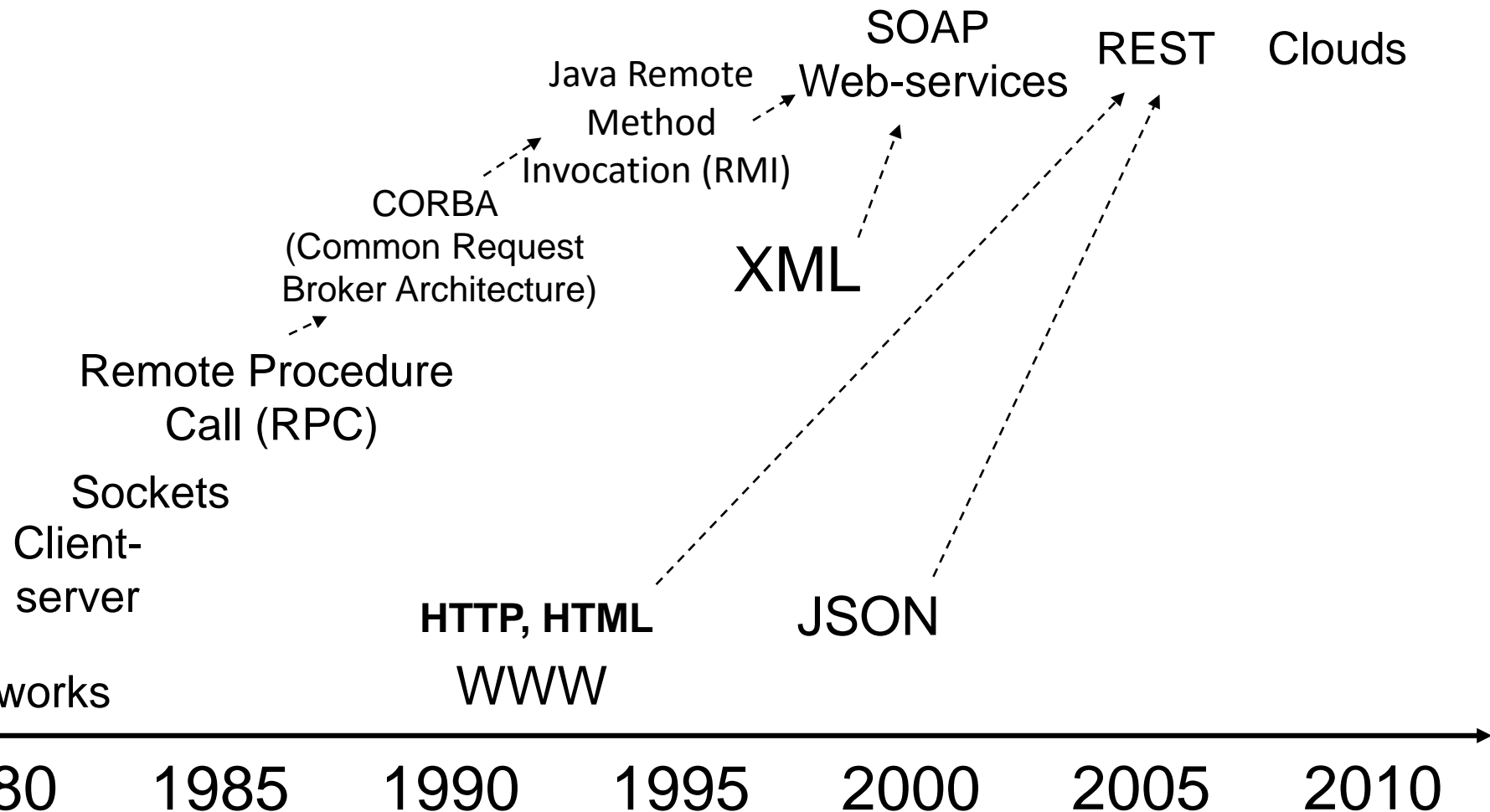


# DISTRIBUTED COMPUTING SYSTEMS

Service-oriented architecture (SOA)

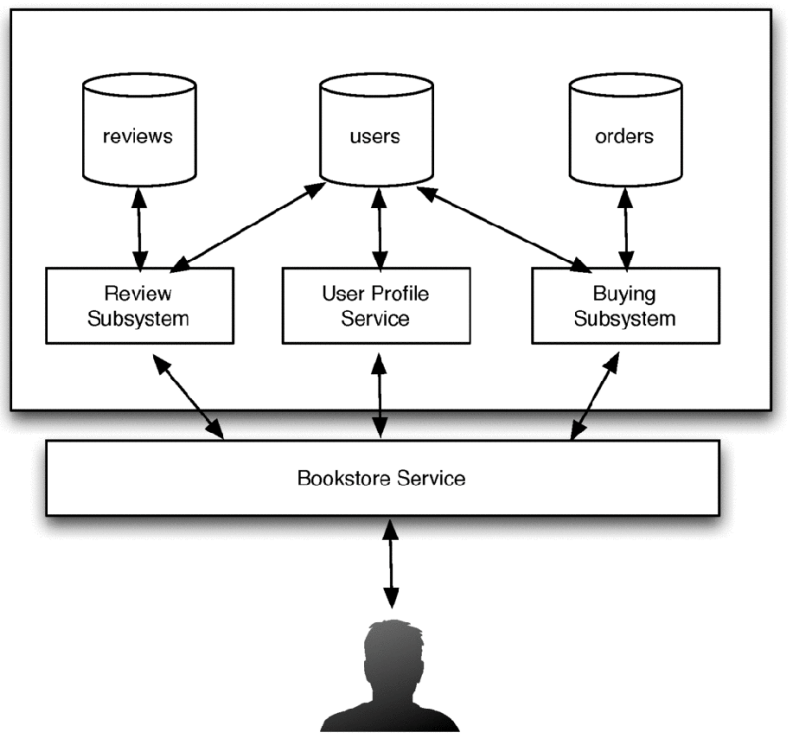
# STANDARDS, PROTOCOLS OF DCS



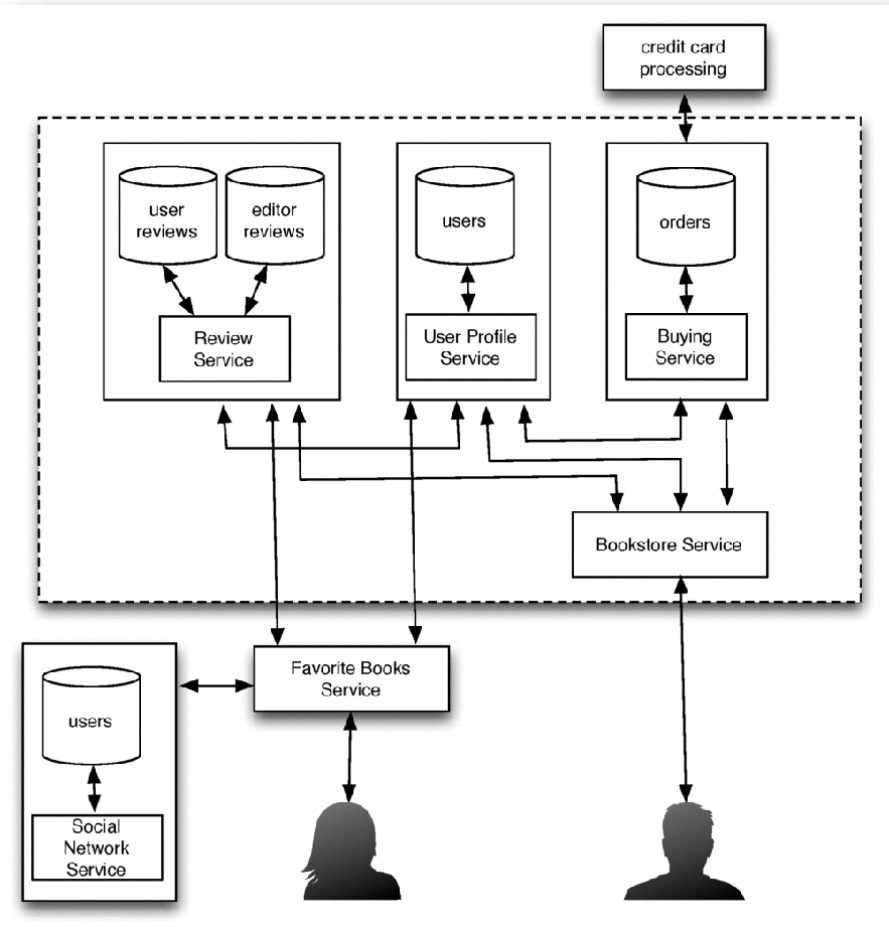
# INTRODUCTION: JEF BEZOS LETTER (2002)

- ⊙ All teams will henceforth expose their data and functionality through service interfaces.
- ⊙ Teams must communicate with each other through these interfaces.
- ⊙ There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model etc... The only communication allowed is via service interface calls over the network.
- ⊙ It doesn't matter what technology they use (HTTP, Corba, etc.)
- ⊙ All service interfaces must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- ⊙ Anyone who doesn't do this will be fired.
- ⊙ Thank you; have a nice day!

# SILO VS SOA APPROACH



«Silo» Approach



SOA Approach

# SERVICE-ORIENTED ARCHITECTURE OF DISTRIBUTED SYSTEMS

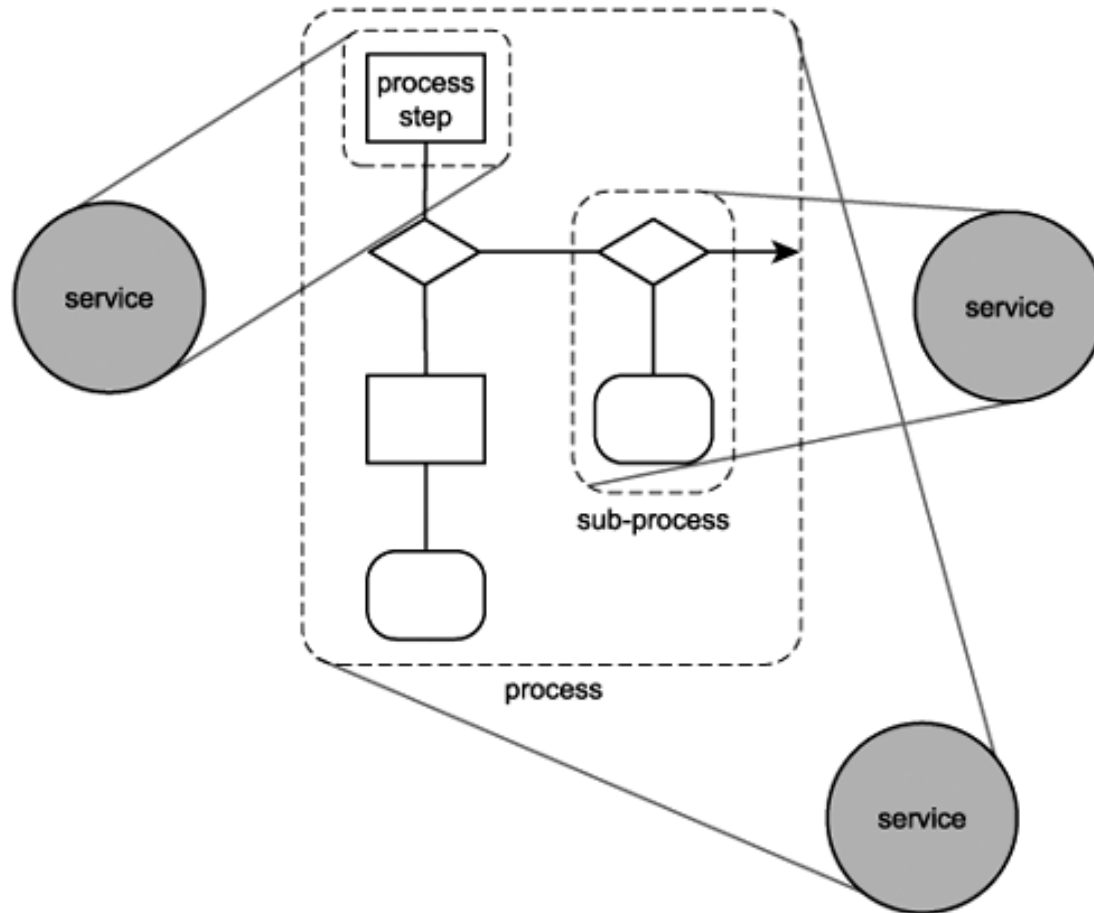
# SERVICE-ORIENTED ARCHITECTURE (SOA)

- ◎ **Service oriented architecture** is a paradigm for organizing and utilizing distributed capabilities that may be provided by various owners
  - ◎ Definition of OASIS (Organization for the Advancement of Structured Information Standards)
- ◎ SOA actually means that components of an application act as interoperable services, and can be used independently and recombined in other applications.

# SOA VS OOP

- ◎ Object-oriented architecture is based on the essence of the “object”
- ◎ At the heart of SOA lies an “action”

# SERVICES ENCAPSULATE ACTIONS





# COMPONENTS OF A SERVICE-ORIENTED ARCHITECTURE

1. Service components (services)
2. Service contracts (interfaces)
3. Service Connectors (transport)
4. Service discovery mechanisms (registries)

# SERVICE COMPONENTS

- ③ **Software components** that provide transparent network addressing.
- ③ **Services** are open, the self-defined components that support rapid construction of distributed applications.
- ③ Services can be *fine-grained* or *coarse-grained*.

# FINE-GRAINED SERVICE

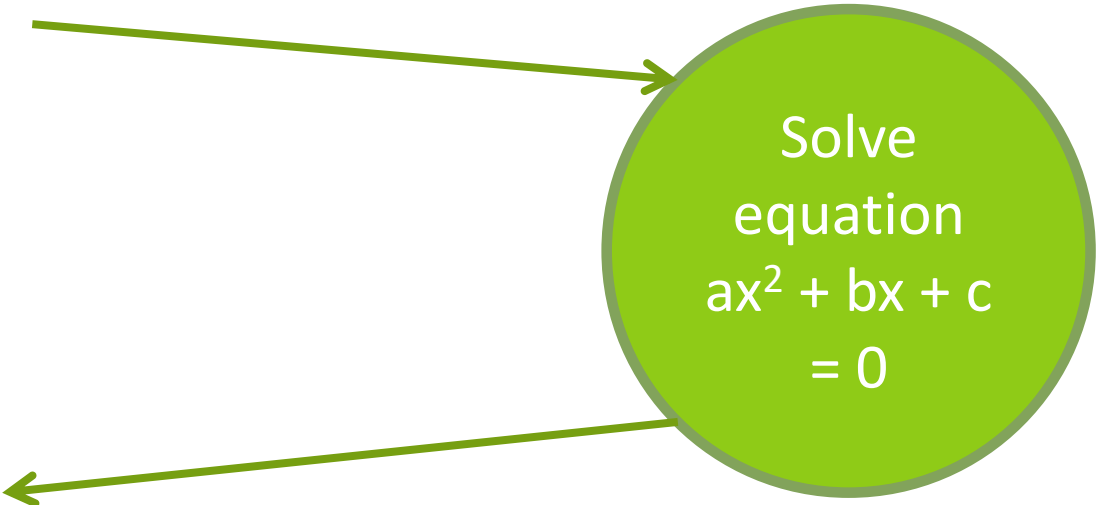
- ③ **Fine-grained** service provides an elementary volume of functional load and ensures a high degree of reuse.
- ③ You must coordinate the work of several services to get the result

# COARSE-GRAINED SERVICE

- ③ **Coarse-grained** service provides a high degree of encapsulation of functionality.
- ③ But makes it difficult to reuse, due to the narrow specialization

# COARSE-GRAINED SERVICE FOR SOLVING QUADRATIC EQUATIONS

$$a = 5; b = 10; c = 3$$

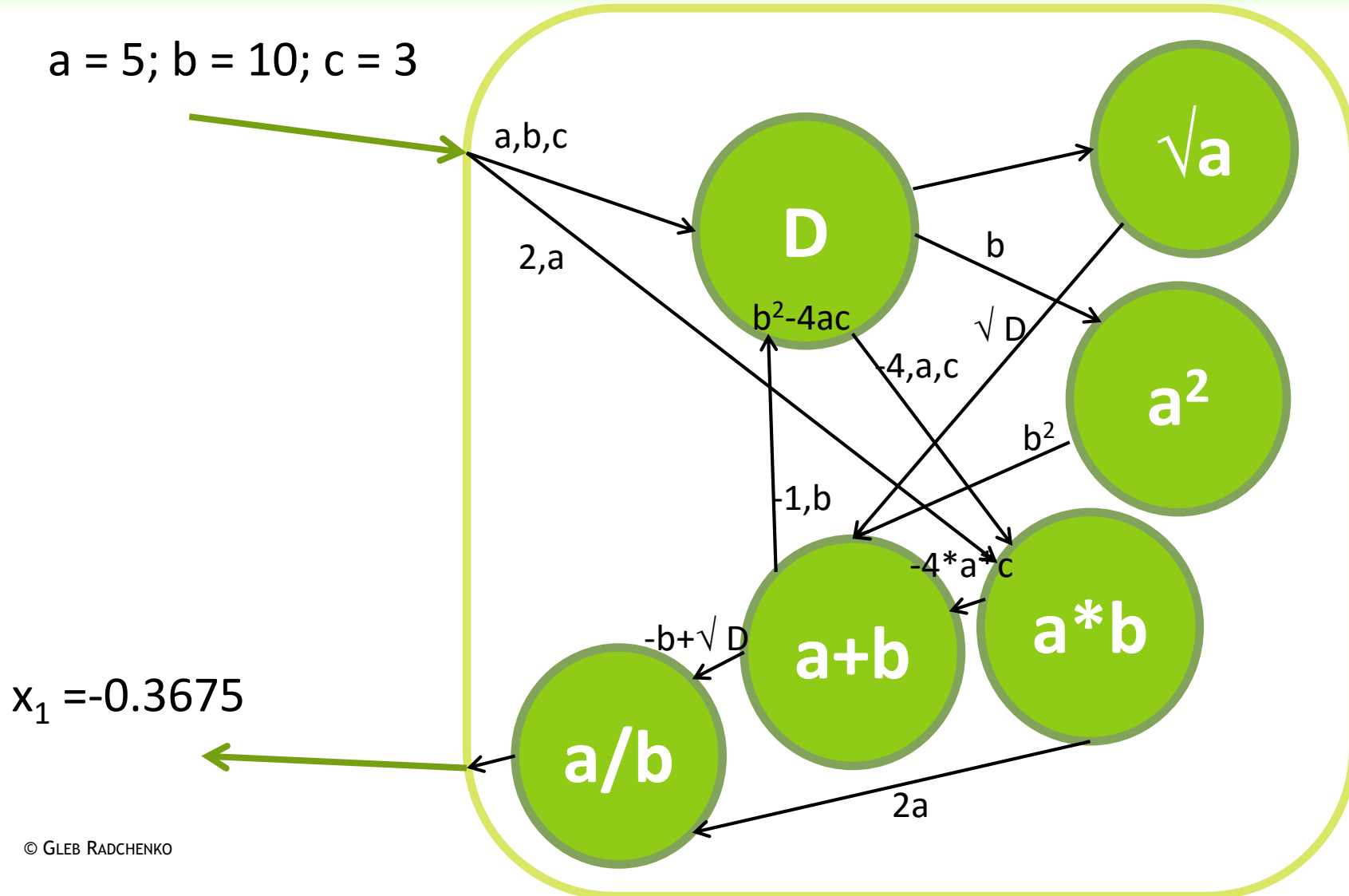


Solve  
equation  
 $ax^2 + bx + c$   
 $= 0$

$$x_1 = -0.3675$$

$$x_2 = -1.6325$$

# ORCHESTRATION OF FINE-GRAINED SERVICES FOR SOLVING QUADRATIC EQUATIONS



# SERVICE CONTRACTS

- ◎ **The interface** is a description of possibilities offered by a particular service.
- ◎ The interface is defined by:
  - ◎ The format of the messages
  - ◎ Input and output parameters

# SERVICE CONNECTORS

- ③ **Transport** provides exchange of information between the components.
- ③ If you use flexible transport protocols for the exchange of information between service components, it improves the software compatibility of service oriented system



# SERVICE DISCOVERY MECHANISMS (REGISTRIES)

- ◎ They are used to locate services with the required functionality.
- ◎ *Static discovery*: focus on storing information about rarely changing systems
  - ◎ telephone exchange station, UDDI
- ◎ *Dynamic discovery*: a system in which there is frequent appearance and disappearance of the service components:
  - ◎ P2P, mobile agents

# WEAKLY AND TIGHTLY COUPLED SOFTWARE SYSTEMS

# COHESION OF SOFTWARE SYSTEMS

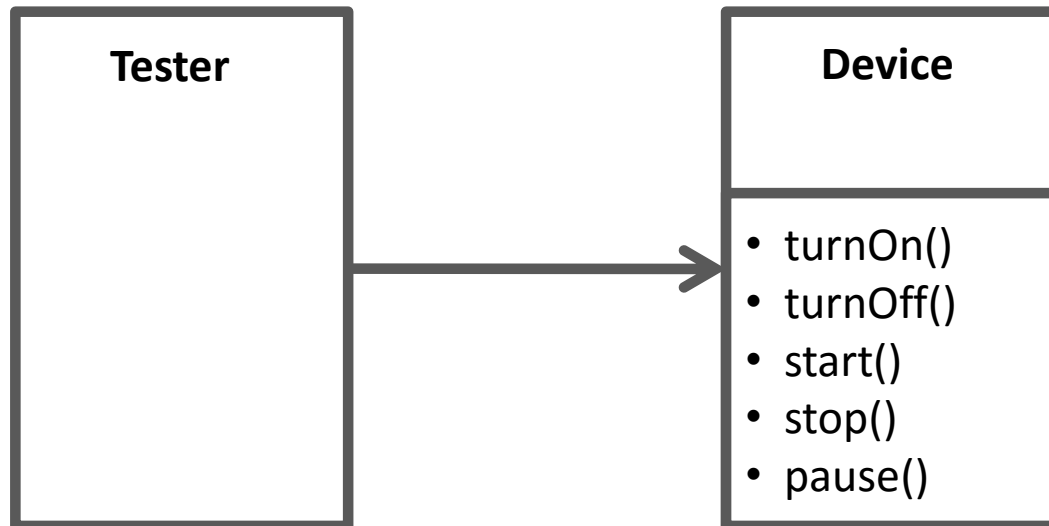
*Cohesion* is a degree of knowledge and dependence of one object from the internal contents of the other.

Software systems can be divided into 2 types:

- 1. Tightly (or strong) coupled systems**
- 2. Loosely coupled systems (SOA)**

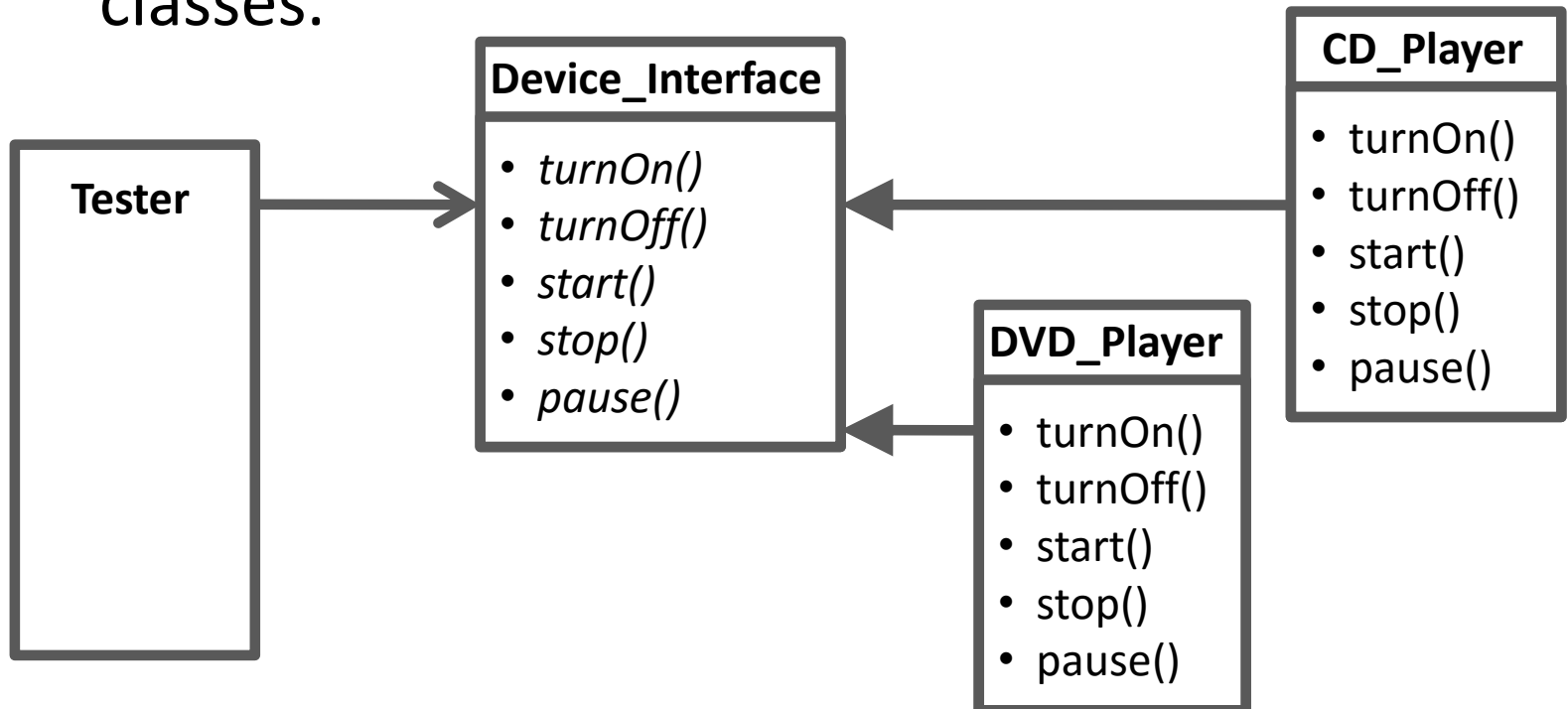
# TIGHTLY COUPLED SYSTEMS

- ⊙ Tight coupling occurs when the dependent class contains a link directly **to a specific class** that provides some possibilities.



# LOOSELY COUPLED SYSTEMS

- ◎ *Loose coupling* occurs when the dependent class contains a **reference to an interface** that can be implemented by one or more specific classes.



# THE BENEFITS OF LOOSELY COUPLED SYSTEMS

- ③ Decreases the number of relationships between the components of the system, reducing the amount of the possible consequences in connection with failures
- ③ It is possible to increase the working system, by creating new classes with single interface