Object-oriented programming

Lecture №13

OOP conception, Complexity

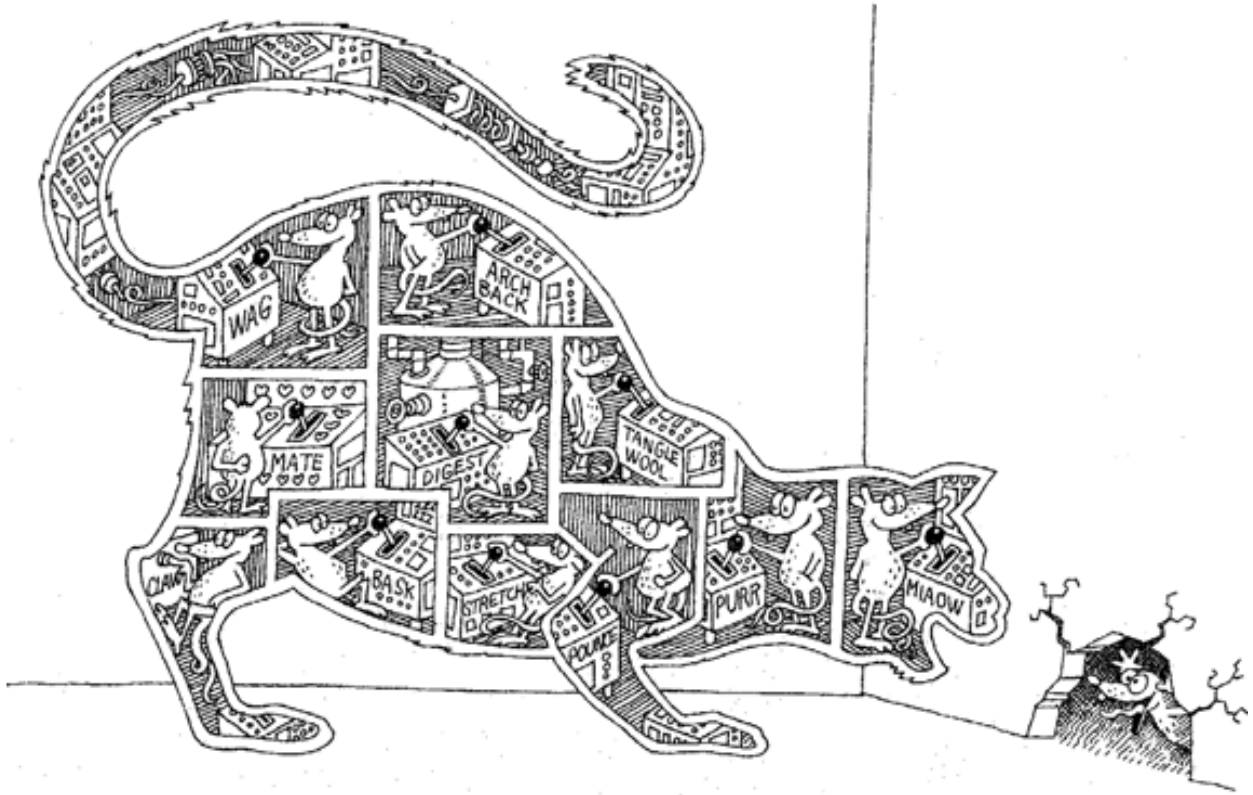
PhD, Alexander Vlasov

Question?

- Thread
- Amdahl’s Law
- Thread States
- Interrupt/Termination
- Method join(), sleep()
- Thread Scheduling and Priority
- Thread Pools
- Monitor. Key word «synchronized». Methods: wait(), notify(), and notifyAll(). Semaphores
- Deadlock

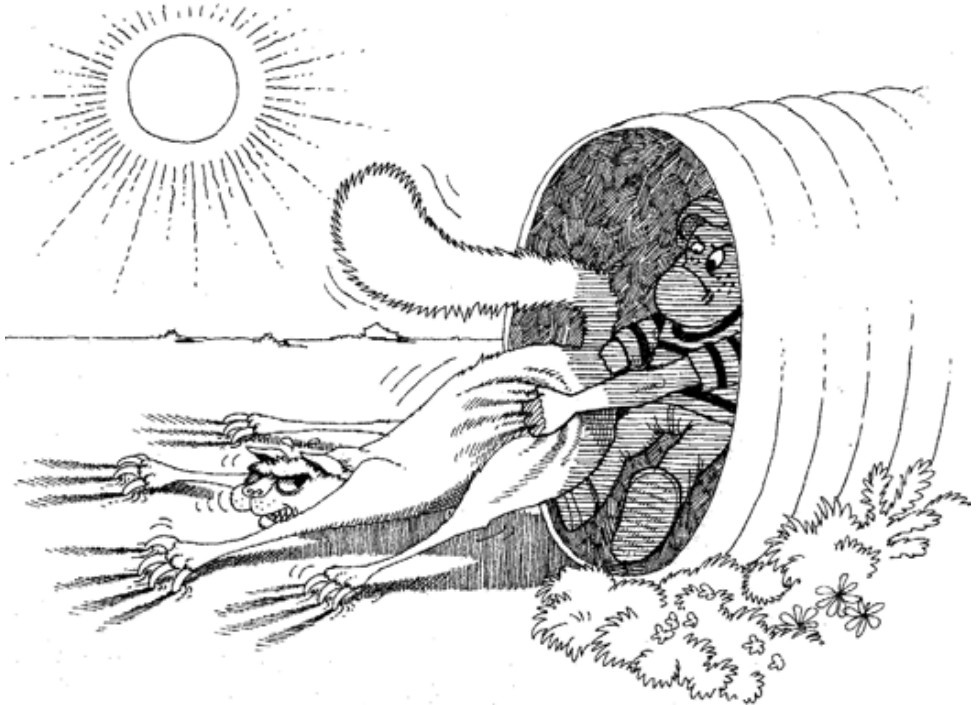
Concurrency

- Concurrency is the property that distinguishes an active object from one that is not active.



Persistence

- Persistence is the property of an object through which its existence transcends time (i.e., the object continues to exist after its creator ceases to exist) and/or space (i.e., the object's location moves from the address space in which it was created).



OOP conception. Summary.

- An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.
- Encapsulation is the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.
- Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.
- Hierarchy is a ranking or ordering of abstractions.
- Typing is the enforcement of the class of an object, such that objects of different types may not be interchanged or, at the most, may be interchanged only in very restricted ways.
- Concurrency is the property that distinguishes an active object from one that is not active.
- Persistence is the property of an object through which its existence transcends time and/or space.

Why Software Is Inherently Complex

- 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

Complex system properties

- Hierarchic Structure
  - complexity takes the form of a hierarchy, where by 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.
- Relative Primitives
  - 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
  - Intracomponent linkages are generally stronger than intercomponent linkages. This fact has the effect of separating the high-frequency dynamics of the components—involving the internal structure of the components—from the lowfrequency dynamics—involving interaction among components.
- Common Patterns
  - Hierarchic systems are usually composed of only a few different kinds of subsystems in various combinations and arrangements.
- Stable Intermediate Forms
  - 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. You have to start over, beginning with a working simple system.

Complexity

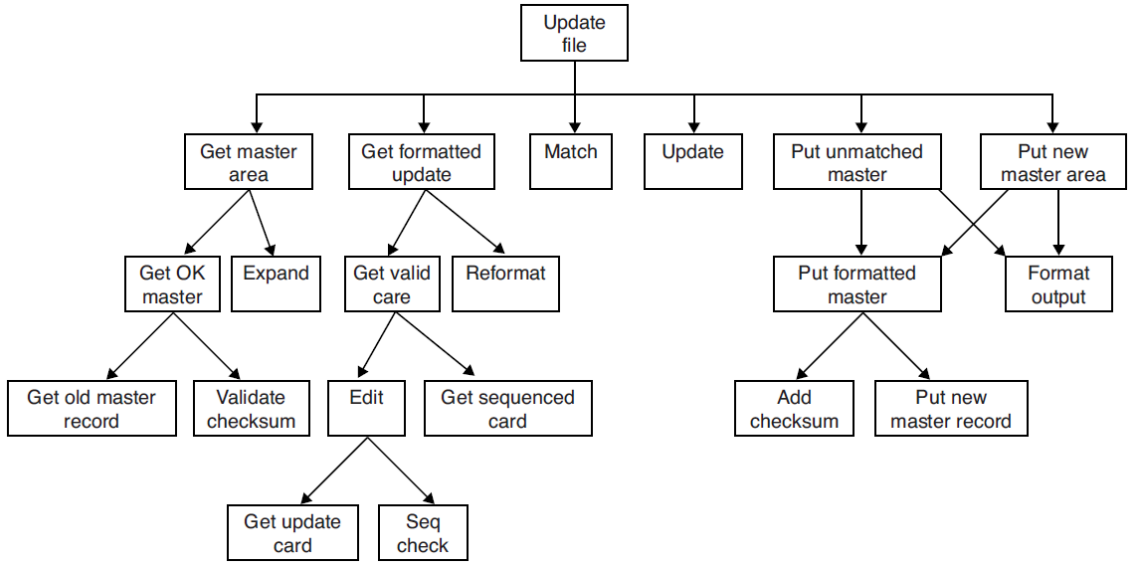
We are thus faced with a fundamental dilemma. The complexity of the software systems we are asked to develop is increasing, yet there are basic limits on our ability to cope with this complexity. How then do we resolve this predicament?

Solution

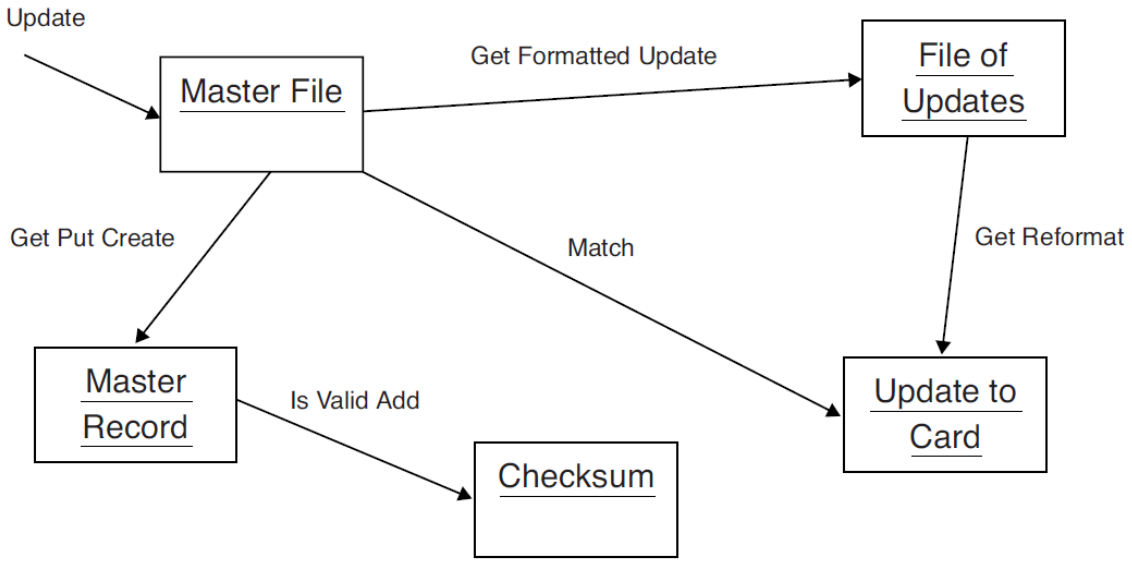
- The technique of mastering complexity has been known since ancient times: divide et impera (divide and rule)

Algorithmic Decomposition

- Algorithmic Decomposition



- Object-Oriented Decomposition



Object-oriented program

- 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.

Step by step

- the type of system complexity
- primitives subsystem
- modules and relationships
- Processes, objects, classes