Glossary of Software Architecture Terms

Roman Kern <rkern@tugraz.at>
Institute for Interactive Systems and Data Science,
Graz University of Technology

Version 1.2.1, 2020/21

Abstract

This documents represents a list of important terminology for the software architecture course. Please beware that this is not an exhaustive list and some important terms might be missing. Although this document can be used a learning aid, it is by itself not sufficient for a proper understanding of the content of the course. In the case you would like to see additions to this document, please inform the author via e-mail.

Addressability Ability to model the scoping information in a consistent way.

Afferent Coupling The number of classes outside a category that depend upon classes within this category.

Anarchic Scalability Goal to achieve for systems which are distributed with scalability in mind and the components are developed independently.

Application Components Components in the implementation architecture capturing the domain level responsibilities.

Approximate Accuracy Instead of delivering an exact result, an approximation of a result is computed.

Architectural Smell Indicator for a potential quality impairment in the software architecture, e.g., cyclic dependencies.

Aspect Oriented Programming Programming paradigm where cross-cutting concerns are factored out, e.g. separate code for logging.

Asynchronous Communication Exchange of data between components, where the requester does not wait for the response.

At-Least-Once In a streaming setting an event is guaranteed to be processed, but might be processed multiple times, e.g. no missed events, but duplicates, when counting.

Batch-Sequential Special case of the data-flow architecture, where each filter waits for the completion of its predecessor.

Behaviour Description of how the system achieved what it is intended to do.

Blackboard Type of data-centric architecture, where clients connect to a central server, which is responsible for data management and informs the clients of changes within the data model.

Blob A single component with too many responsibilities, e.g. a component to rule them all.

Callback Communication Exchange of data between components, where the requester does not wait for the response, but instead gets the answer asynchronously via means, which have been established beforehand.

CAP Theorem Consistency, Availability and Partitioned - a distributed system cannot completely achieve all these three criteria, a trade-off is necessary.

Code on Demand In a client server setting, the server might respond with an executable/interpretable script in order to extend the functionality.

Code Smell Indicator for a potential quality impairment in the code, e.g., god class, blob.

Command Cluster A set of components where each of them only contribute few responsibilities.

Component Stereotype Assign a well known semantic to a component, e.g. user initiated components in the UI.

Conceptual Architecture Model of the architecture focusing on domain level responsibilities, i.e. what groups of functionality does exist and how do these groups interact to achieve a certain goal, e.g. a single use case.

Concurrent Subsystem Abstraction of a number of components, which can be seen as a separate independent system, e.g. a database server.

Container Components Components in the implementation architecture designed to give a execution environment (usually for application components).

Cohesion Degree on how strong the responsibilities of component relate to each other, i.e. how clear is the separation of the responsibilities.

Constructive Cost Model (COCOMO) Method to estimate the cost of building a (software) system.

Coupling Degree on how strong components depend on each other.

Cross-Cutting Concerns Requirement not covered by a single component, but each component might be effected.

CRUD Create, Retreive, Update & Delete

Cyclomatic Complexity Measure for the complexity of a system/program based on the control flow graph.

Data-Centered Architecture An architecture with the goal to achieve data integrity, typically by choosing a central component for data management, e.g. a database.

Data-Flow Architecture An architecture with the goal to achieve decoupling between components, by defining a data flow (pipes) with a series of transformations (filters).

Design by Contract Software design principle guided by the definition of rules how the objects interact, e.g. via preconditions, postconditions and invariants.

Development-Time Binding Binding of services during build time, i.e. the developer specifies which services are used.

Distributed Computing Systems designed to execute code in parallel, with a low degree of shared resources, e.g. run an individual nodes.

Efferent Coupling The number of classes inside a category that depend upon classes outside of this category.

Exactly-Once In a streaming setting an event is guaranteed to be processed exactly once, e.g. no duplicates or missed events when counting.

Executable Prototype Prototype designed to serve as a starting point of the system development, where iteratively more and more functionality (and components) are added.

Execution Architecture Model of the architecture focusing on the runtime aspects, i.e. what types of parallel executing components exist.

Fault Trees Model Models of the system designed to capture the dependencies between components in terms of error propagation, e.g. component A fails if component B or C fails.

Functionality Description of what the system can do.

Idempotence The same operation has the same effect applied once or multiple times.

Implementation Architecture View of the architecture focusing on how the system is build.

Information Flow Connector used in the conceptual architecture used to model what type of information is needed for component to accomplish its responsibilities.

Infrastructure Components Components in the implementation architecture designed to make the system run.

Instability Index that indicates the stability of a component/sub-system.

Interceptor Pattern Architecture pattern where components allow other components to register themselves. As soon as a certain even occurs the registered components are called back.

Kappa Architecture Architecture designed for distributed systems processing data streams.

Lambda Architecture Architecture designed for large data driven systems consisting of a batch layer, service layer and speed layer.

Law of Demeter Principle that states, that methods should only be invoked that are directly accessible (to reduce the amount of changed entities in case of refactoring).

Layered Architecture Architecture pattern that organises the components into individual layers (on top of each other) with restrictions on the connections between the components.

Lines of Code The count of source code lines, excluding comments, empty lines, etc.

Liskov Substitution Principle Contract between a client and a class, which guides when an object can be substituted (in regard to the pre- and postconditions).

MeTRiCS The non-runtime quality attributes: maintainability, evolvability, testability, reusability, integrability, configurability, scalability.

Micro-Batch In a streaming setting multiple events are collected and processed as group instead of individually (improves throughput).

Micro-Service A system consistingdistributed, streaming of a number of independently deployable services.

Mock-Up Schematic sketch, on how the user interface might look like and how the interaction might look like.

Model Abstraction of the system, focusing on a single (or multiple) aspect, e.g. security, domain level responsibilities, executions.

Notification Architecture Architecture patterns with the callback mechanism as the central element to model the process flow.

One-at-a-Time In a streaming setting each events is processed individually (improves latency).

Open-Closed Principle Design software entities in such a way, that is easy to extend them without the need to modify them.

Parallel Computing Systems designed to execute code in parallel, with a high degree of shared resources, e.g. memory.

Peer to Peer Architecture based on the client-server pattern, where each nodes is client and server at the same time.

Pipes and Filters Data-flow architecture based on components (filters, the processing unit) that are connected via pipes (communication channel), ideally the filters are independent from each other.

Pipeline Simple variation of the pipes and filter architecture, where the topology of the filters represents a single chain.

PURS The runtime quality attributes: performance, usability, reliability, security.

Push/Pull Notification A push notification contains all relevant information for further processing, the pull notification just contains the minimum (if the listener requires additional information it needs to pull it from the source).

Principle of Least Knowledge Minimise dependencies via keeping references to other components minimal, see also Law of Demeter.

Quality Attributes Key characteristics of a software system, including the run and the build time.

Repository Type of data-centric architecture, where clients connect to a central server, which is responsible for data management.

REST Representational State Transfer

RESTful A set of guidelines to help independent components (services) to work together (interoperable).

Rich Client Clients that implements a large part of the business logic, typically desktop applications.

ROA Resource-Oriented Architectures.

RPC Remote Procedure Call

Runtime Binding The services, which are used by the system are defined during runtime, i.e. at startup the services are looked-up.

SaaS Software as a Service

Separation of Concern Principle to design systems in a way that the individual responsibilities (e.g. functionality) are clearly separated and assigned to individual components.

Share Nothing Architecture Highly scalable architecture based on distributed components, that operate independent from each other.

Synchronous Communication Exchange of data between components, where the requester does wait for the response.

SOA Service Oriented Architectures

SOAP Simple Object Access Protocol - a protocol designed to exchanged structured information between services (used for method calls).

Supernode Peer node within a peer-to-peer architecture that serves additional purposes.

Technical Prototype Prototype designed to test out a single, isolated aspect of the system (proof of concept). Should be not be used as base for the development of the system.

Time to Market Time it takes to ship the (finished) product (system).

Thin Client Clients that implement only a small part of the application logic, typically (traditional) web applications.

Types of Models There are a number of ways how models can be categorised. One of the most important is the distinction between structural and behavioural models, i.e. between the static of the system and the dynamics of the system.

Uniformity Goal to keep the communication in a consistent way, e.g. data structures and protocol is shared between many components.

WSDL Web Services Description Language - language designed for the specification of interfaces, in particular for services.