

## Rules for Appropriate Uses of Architectural Styles

Style	Nature of Computation	Quality Concerns
<b>Data Flow</b>	The input and output data of the problem to be solved are both well defined and easily identified. The computation is to produce output from the input as the direct result of sequentially transforming an input in a time-independent fashion.	Integratability from relatively simple interface between components. Reusability by interchanges of components to change functionality.
Batch Sequential	There is a single output operation sequential that is the result of reading and processing a single collection of input. The processing of the input consists of a number of sequentially connected	Reusability and modifiability, especially when performance is not a major concern and synchronisation related complexity can be reduced.
Pipe and Filter	The computation involves transformations on continuous streams of data. The transformations are incremental, i.e. applied on the elements of the stream of data. One transformation can begin before the previous step has completed.	Scalability, especially to process the input data of unbounded length of data stream. Performance, to respond to the input as soon as an element of the input data is fed into the system before the whole stream of data is obtained.
<b>Call and Return</b>	The order of computation is fixed. Components cannot make useful progress while waiting for the results of requests to other components.	Modifiability, integratability, reusability.
Layered Systems	The computational tasks can be divided between those specific to the application and those generic to many applications but specific to the underlying computing platform.	Portability across computing platforms. Reuse of an already developed computing infrastructure layer, such as operating system network packages, etc.
Main Program and Subroutine with Shared Data	The problem to be solved at a higher level of abstraction can be decomposed into a number of smaller problems at a lower level of abstraction that can be solved and their results can be put together to solve the higher level problem.	Performance, especially the space of memory used to store intermediate results can be significantly reduced without duplication when shared with components, and time to access the intermediate results can be reduced through direct access rather than other means of communication. Reusability, especially when a subroutine solves a
Data Abstraction	Computation is based on a abstraction relatively fixed variety of entities and each of these entities has a fixed number of operations.	Reusability, especially the reuse of the components that represent those computational entities that reoccur frequently in the application domain. Modifiability, especially when the representations of the computational entities are likely to change.
Abstract Data Type		Integratability, especially through careful attentions to the design of the interfaces between the components.
Object-oriented	In addition to the common features of the computation in all data abstraction sub-styles, the entities in the system have commonalities and fall into a nature hierarchy of inheritance.	Reusability and modifiability as in all data abstraction sub-styles. Moreover, it also aims at reusability and modifiability when the types of entities are relatively stable in the application domain while the system's functionality is likely to be changed, especially
<b>Independent Components</b>	One component can continue to make progress some what independent of the states of other components. The system is to be run on a network of computer systems.	Modifiability, especially when performance tuning via reallocating work among processes and reallocating processes to computers is necessary. Performance, especially to achieve maximal utilisation of the computational resource is important.
Eventbased Implicit Invocation	Computations are triggered by a collection of events. Event originators and event processors can be decoupled.	Flexibility for the modification system's functionality especially enhancement of functionality. Scalability in the sense of adding processes that are triggered by the events already detected/signalled in the system. Modifiability for change the way that events are
Client Server	Computation can be reviewed as providing services to a number of users who share the access to the information and/or computation resources. Computation tasks can be divided between instigators of service requests and executors of those requests or between producers and consumers of data.	Scalability in terms of the number of users who can have access to the system simultaneously as well as from a wide geographic area. Modifiability, especially when users' interface may change and/or new functionality is likely to be added into the system as new services.
<b>Data Centered</b>	The central issue in the computation is the storage, representation, management and retrieval of a large amount of interrelated long-lived data. The data are usually highly structured. The order that components are executed in is determined by a stream of incoming requests to access/update the data.	Scalability in terms of the amount of data to be stored and processed by the system. Modifiability to change the producing and processing the data while the structure of the data remains relatively stable.
<b>Virtual Machine</b>	Data consists of two parts; one is the information to be processed and the other controls how the first part is to be processed.	Portability of the processing of the data in different hardware and software platforms.

## Behavioural Features of Architectural Styles

Style	Synchronicity	Data Access Mode	Data Flow Continuity	Binding Time
Data Flow	Asynchronous	Passed	Sporadic or Continuous, Low or High Volume	Any
Pipe-and-Filter	Asynchronous	Passed	Sporadic	Code, Compile, Invocation
Batch Sequential	Lockstep (Sequentially)	Passed, Shared	Sporadic High Volume	Any
Independent Components	Synchronous, Asynchronous	Any	Sporadic low volume	Any
Event-based Implicit invocation	Asynchronous	Multicast	Sporadic low volume	Any
Communicating Processes	Any except sequential	Message	Sporadic low volume	Any
Call-and-Return	Lockstep	Passed, Shared	Sporadic low volume	Any
Layered Systems	Lockstep	Passed	Sporadic low volume	Any
Abstract Data Type	Lockstep (sequential)	Passed	Sporadic low volume	Code, Compile
Object-Oriented	Lockstep (sequential)	Passed	Sporadic low volume	Code, Compile, Run-time
Data-centered	Asynchronous	Shared, Passed, Multicast	Sporadic low volume	Code
Virtual Machine	Lockstep (Sequential)	Shared	Continuous	Code

## Structural Features of Architectural Styles

Style	Components	Connectors	Control Topology	Data Topology	Interaction Topology	Interaction Direction
Data Flow	Processing Elements	Data Flows	Arbitrary	Arbitrary	Identical	Same
Pipe-and-Filter	Filters (Transducers)	Pipes (Data Streams)	Directed Graphs	Directed Graphs	Identical	Same
Batch Sequential	Phases (Stand-alone Programs)	Batch Data (e.g. Files)	Linear	Linear	Identical	Same
Independent Components	Processes, Modules	Communication protocols	Arbitrary	Arbitrary	Possible	Either
Event-based Implicit invocation	Processes and event handler	Events	Arbitrary	Arbitrary	Identical	Same
Communicating Processes	Procedures (subroutines) and Data stores	Procedure calls and Data accesses	Arbitrary	Arbitrary	Not always	Any
Call-and-Return	Procedures (subroutines) and Data Stores	Procedure calls and Data Accesses	Arbitrary	Arbitrary	Not Always	Any
Layered Systems	Layers (sets of procedures)	Procedure Calls	Hierarchical	Hierarchical	Often	Any
Abstract Data Type	Managers (Abstract data type)	Procedure Calls	Arbitrary	Arbitrary	Identical	Same
Object-Oriented	Manager (objects/ classes)	Procedure Call	Arbitrary	Arbitrary	Identical	Same
Data Centered	Data store and Computation clients	Data access, Data queries	Star	Star	Possible	Opposite, if structures and identical
Virtual Machine	Data memory, State memory, Program memory, Interpretation engine	Data paths	Fixed	Fixed	No	Not applicable