Brief Glossary of Terms

Architecture Description Language
Formal language for describing software architecture. See section 3.2.2.

Backward Compatibility
A component is backward compatible with an older version of itself if the new version can be used in all contexts where the old version can be used. See section 4.2.2.

Binding
The act of connecting two interfaces so that two implementations can work together. See section 3.3.4.

Cable
A binding between two interfaces. See section A.4.9.

Component
A unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. See section 3.3.1.

Composition
Technique to handle diversity by identifying the points where software is the same for different products, and factoring those out as components. See section 5.2.5.

Compound Component
A component that contains and encapsulates (instances of) other components. See section 3.3.5.

Configuration Management
Classical technique to handle product variation and evolution. Works at compile-time only and is often architecture agnostic. See section 6.4.

Diversity
The variation between members of a product family or product population. See section 7.3.2.

Diversity Interface
Koala construct for specifying and handling diversity. See section 3.4.4.

Diversity Spreadsheet
Technique for specifying diversity parameters of subcomponents in terms of the diversity parameters of the compound component. See section 3.4.5.

Downward Compatibility
A client is downward compatible with a server if it also runs on an older version of the server. See section 4.2.2.

Evolution
The change of software artifacts over time. See section 3.5.

Fiber
See Function Binding.

Forward Architecting
The construction of an architecture followed by the systematic derivation of an implementation, so that architecture controls the implementation. See section 2.1.

Forward Compatibility
A component is forward compatible with a newer version of itself if the old version can be used in all contexts where the new version is used. See section 4.2.2.

Function Binding
An implementation in Koala of a function in an interface. See section 3.4.2.

Horizontal Communication
Technique for writing reusable control software by letting drivers communicate directly with each other, rather than through managers. See Chapter 8.

Independent Deployment
The phenomenon that components are not designed for specific products, but rather have their own evolution in time. See section 4.2.1.

Interface
A small set of semantically related functions. See section 3.3.1.

Interface Compatibility
The required relation between two interfaces in a legal binding. See section 3.4.1.

Interface Definition
Description of the syntax and semantics of an interface. See section 3.3.2.

Interface Subtyping
See Interface Compatibility.

Mandatory Interface
An interface that must be connected at its tip. See section 3.4.7.

Module
A unit of code in Koala. See section 3.3.6.

Optional Interface
An interface that may but need not be connected at its tip. See section 3.4.7.

Part-of Relation
A fundamental relation in software architecture, a.k.a. as decomposition. See section 2.3.3.

Partial Evaluation
The conversion of a program to a more specialized program by filling in certain parameters. See section 3.4.3.

Portability
The use of a client with a different server. See section 4.6.

Product Family
A set of products with many commonalities and few differences. See section 7.2.2.

Product Population
A set of products with many commonalities but also with many differences. See section 7.2.4.

Product Line
A proactive and systematic approach for the development of software to create a variety of products. See section 5.1.

Provides Interface
An interface that provides access to functionality provided by a component. See section 3.3.1.

Relation Partition Algebra, or RPA
A mathematical framework to model software architecture and implementation, allowing to reason about each individually and both in their mutual relation. See section 2.3.

Requires Interface
An interface through which a component accesses functionality provided by its environment. See section 3.3.1.

Resource Constraints
The phenomenon that resources such as memory and processing time are not infinite. See section 7.2.1.

Reusability
The use of a server with a different client. See section 4.5.

Reverse Architecting
To infer the architecture from an implementation with the purpose to reason about or modify the implementation. See section 2.1.

Software Architecture
The specification of a design. See section 2.2.

Switch
A Koala construct for creating flexible (compile- or run-time) binding. See section 3.4.6.

Third-Party Binding
Construction where components A and B have no knowledge of each other but instead are bound by a third component C. See section 5.5.1.

Upward Compatibility
A client is upward compatible with a server if it also runs on a newer version of the server. See section 4.2.2.

Variation
Common technique to handle diversity, by identifying the points where software differs for different products. See section 5.2.5.