Organized and Disorganized Complexity

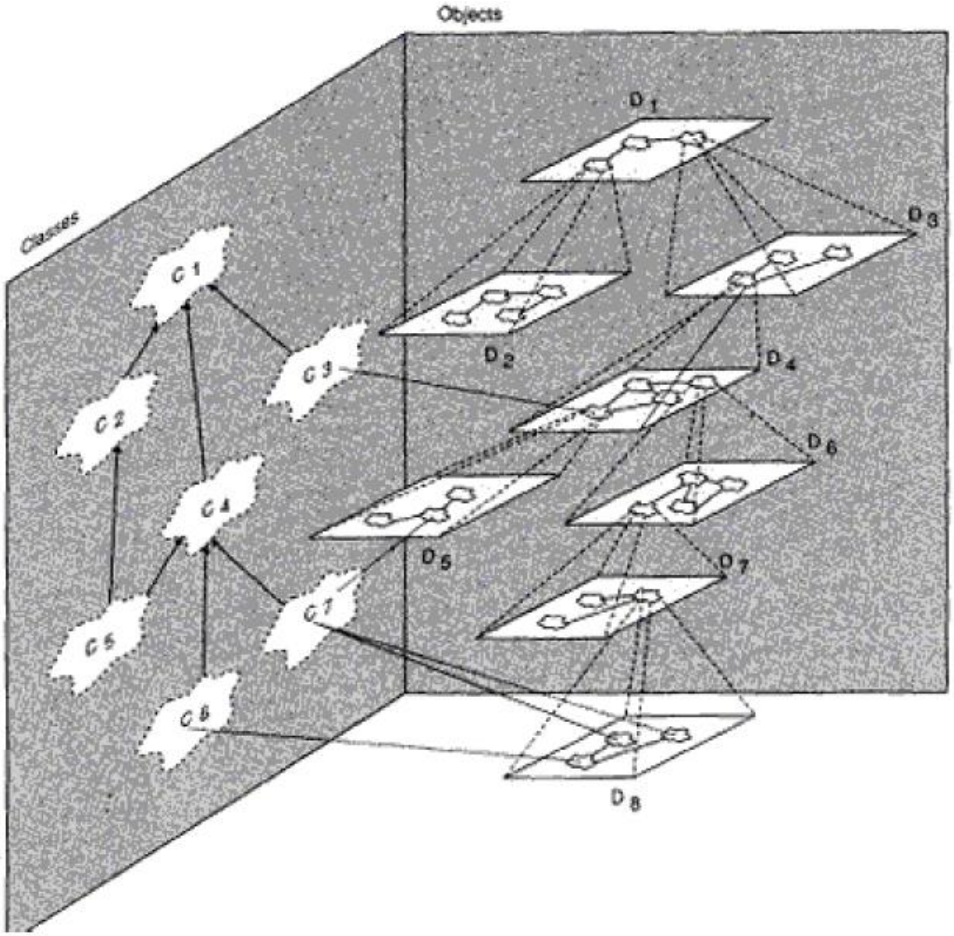
The Canonical Form of a Complex System

- The discovery of common abstractions and mechanisms greatly facilitates our understanding of complex systems.

- EX: with just a few minutes of orientation, an experienced pilot can step into a multiengine jet aircraft he or she has never flown before and safely fly the vehicle.

  Having recognized the properties common to all such aircraft, such as the functioning of the rudder, ailerons, and throttle, the pilot primarily needs to learn what properties are unique to that particular aircraft. If the pilot already knows how to fly a given aircraft, it is far easier to know how to fly a similar one.

The Canonical Form of a Complex System

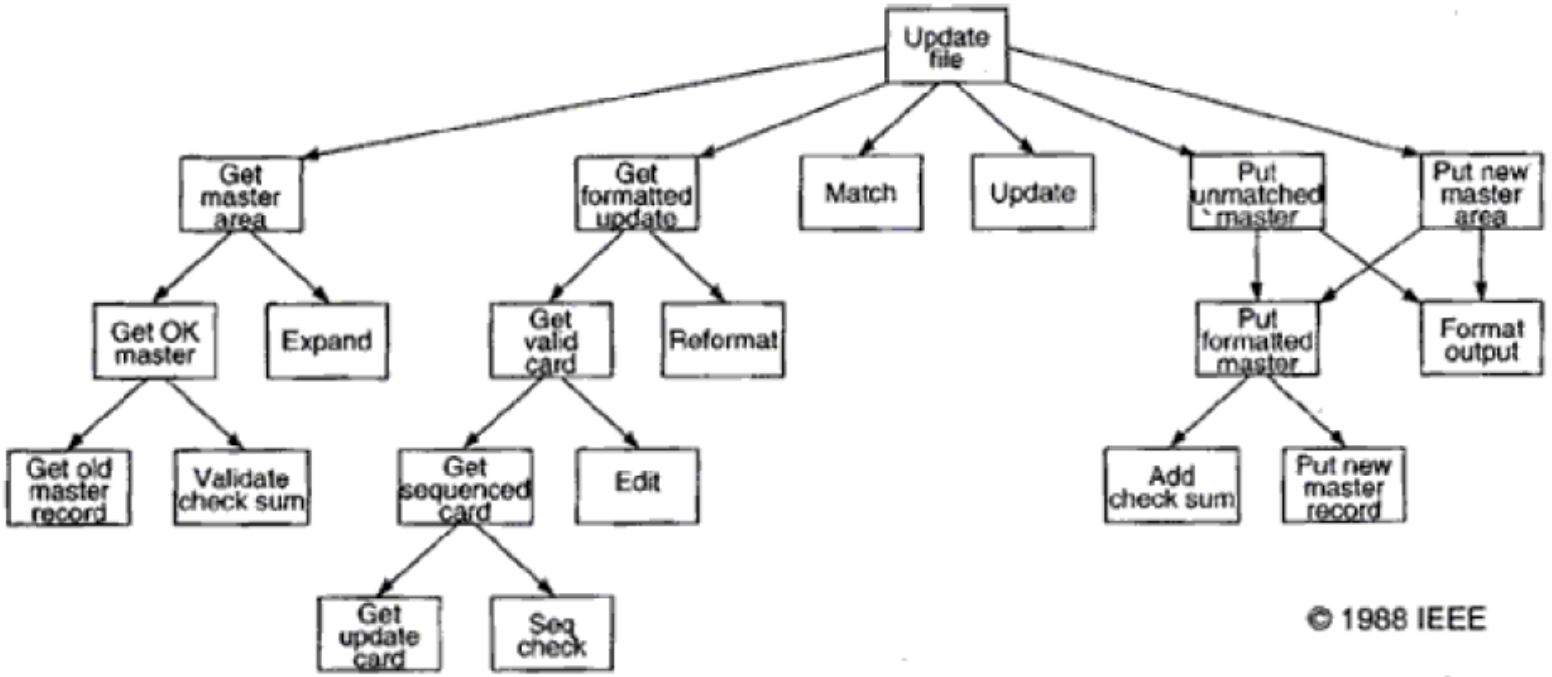


The Role of Decomposition

- As Dijkstra suggests, “The technique of mastering complexity has been known since ancient times: *divide et impera (divide and rule)*"
- When designing a complex software system, it is essential to decompose it into smaller and smaller parts, each of which we may then refine independently.

1. **Algorithmic Decomposition**
2. **Object-Oriented Decomposition**
3. **Algorithmic versus Object-Oriented Decomposition**

Algorithmic Decomposition



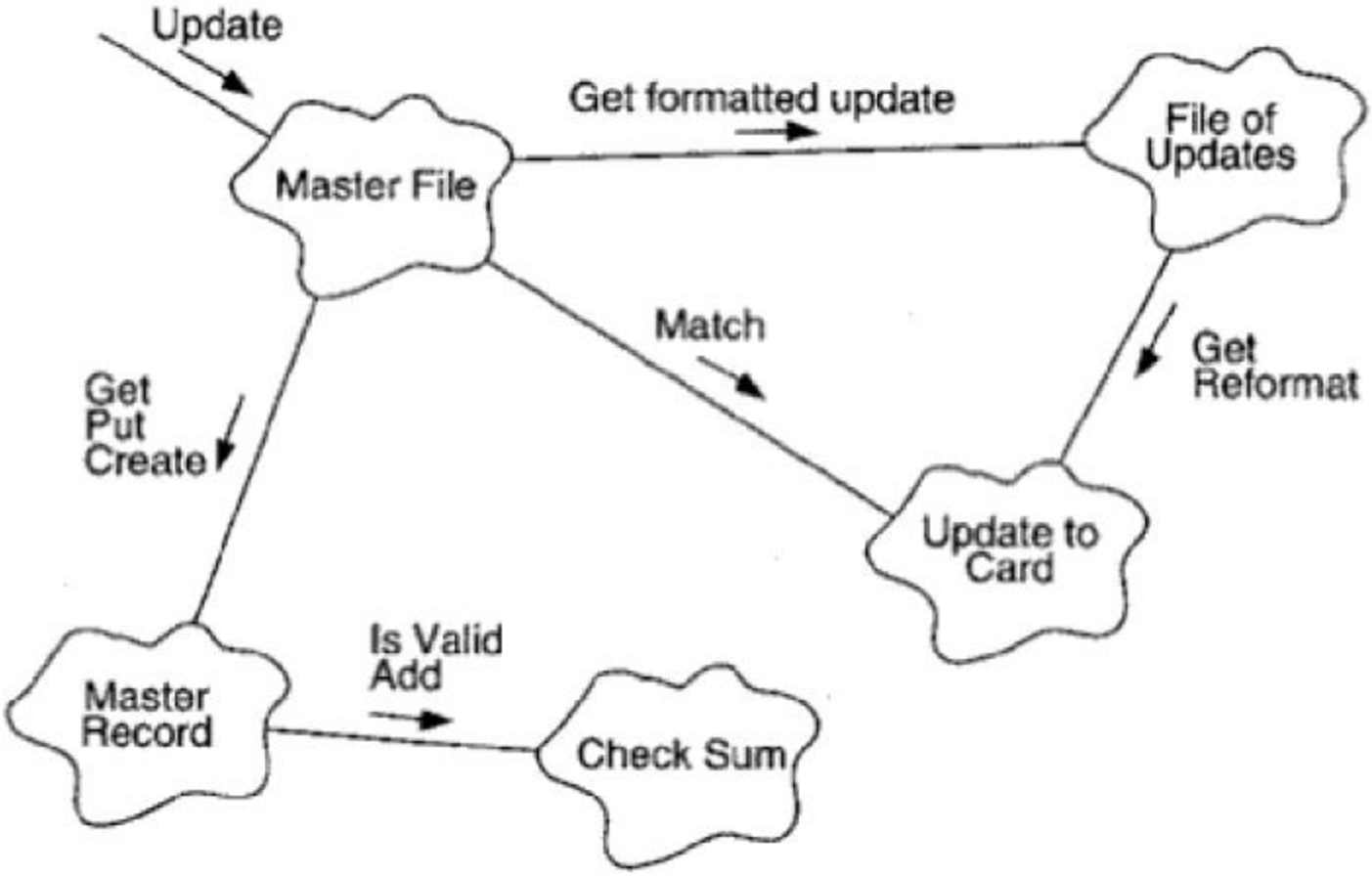
Algorithmic Decomposition

- Most of us have been formally trained in the dogma of top down structured design, and so we approach decomposition as a simple matter of algorithmic decomposition

Object-Oriented Decomposition

- we have decomposed the system according to the key abstractions in the problem domain.

Object-Oriented Decomposition



Algorithmic versus Object-Oriented Decomposition

- Which is the right way to decompose a complex system - by algorithms or by objects?

  Both views are important.
- The algorithmic view highlights the ordering of events, and the object-oriented view emphasizes the agents that either cause action or are the

  subjects upon which these operations act.
- The object-oriented view first because this approach is better at helping us organize the inherent complexity of software systems, just as it helped us to describe the organized complexity of complex systems.
- Object oriented decomposition has a number of highly significant advantages over algorithmic decomposition.
- Object-oriented decomposition greatly reduces the risk of building complex software systems