ...)
- ...

Why Software Is Inherently Complex

1. Complexity of problem domain
2. Difficulty of managing development process
3. Flexibility afforded by software
4. Difficulty of characterizing discrete system behavior

Complexity of Problem Domain

- Many, often contradictory, requirements
  - functional (what must be done)
  - non-functional (usability, cost, performance, consumption,...)
- Communication gap between customers and developers
- Evolving requirements

Difficulty of Managing Development Process

- Fundamental task of software development:

  engineering the illusion of simplicity

- However, ...

Difficulty of Managing Development Process

- Modern systems are huge (10^6 LOC, 10^2 modules)
- Development team is necessary
- More developers =>
  - more complex communication
  - more difficult coordination
  - harder to maintain design unity/integrity

Flexibility Afforded by Software

- Software is the ultimate flexible product
- It is technically possible for any developer to create anything with it
- This is both a blessing and a curse
- Other industries have specialization, codes and quality standards
- Software development remains a mostly artisanal labor-intensive business

Difficulty of Characterizing Discrete Systems Behavior

- Physical (analog) systems exhibit continuous behavior
  - Small external perturbations produce small changes in behavior
- Software (digital) systems exhibit discrete behavior
  - Small changes in input can produce large changes in output

Difficulty of Characterizing Discrete Systems Behavior

- Discrete systems have a combinatorial state explosion
- Describing their behavior precisely and formally is very challenging in general
- Most software professionals are poorly trained for that
- Testing for flaws is intrinsically insufficient

Common Features of *Good* Complex Systems

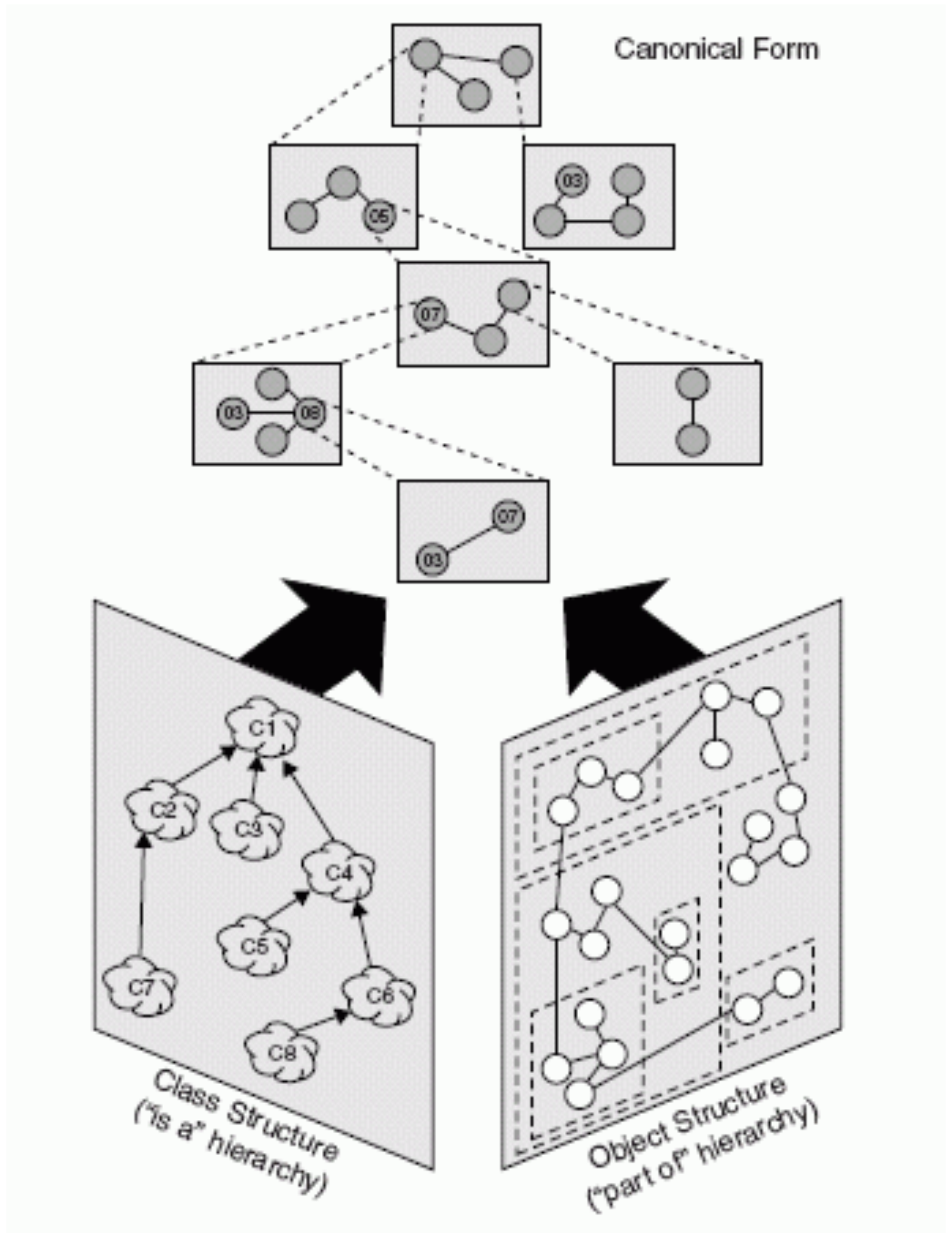
1. Nearly decomposable, hierarchic structure
2. Primitive components
3. Separation of concerns
4. Combination of common patterns
5. Stable intermediate forms

Organized Complexity

- Many hierarchies can be found in a complex system
- Most important for us:
  - object structure ("part of" relation)
  - class structure ("is a" relation)
- We refer to them together as the system's architecture

Canonical Form of a Complex System

- Classes capture common features of a set of objects
- Each object is an instance of a class
- Objects are composed of and interact with other objects



Successful Complex Software Systems

- Exhibit the 5 attributes characterizing good complex systems
- Have well designed and built (i) class and (ii) object structures
  - (i) captures common features and behavior within a system
  - (ii) illustrates how different objects collaborate with one another

The Software Development Predicament

- The complexity of software systems is ever increasing
- The human ability to cope with complexity is fundamentally limited
- Time-honored technique to master complexity: *divide et impera*
- Decomposition and abstraction are key

Two Alternative Decomposition Approaches

1. Algorithmic Decomposition

   Each component denotes a major step in the system's overall process

2. Object-Oriented Decomposition

   Each component denotes a major agent in the system's overall process

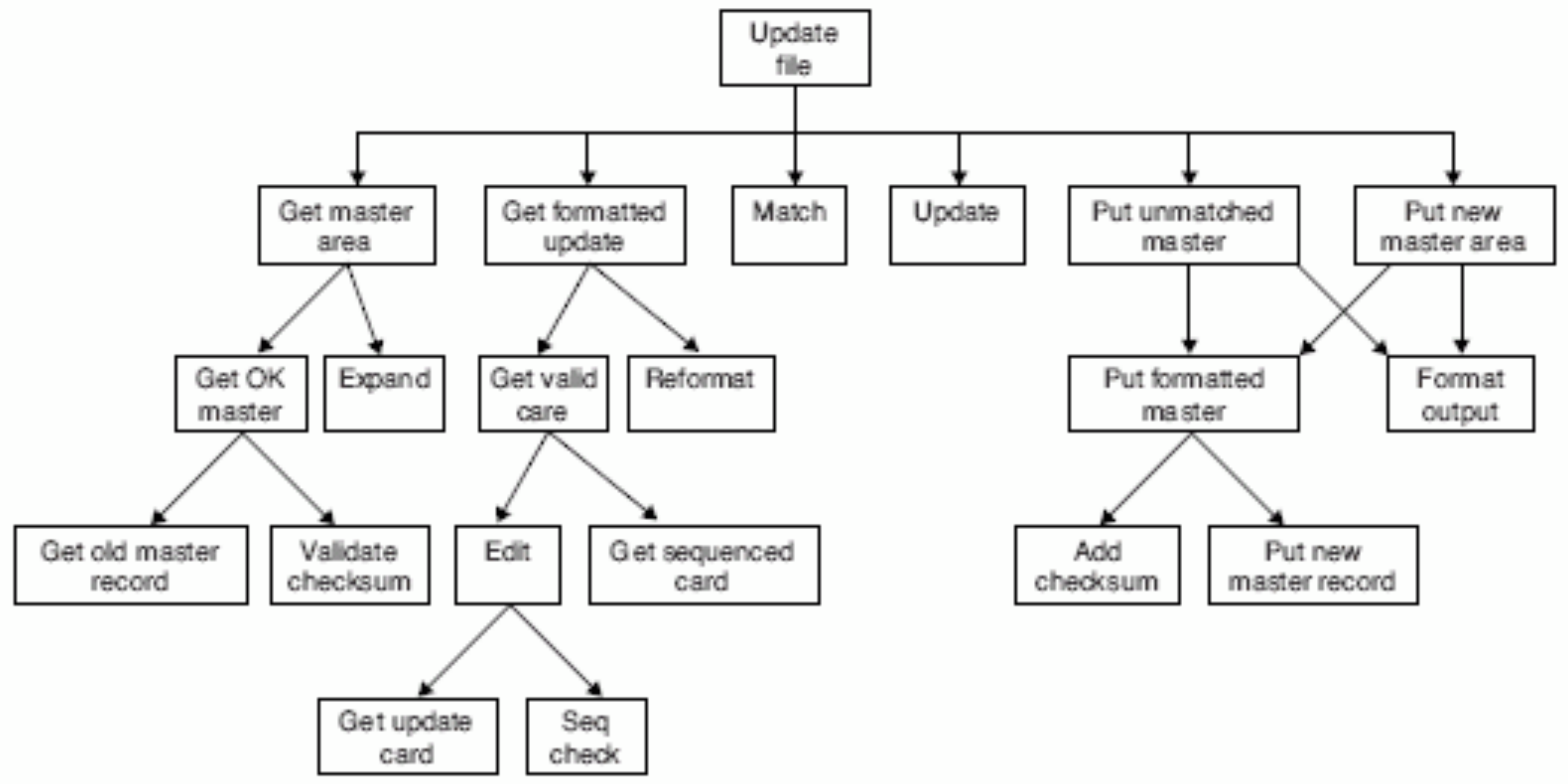


Figure 1–3 Algorithmic Decomposition

Design of a program that updates the content of a master file

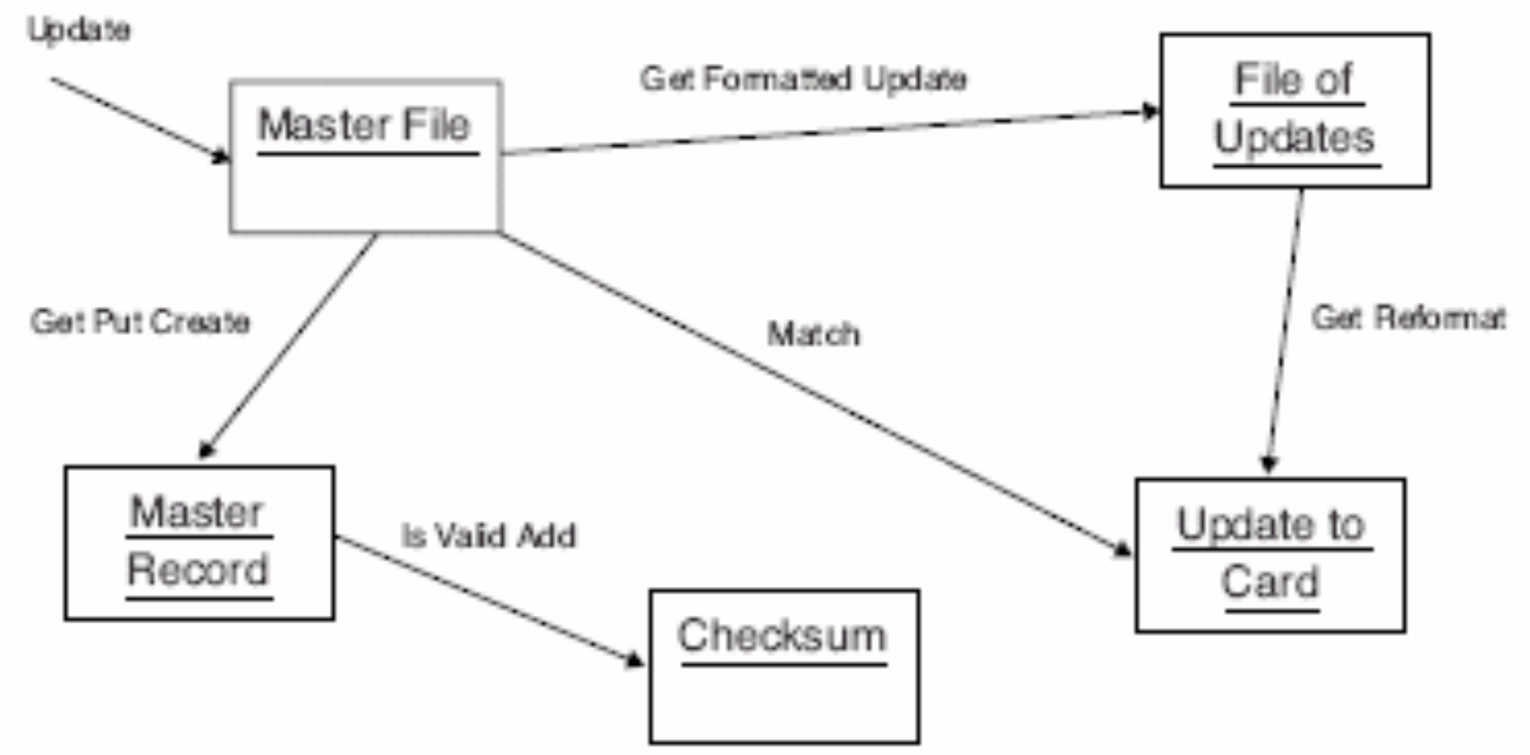


Figure 1–4 Object-Oriented Decomposition

Design of a program that updates the content of a master file

Main Advantages of OO Decomposition

It facilitates

- reuse of components and mechanisms
- system evolution over time
- separation of concerns

The Role of Design in Software Development

Construct a system that [Mostow]:

- 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

The Importance of Model Building in Design

- Widespread in all engineering disciplines
- Appeals to the principles of abstraction, decomposition, and hierarchy
- Models
  - can be evaluated and modified before the actual system is built
  - allow us to focus on important aspects by abstracting away irrelevant details

Basic Elements of Software Design Methodologies

Notation The language for expressing models

Process The activities leading to the orderly construction of a system's model

Tools The artifacts that facilitate the creation and validation of models

Effective OO Design and Development

Requires mastery of these underlying principles:

- abstraction
- encapsulation
- modularity
- hierarchy
- typing
- concurrency
- persistence

References

1. G. Booch *et al*. Object-Oriented Analysis and Design with Applications, 3rd Edition. Addison-Wesley, 2007.