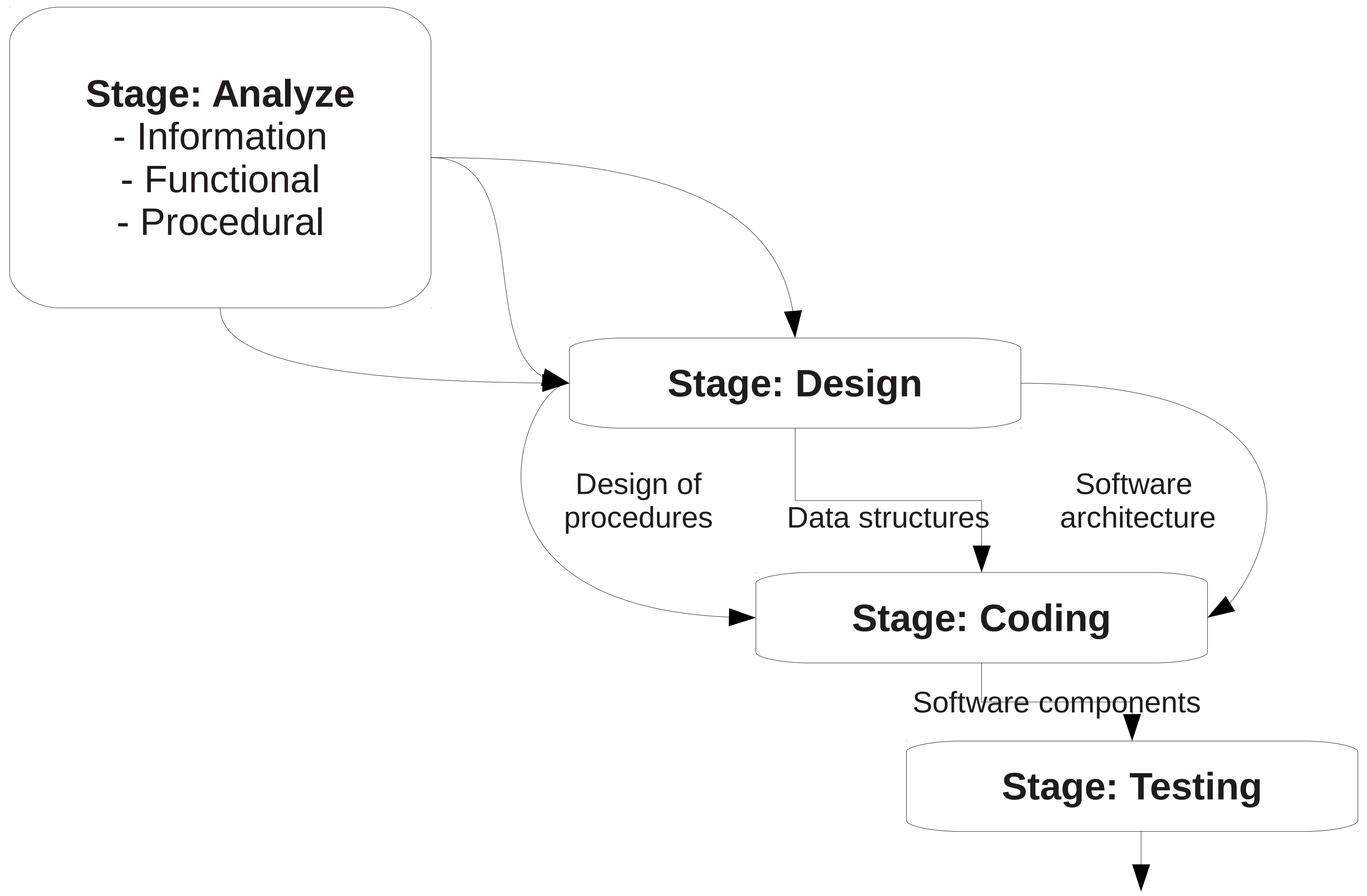
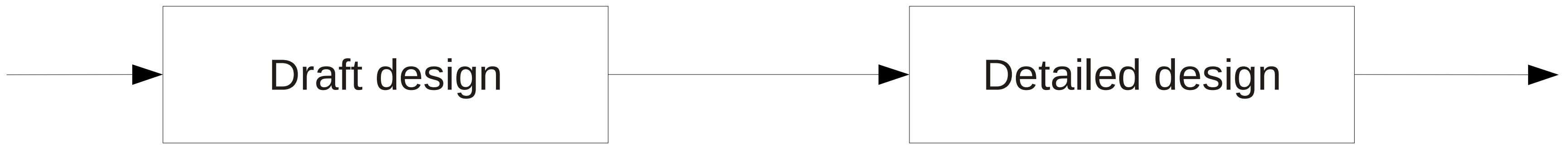
Lection №6

Software architecture

Information flow



Information links



In the world

In usual software architecture are described in “views”. IEEE 1471-2000 «Recommended Practice for Architecture Description of Software-Intensive Systems»
Possible views:

* Functional/logic view
* Code/module view
* Development/structural view
* Concurrency/process/thread view
* Physical/deployment view
* User action/feedback view
* Data view

Definitions

architecture: The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.. [IEEE 1471]

system: A collection of components organized to accomplish a specific function or set of functions. The term system encompasses individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest. [IEEE 1471]

A system’s **environment**, or context', can influence that system. The environment can include other systems that interact with the system of interest, either directly via interfaces or indirectly in other ways. The environment determines the boundaries that define the scope of the system of interest relative to other systems. [IEEE 1471]

A **mission** is a use or operation for which a system is intended by one or more stakeholders to meet some set of objectives. [IEEE 1471]

system stakeholder: An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system. [IEEE 1471]

Definition

Архитектура - это набор значимых решений по поводу организации системы программного обеспечения, набор структурных элементов и их интерфейсов, при помощи которых компонуется система, вместе с их поведением, определяемым во взаимодействии между этими элементами, компоновка элементов в постепенно укрупняющиеся подсистемы , а также стиль архитектуры который направляет эту организацию -- элементы и их интерфейсы, взаимодействия и компоновку. [Крачтен (Kruchten)]

Архитектура программы или компьютерной системы - это структура или структуры системы, которые включают элементы программы, видимые извне свойства этих элементов и связи между ними. [Басс (Bass) и др.]

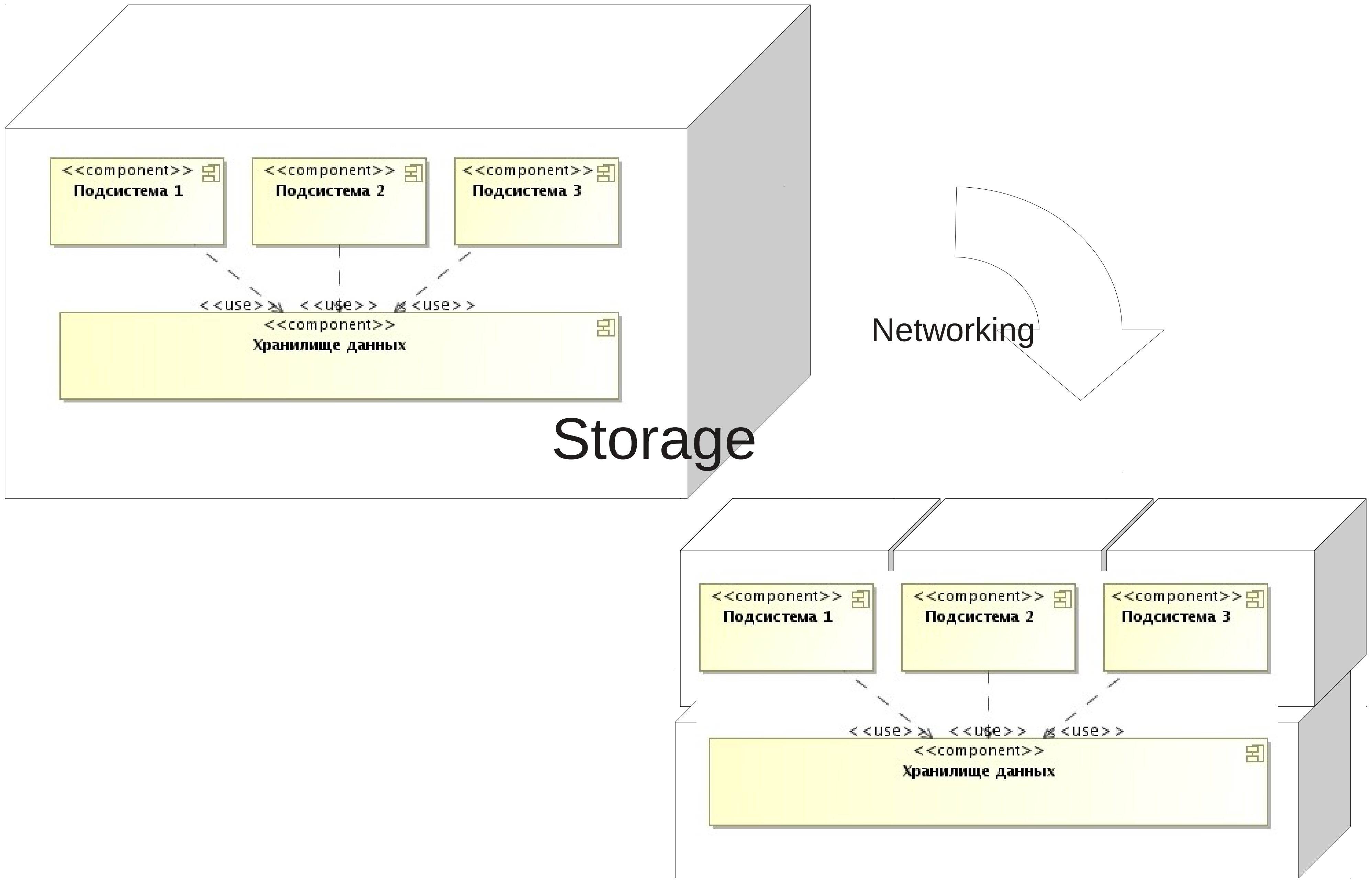
[Архитектура - это] структура организации и связанное с ней поведение системы. Архитектуру можно рекурсивно разобрать на части, взаимодействующие посредством интерфейсов, связи, которые соединяют части, и условия сборки частей. Части, которые взаимодействуют через интерфейсы, включают классы, компоненты и подсистемы. [UML 1.5]

Архитектура программного обеспечения системы или набора систем состоит из всех важных проектных решений по поводу структур программы и взаимодействий между этими структурами, которые составляют системы. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система, чтобы быть успешной. Проектные решения предоставляют концептуальную основу для разработки системы, ее поддержки и обслуживания. [Мак-Говерн (McGovern)]

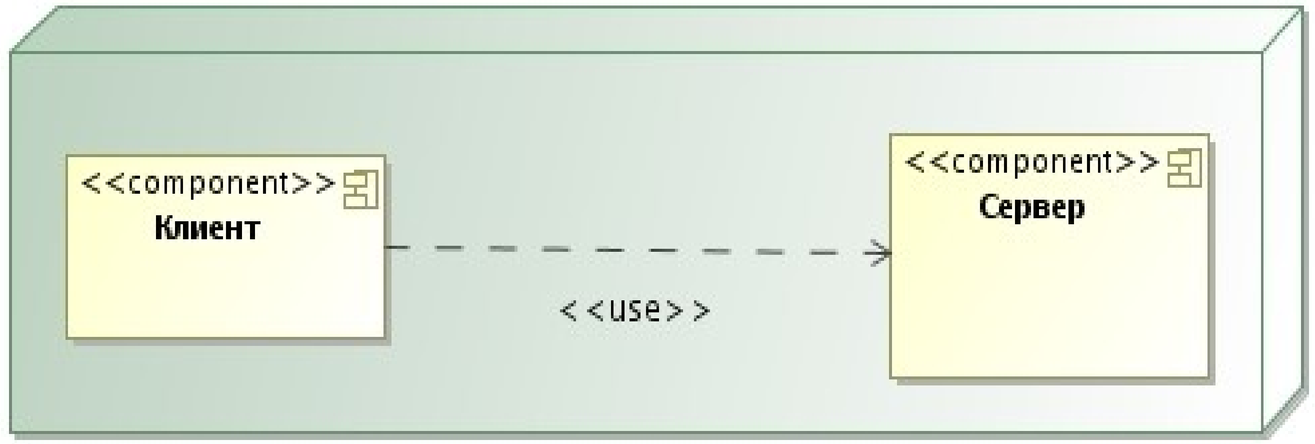
Architecture ...

- Defines structure
- Defines behavior
- Concentrates on main elements
- Balances users' needs
- Based on logical decisions
- Influenced by environment
- Affects on users' communications
- Has special area(s) of use

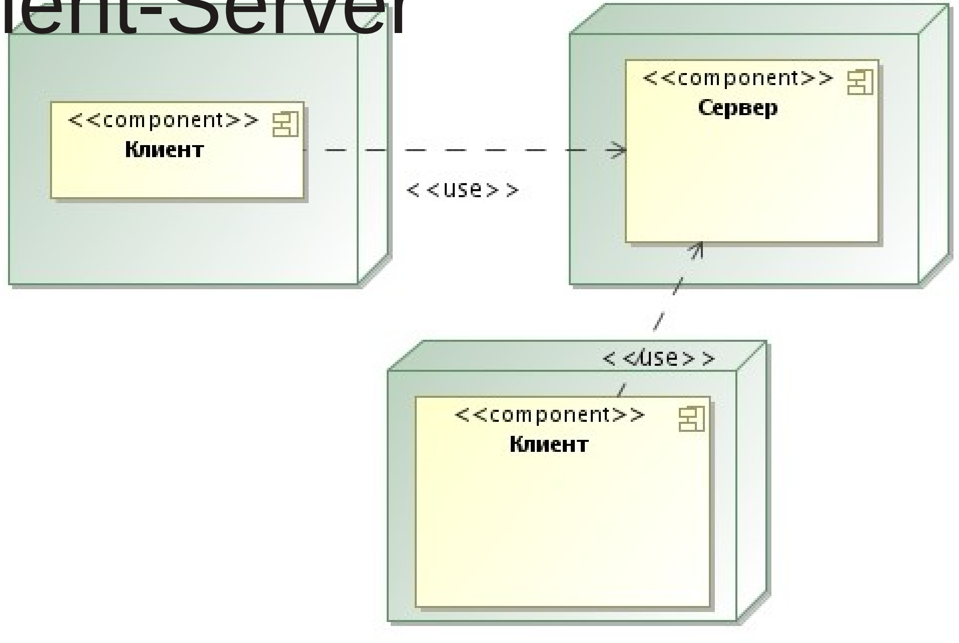
Architectures



Architectures



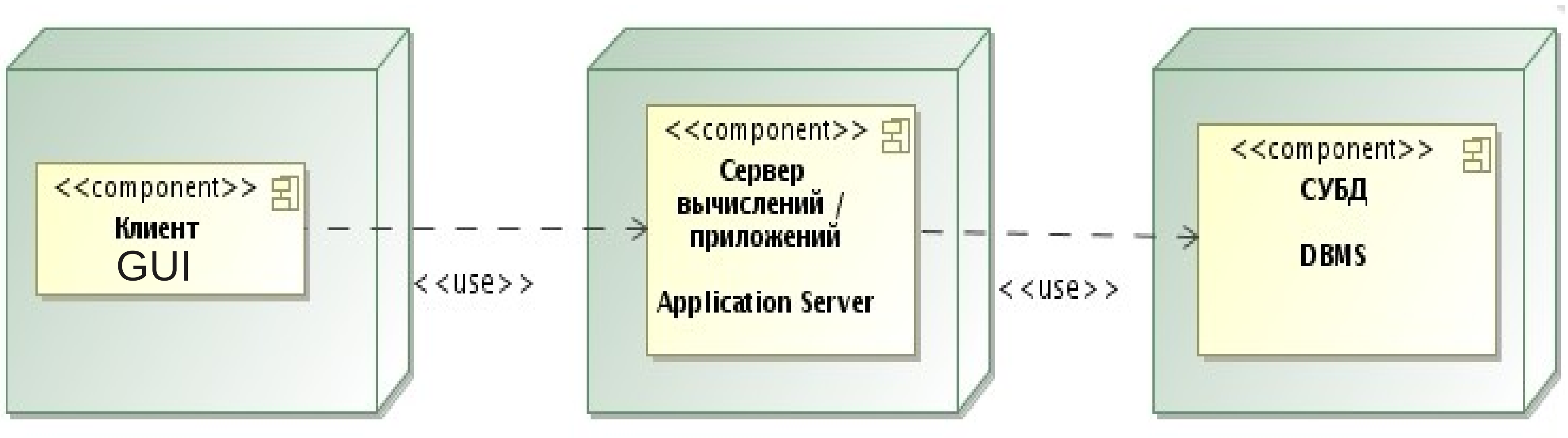
2-nd tier: Client-Server



Тонкий клиент — Thin client

Толстый клиент — Thick client

Architectures



3-d tier architecture

Browser

Internet

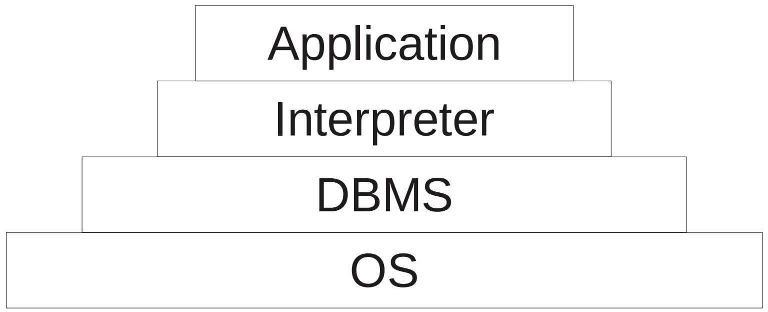
Web-server +logic

DBMS client

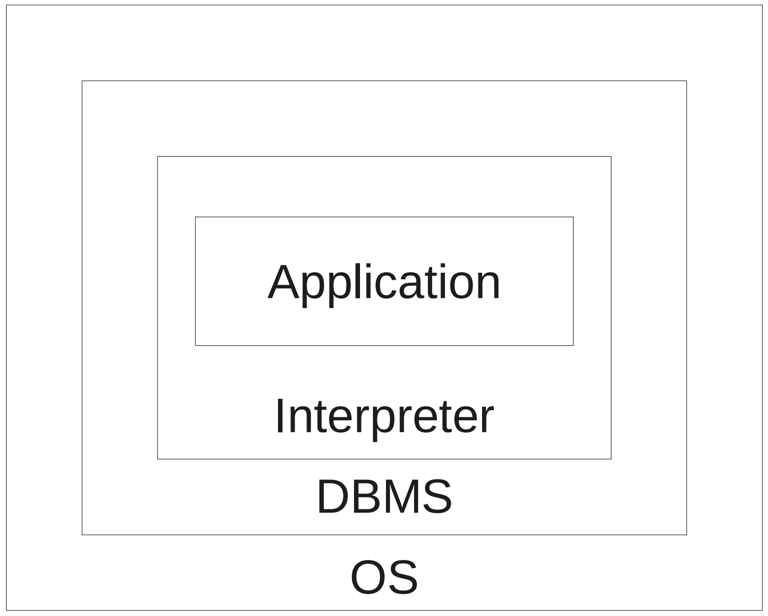
LAN

DBMS

Architectures



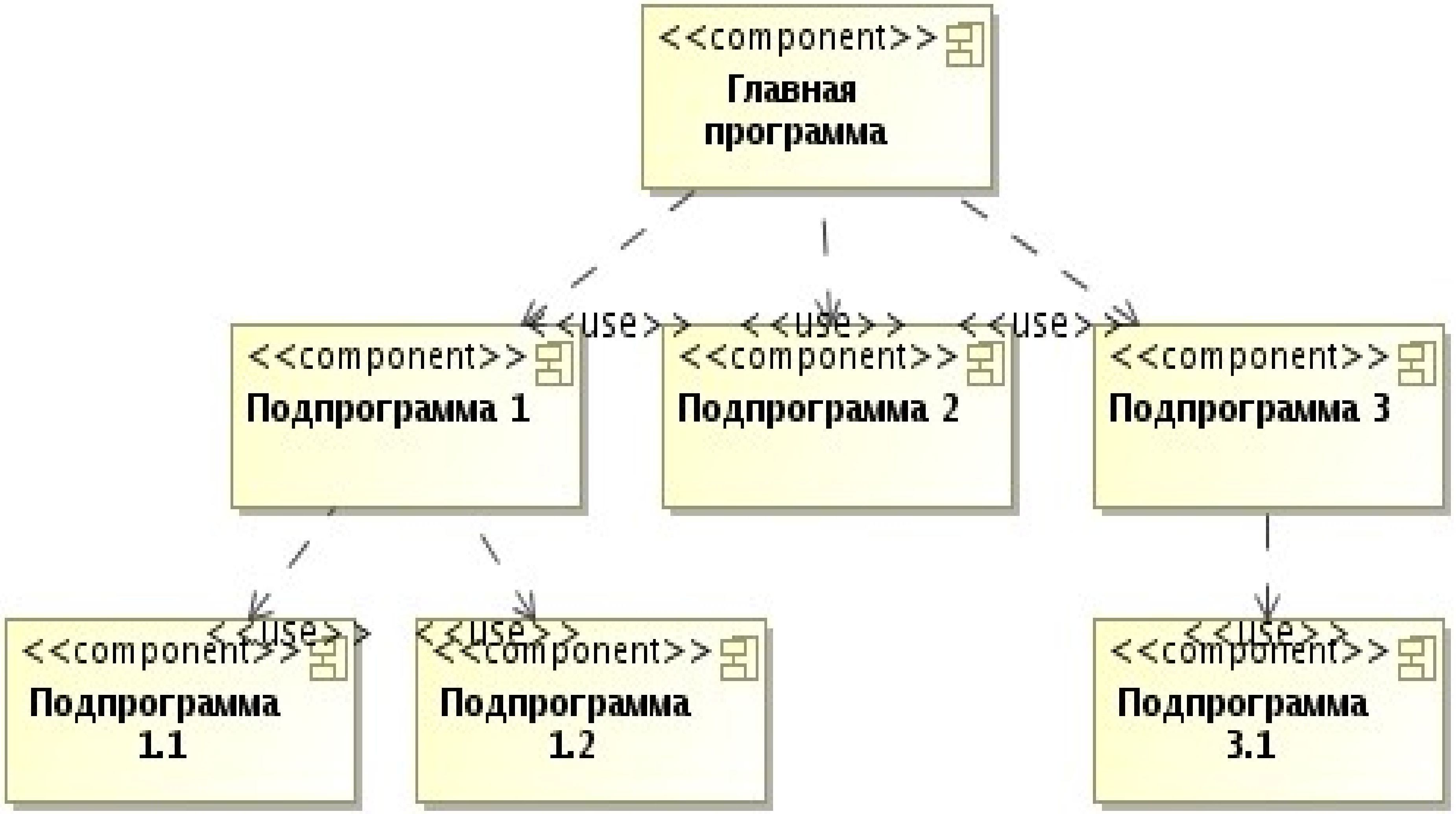
Abstract machine



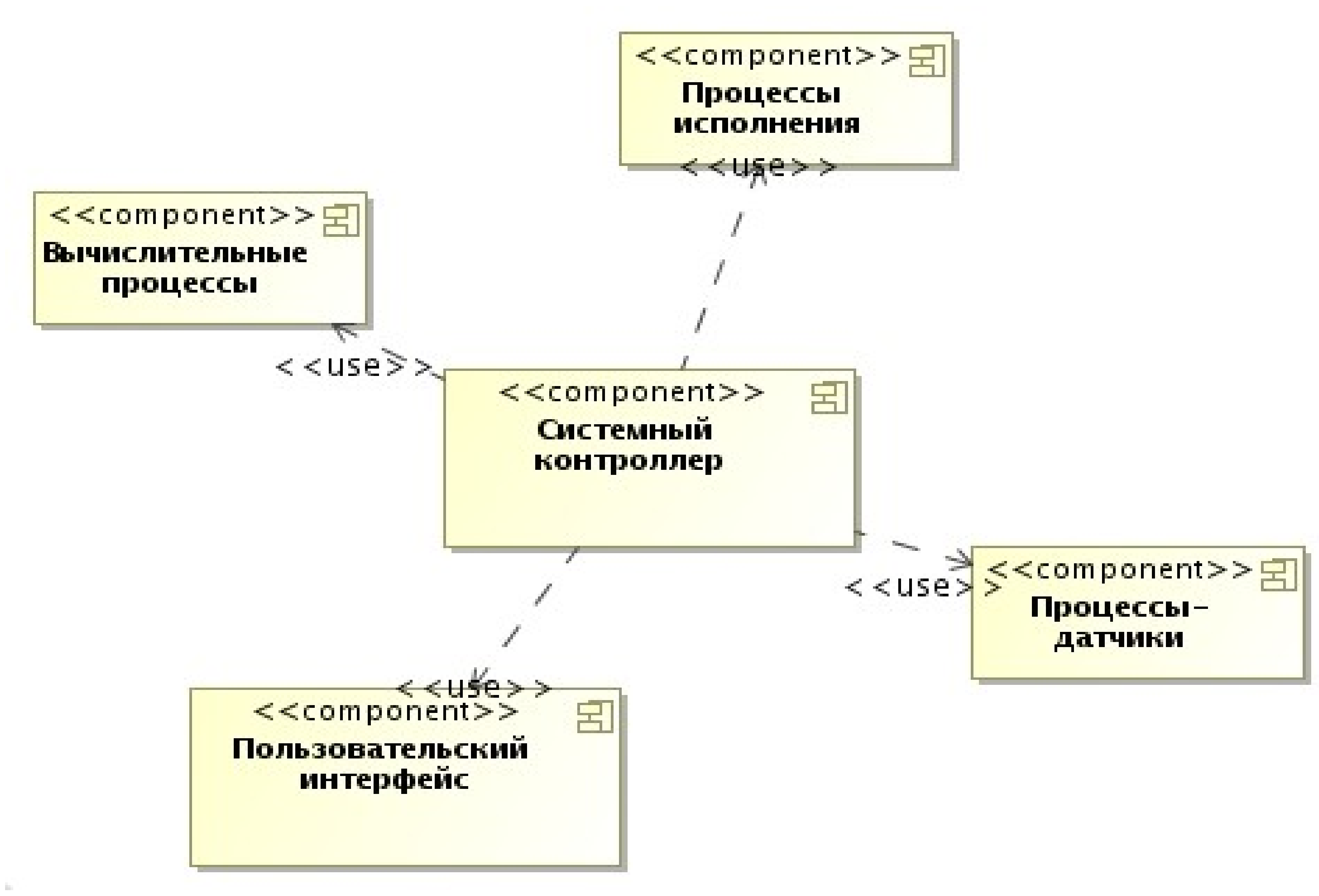
Architecture's templates

* Blackboard
* Client-server (2-tier, n-tier, peer-to-peer, Cloud Computing all use this model)
* Database-centric architecture (broad division can be made for programs which have database at its center and applications which don't have to rely on databases, E.g. desktop application programs, utility programs etc.)
* Distributed computing
* Event Driven Architecture
* Front-end and back-end
* Implicit invocation
* Monolithic application
* Peer-to-peer
* Pipes and filters
* Plugin
* Representational State Transfer
* Rule evaluation
* Search-oriented architecture (A pure SOA implements a service for every data access point)
* Service-oriented architecture
* Shared nothing architecture
* Software componentry (strictly module-based, usually object-oriented programming within modules, slightly less monolithic)
* Space based architecture
* Structured (module-based but usually monolithic within modules)
* Three-tier model (An architecture with Presentation, Business Logic and Database tiers)

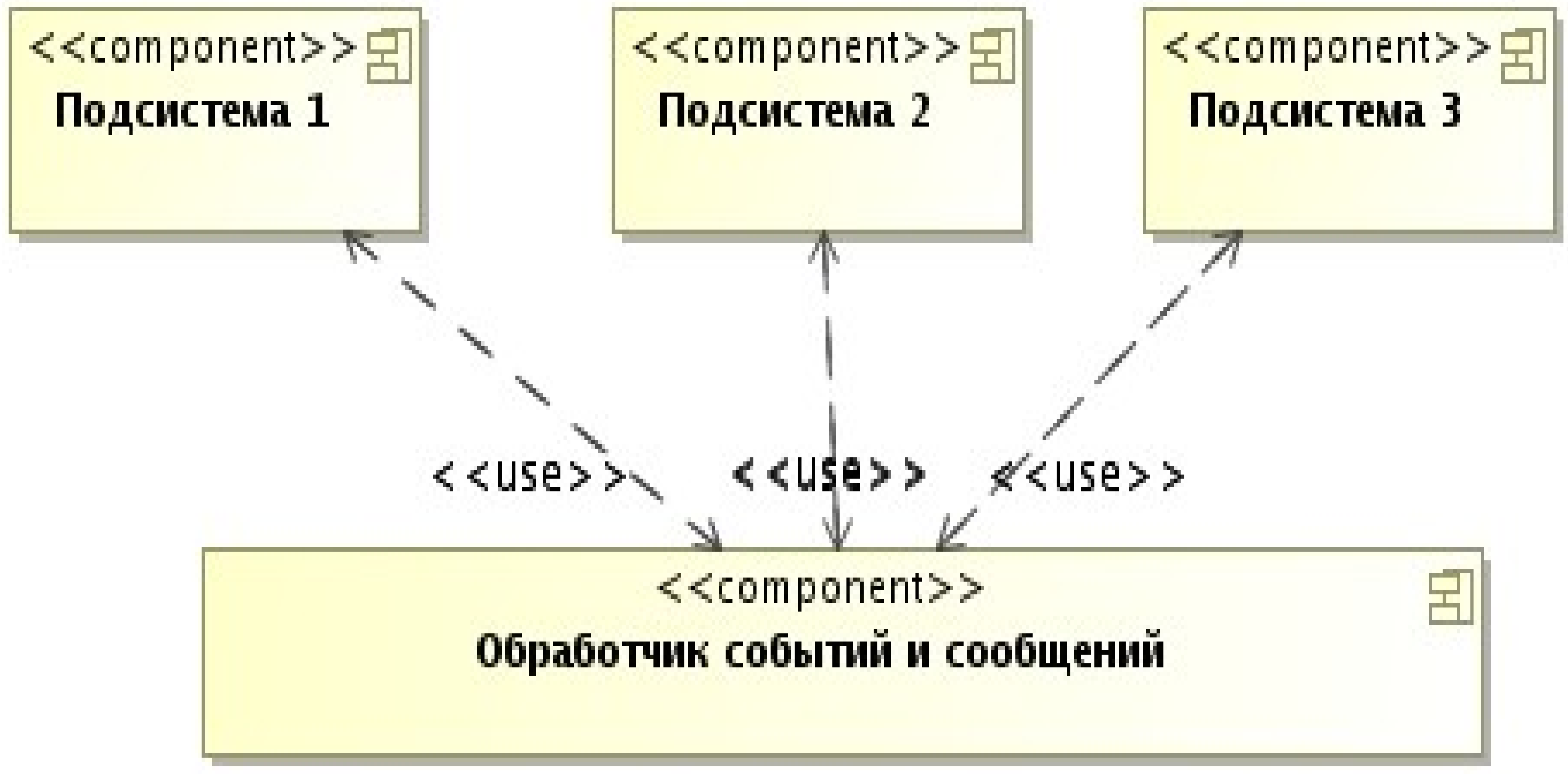
Control models



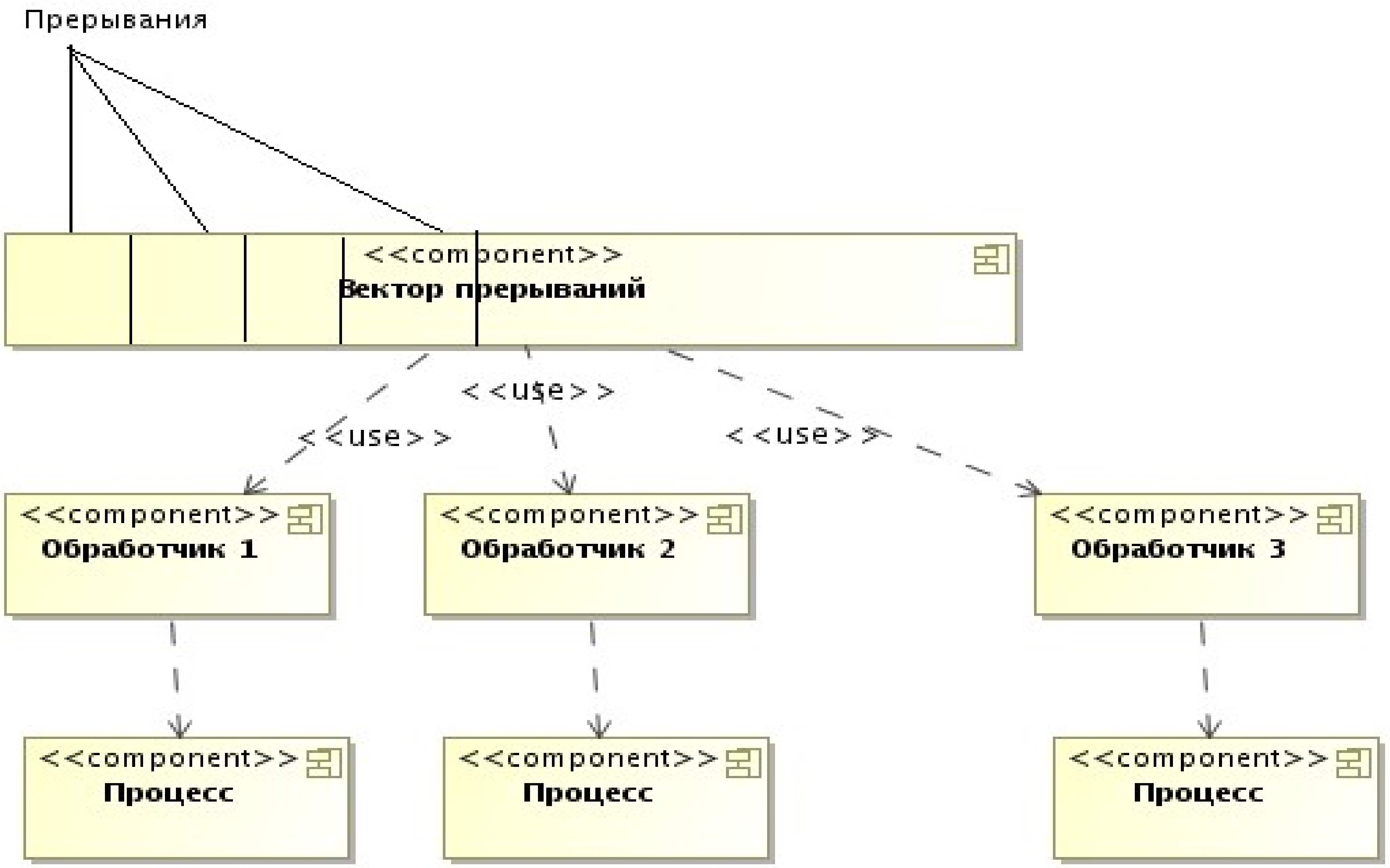
Control models



Control models



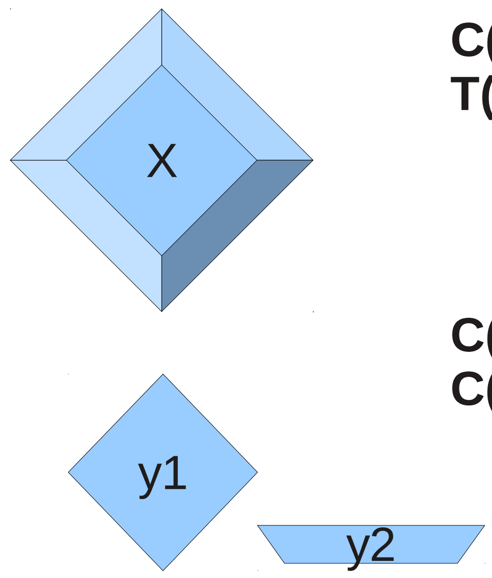
Control models



Modularity

Module — part of programm, which is a part of physical structure of a system.

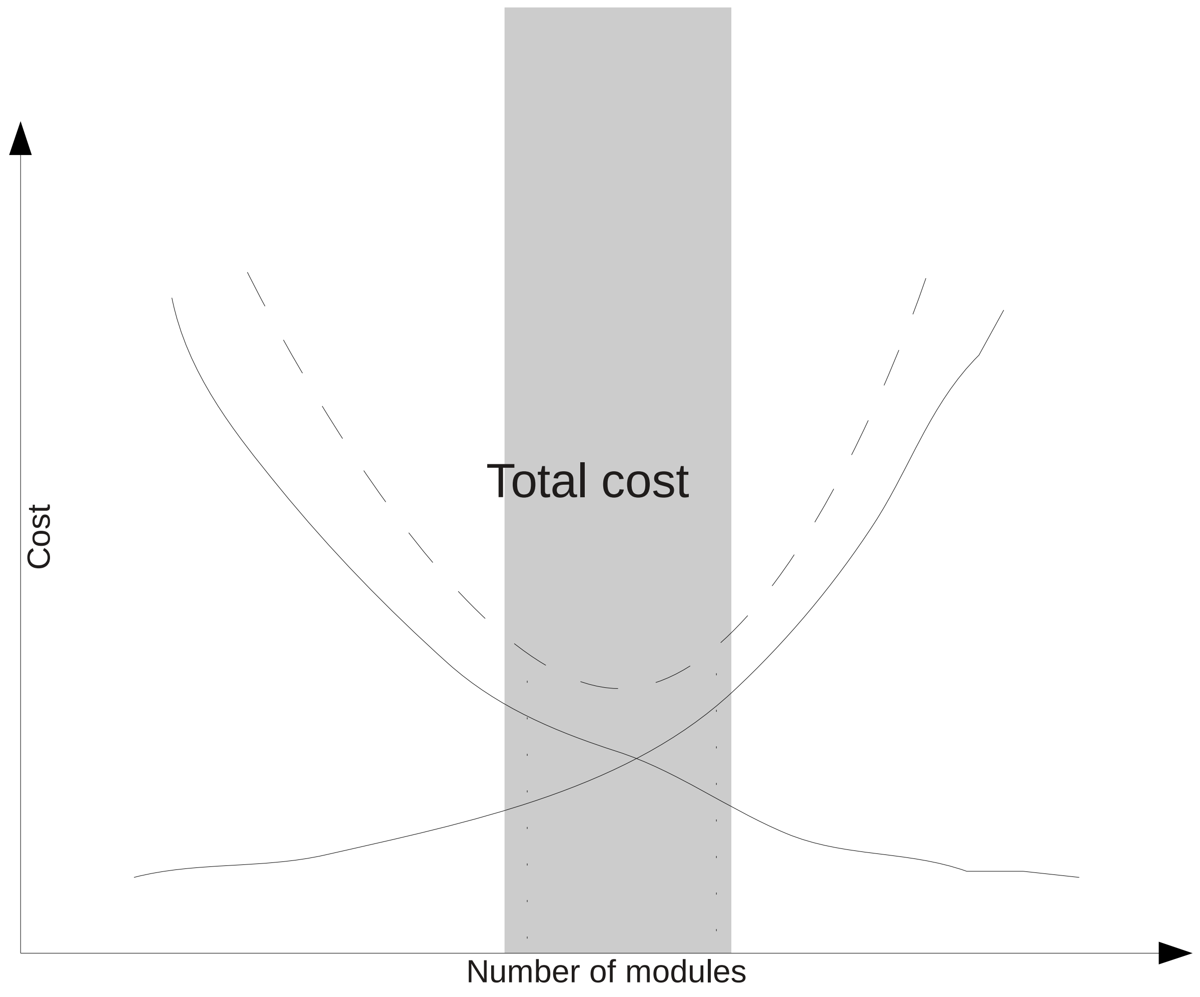
Modularity — system ability to be decompose/disassembled on number internally linked but independent parts.



C(X) — complexity to solve problem X
T(X) — time to solve problem X

C(X) > C(y1) => T(X) > T(y1)
C(X) >= C(y1) + 4*C(y2) => T(X) >= T(y1)+4*T(y2)

Modularity



Module (optimal)

Easy from outside than inside

Easy to use than to build

Information secrecy

Content(procedures, data) of modules must be hidden from each other.

It means:
All modules are independent, they must exchange only needed information, access to module structures is restricted.

Advantage: ability to use different teams, easy to modify system. An ideal module is a «black box».

Cohesion

Связность модуля (cohesion) – is a measure of how strongly-related or focused the responsibilities of a single module are. As applied to object-oriented programming, if the methods that serve the given class tend to be similar in many aspects, then the class is said to have high cohesion. In a highly-cohesive system, code readability and the likelihood of reuse is increased, while complexity is kept manageable.

Cohesion

Cohesion is decreased if:

- The functionalities embedded in a class, accessed through its methods, have little in common.
- Methods carry out many varied activities, often using coarsely-grained or unrelated sets of data.

Disadvantages of low cohesion (or “weak cohesion”) are:

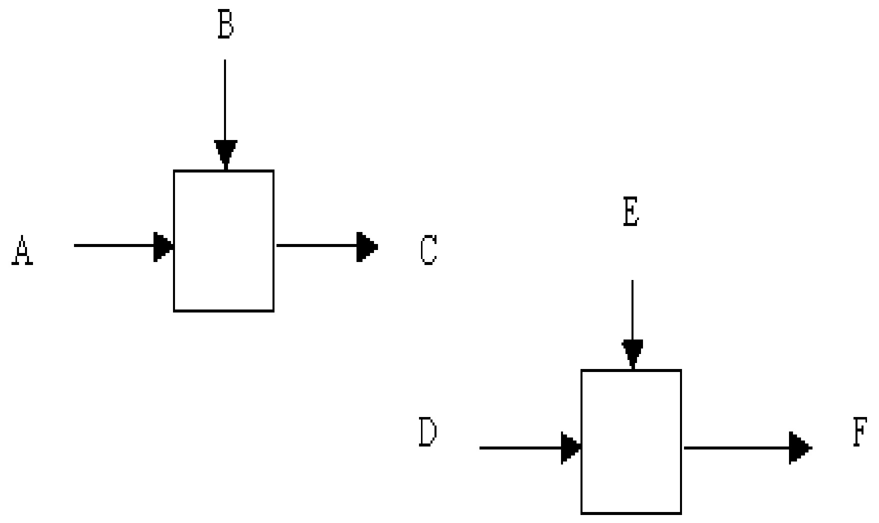
- Increased difficulty in understanding modules.
- Increased difficulty in maintaining a system, because logical changes in the domain affect multiple modules, and because changes in one module require changes in related modules.
- Increased difficulty in reusing a module because most applications won’t need the random set of operations provided by a module.

Cohesion

Существует 7 типов связности:

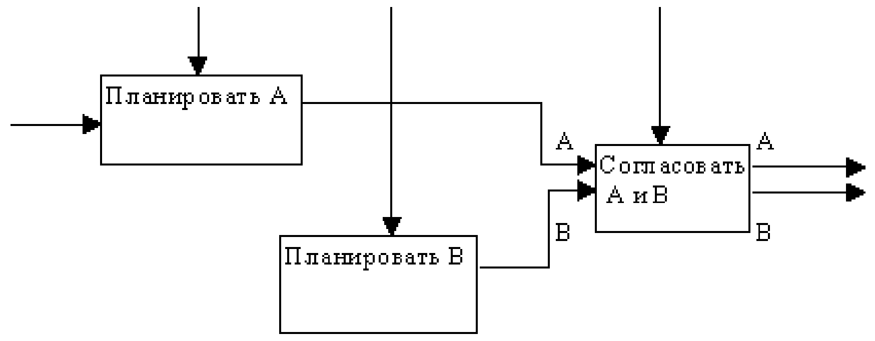
1. Coincidental cohesion (CC=0)
2. Logical cohesion (CC=1)
3. Temporal cohesion (CC=3)
4. Procedural cohesion (CC=5)
5. Communicational cohesion (CC=7)
6. Sequential cohesion (CC=9)
7. Functional cohesion (CC=10)

Coincidental cohesion



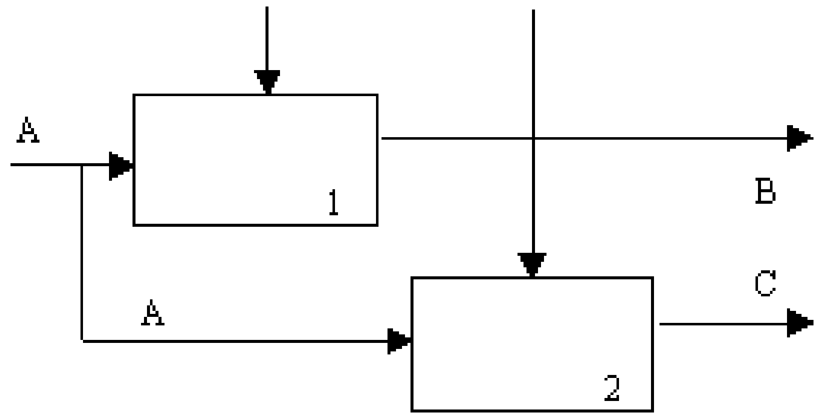
Coincidental cohesion is when parts of a module are grouped arbitrarily; the only relationship between the parts is that they have been grouped together.

Procedural cohesion



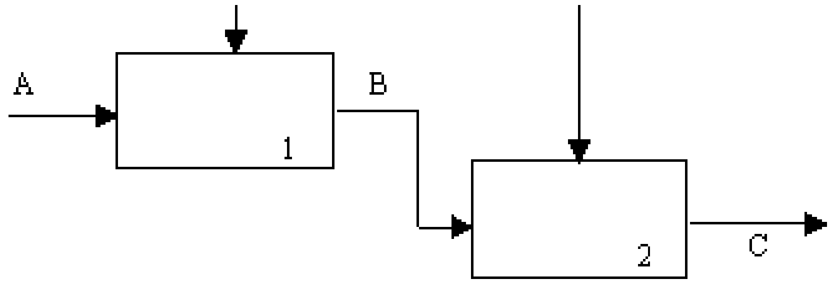
Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).

Communicational cohesion



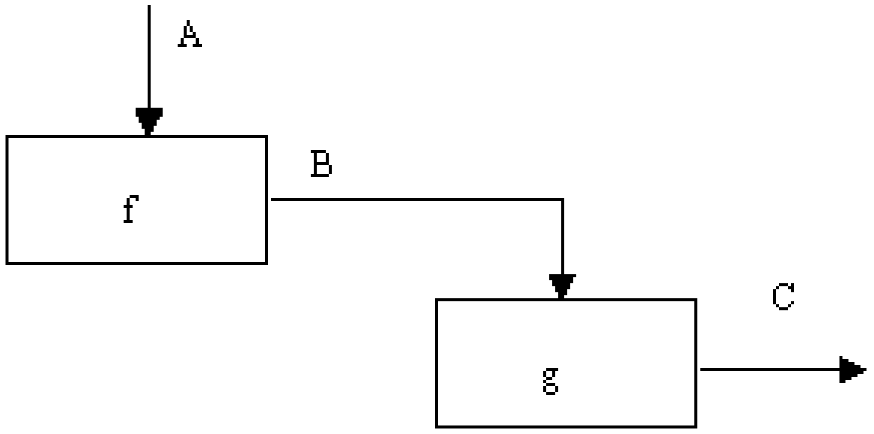
Communicational cohesion is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information).

Sequential cohesion



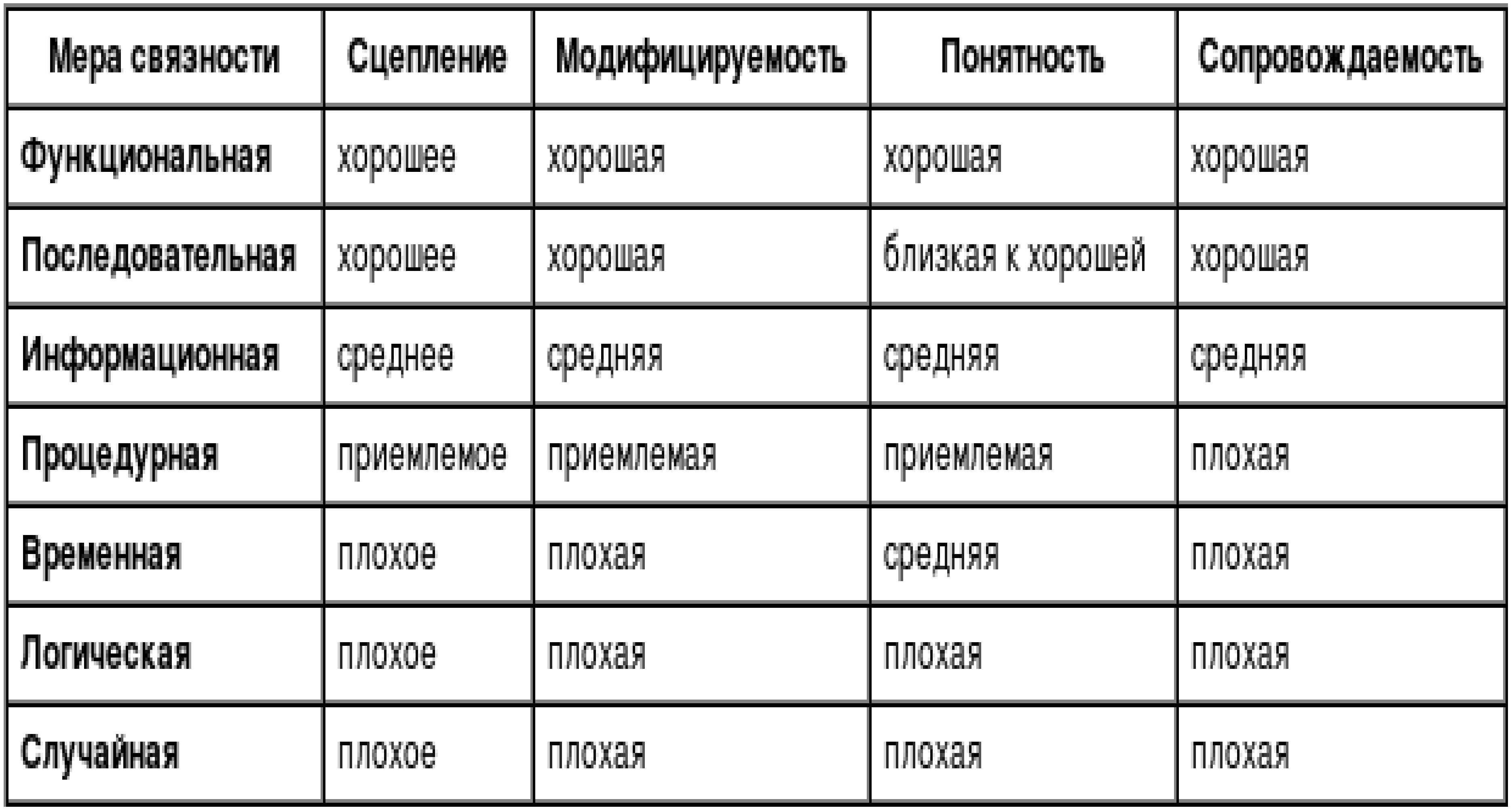
Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data)

Functional cohesion



Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module.

Cohesion



Мера связности	Сцепление	Модифицируемость	Понятность	Сопровождаемость
Функциональная	хорошее	хорошая	хорошая	хорошая
Последовательная	хорошее	хорошая	близкая к хорошей	хорошая
Информационная	среднее	средняя	средняя	средняя
Процедурная	приемлемое	приемлемая	приемлемая	плохая
Временная	плохое	плохая	средняя	плохая
Логическая	плохое	плохая	плохая	плохая
Случайная	плохое	плохая	плохая	плохая

Алгоритм определения связности

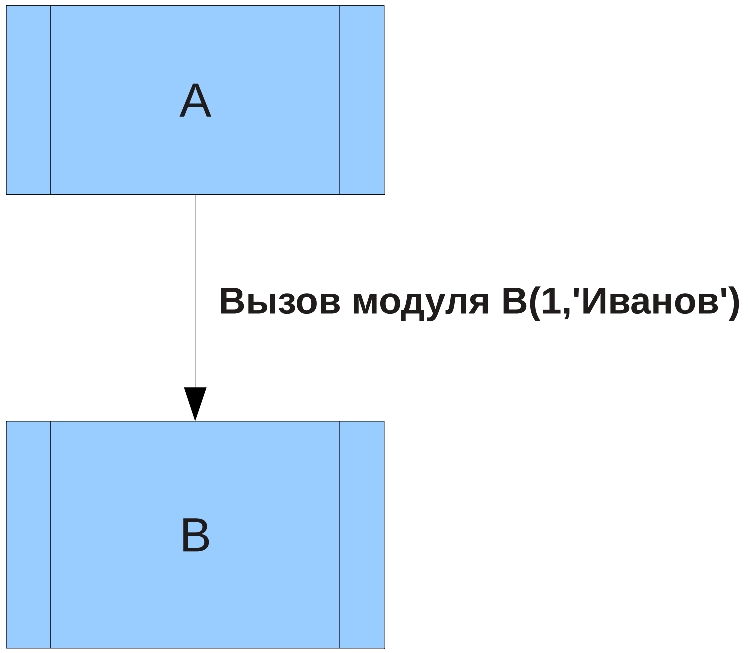
1. Если модуль — единичная проблемно-ориентированная функция, то связанности — функциональный. Иначе 2.
2. Если действия внутри модуля связаны, то 3. Иначе 6.
3. Если действия внутри модуля связаны данными, то 4. Если связаны потоком управления, то 5.
4. Если порядок действий важен, то последовательный, иначе коммуникативный. Конец.
5.
6. Если действия внутри модуля принадлежат к одной категории, то логический, иначе по совпадению.

Правило параллельной цепи — самый сильный уровень.
Правило последовательно цепи — самый слабый уровень.

Coupling

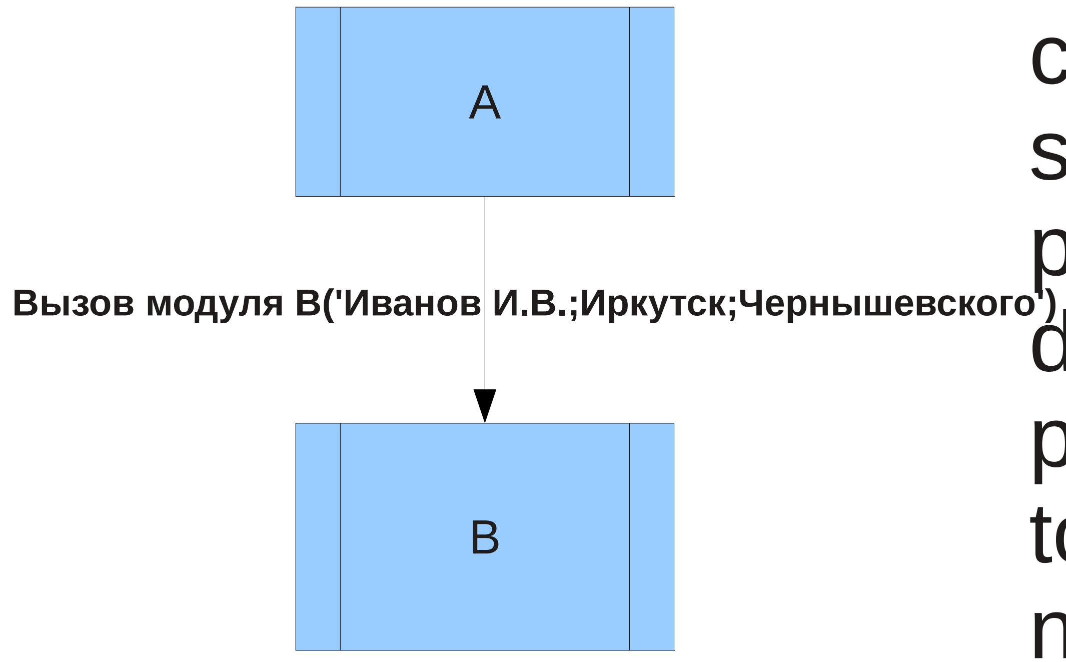
Сцепление (coupling) модуля is the degree to which each program module relies on each one of the other modules.

Data coupling



Data coupling is when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).

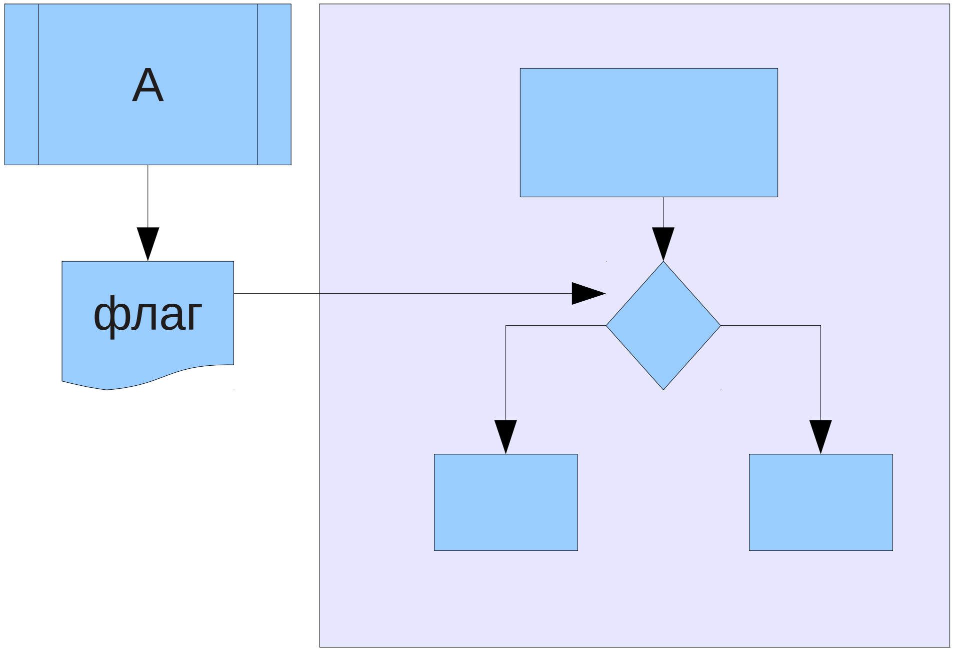
Stamp coupling (Data-structured coupling)



Stamp coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it).

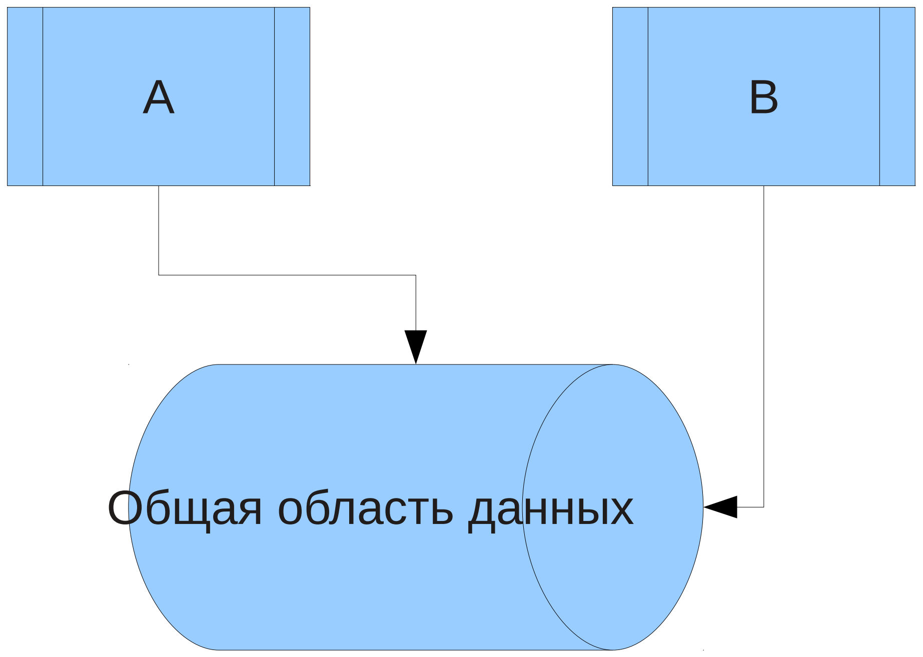
This may lead to changing the way a module reads a record

Control coupling



Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).

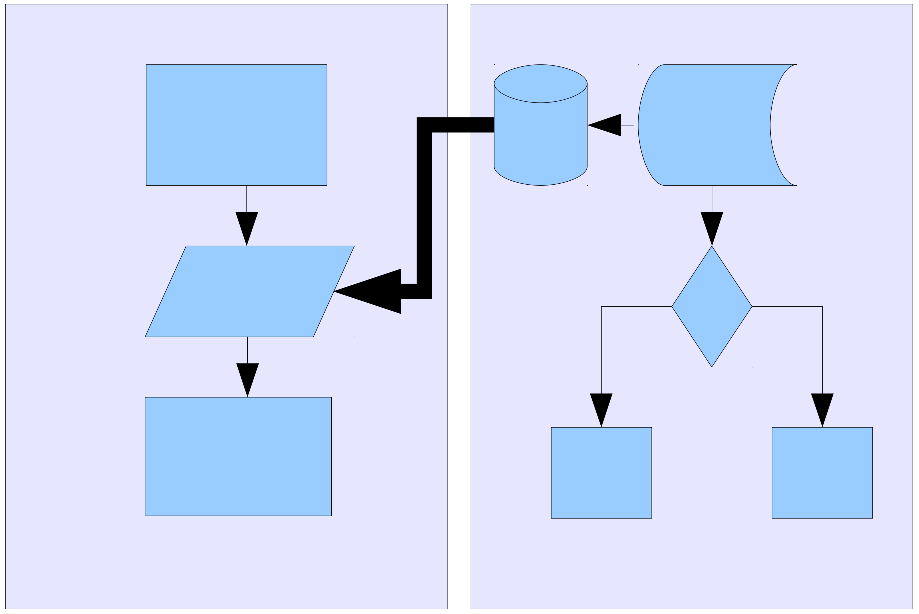
Common coupling



Common coupling is when two modules share the same global data (e.g., a global variable).

Changing the shared resource implies changing all the modules using it.

Content coupling



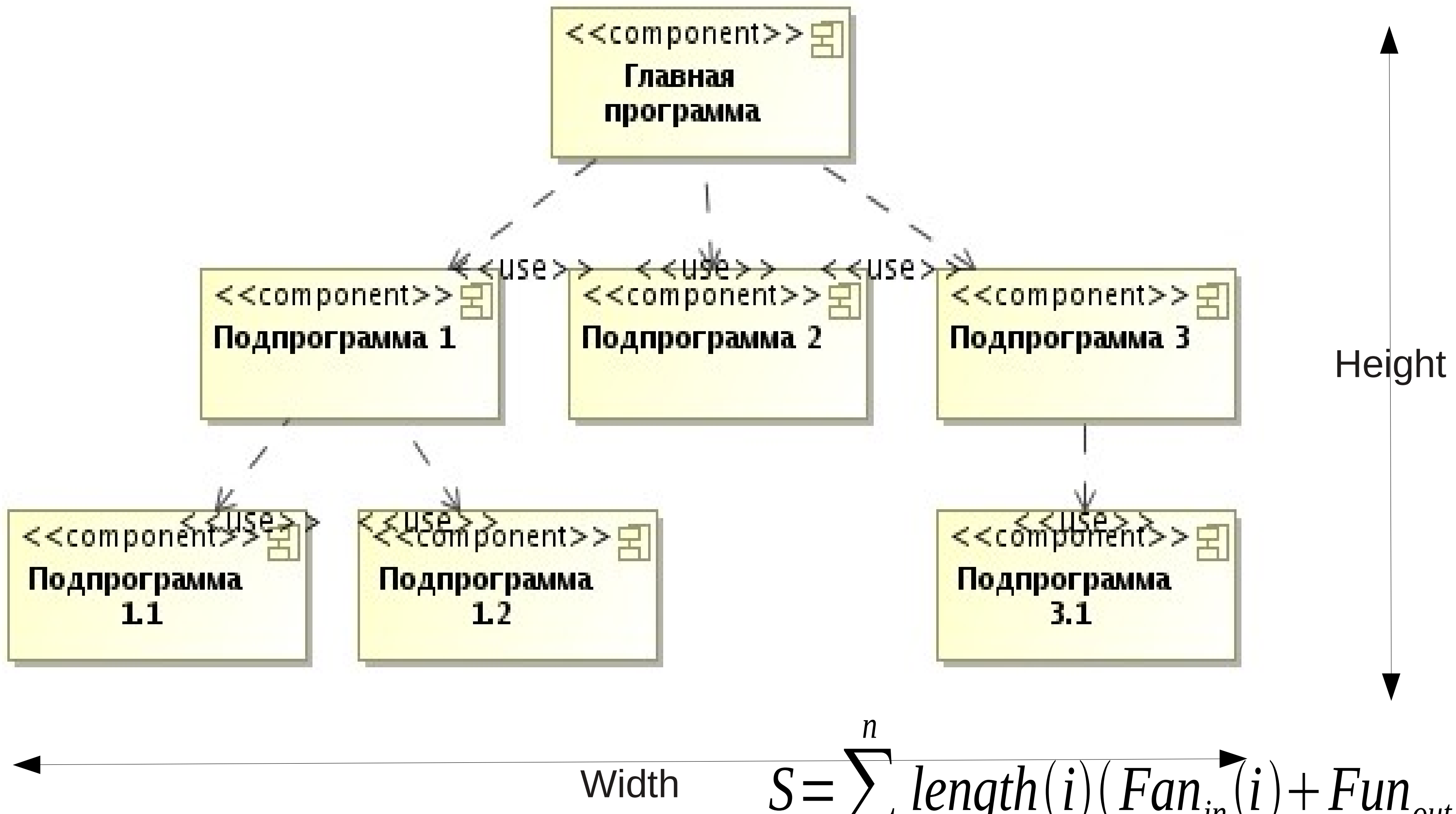
Content coupling is when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module).

Therefore changing the way the second module produces data (location, type, timing) will lead to changing the dependent module.

Compexity of software

М.Холстед (1977)
Том МакКейб (1978)

Compexity of software



$$S=\sum_{i=1}^{n} length(i)\left(Fan_{in}(i)+Fun_{out}(i)\right)^2$$