Glossary

Abstraction

Abstraction is a powerful concept of hiding details and concentrating on essential information only. It refers also to a domain in the three-dimensional model of development in the KobrA method in the abstraction/concretization dimension.

Acceptance Testing

Acceptance testing refers to the test activities on the final application according to the user's or customer's requirements.

Application Engineering

Application engineering refers to all activities that deal with assembling and integrating existing components into a single application. It refers also to the instantiation of a final product out of a product family in product line engineering.

Assertion

An assertion is a boolean expression that defines the necessary conditions for the correct execution of an object or a component.

Average Case Execution Time

Average runtime of a program or a component's interaction over all feasible runs according to varying input parameters.

Basic Contract

A basic contract includes the signatures that a component provides or requires in terms of invocable operations, signals that a component sends or receives, and exceptions that a component throws or catches. It is also termed a syntactic contract.

Behavioral Contract

This is the same as the basic contract, plus it defines the overall functionality and behavior of a component in terms of pre- and postconditions of operations and externally visible, provided and required state transitions.

Behavioral Model

This is one part of a KobrA component specification which describes in terms of statechart diagrams or statechart tables how a component behaves in response to external stimuli.

Best Case Execution Time

This is the shortest feasible runtime of a program or a component over all feasible runs according to varying input parameters.

Built-in Testing

Built-in testing refers to all software artifacts that are built into a component, such as assertions, testing interfaces and tester components, to support testing activities.

COTS Component

A commercial off-the-shelf component is a ready-to-use physical component from a third party that can be incorporated into an application.

Certification

Certification is the process, carried out by a third party, of assuring that a vendor's claims concerning a product (e.g., a component) are justified. The third party may be an independent certification organization, the owner of a broker platform that distributes components, or a user group that publishes the opinions of its members.

Client

A client of a component instance *SI* is any other component instance that invokes the operations of *SI*. A client of component *S* (development-time description) is any other component with instances that are clients of *S's* instances.

Component

A component is a reusable unit of composition with explicitly specified provided and required interfaces and quality attributes, which denotes a single abstraction and can be composed without modification. In this book, the term component refers to a development-time description of a unit of composition, in contrast to a component instance or a physical component.

Component Adapter

A component adapter is an additional component in its own right that is inserted between two initially alien components to transfer what one component "means" into something that the other component "understands" and vice versa. It adapts the deviating contracts of two components so that they can interact in a meaningful way.

Component Contract

The interaction of components is based on the client/server model and the mutual agreement that the client meets the precondition of the server (client's contract) in order for the server to guarantee its postcondition (server's) contract. If the client fails to meet the precondition of a service, the server is not bound to its postcondition. This is a mutual agreement on which all component interactions are based, and termed a component contract.

Component Deployment

Component deployment is referred to as the act of configuring and running a component instance in its runtime environment.

Component Engineering

Component engineering represents all development activities that deal with developing individual components, rather than assemblies of components, that is applications.

Component Framework

This is an assembly of components that represents a product family. It can be instantiated, i.e., other components added, to retrieve the final application.

Component Instance

This is a component at runtime, according to the same idea that an object is an instance of its corresponding class.

Component Meta-model

This is the organization of a component expressed in terms of a modeling notation (a model of a model). A component meta-model describes the parts of a component description in a graphical form.

Component Realization

This is a collection of descriptive documents defined by the KobrA method that describe how the private design of a KobrA component fulfills its specification. The documents comprise structural models, activity models, interaction models, and a quality documentation.

Component Specification

This is a collection of descriptive documents defined by the KobrA method that describe a component's interfaces. The documents comprise structural models, behavioral models, functional models, non-functional specifications, and a quality documentation.

Component State

A state is a distinct setting of a component's internal attributes that determines the component's externally visible behavior.

Component Wrapper

This is the same as a component adapter.

Component-Based Development

This term represents all activities that deal with the development of software applications from existing reusable parts. It is separated into two subactivities, component engineering dealing with how individual components need to be built to be reusable, and application engineering dealing with how components need to be integrated into final applications.

Composition

This is the act of building or composing an application from existing parts. It is also a domain in the three-dimensional development model of the KobrA method in the composition/decomposition dimension.

Concretization

This is the act of turning an abstract software application described in the form of models into a more concrete software system, for example, described in the form of source code. It is also a domain in the three-dimensional development model of the KobrA method in the abstraction/concretization dimension.

Conformance Map

A conformance map describes a COTS component's externally visible features in terms of a mapping between the notation of the reused COTS component and the notation of our own development process, for example, KobrA and UML.

Containment

This is a development-time association defining a parent/child relation between components.

Context Realization

This describes the properties of a component's environment according to the KobrA method. It comprises all the models of a normal component realization, and, in addition, enterprise models that capture the abstract business for the intended system.

Decision Model

In a product family development, a decision model describes the groups of features possessed by the component for different members of the product family. It identifies the variation points and the corresponding resolution space in terms of effects of each resoltution option on the other models of the component.

Decomposition

This is the act of separating an application into finer-grained parts that are individually easier to tackle. It refers also to a domain in the three-dimensional development model of the KobrA method in the composition/decomposition dimension.

Deployment Environment

This is the runtime platform in which an application is eventually executed, in contrast to the development environment.

Development Method

A development provides a number of concepts, techniques, tools, and procedures that guide a software development in what engineers have to do, how they have to do it, and when they have to do it.

Embodiment

This is the act of transferring an application from a more abstract representation into a more concrete representation along the abstraction/concretization dimension of the KobrA method.

Encapsulation

Encapsulation is a key concept of object and component technology based on abstraction. Encapsulation separates what a component does from how it does it.

Execution Time Analysis

This is concerned with all activities carried out in order to assess a component's best-, average-, and worst-case timings.

Export Interface

This is the collection of services that a component provides. It is also termed provided interface.

Framework Engineering

This refers to all activities carried out to devise the core of a product family, the component framework.

Functional Model

This is the part of a component specification which describes the externally visible effects of the operations supplied by a component. Each component operation has one functional model.

Genericity

This is a domain in the three-dimensional development model of the KobrA method that refers to the genericity/specificity dimension. This dimension deals with the framework and application engineering activities in a product family development.

Import Interface

This is the collection of services that a component requires from its environment or runtime platform. It is also termed required interface.

Information Hiding

The principles of abstraction and encapsulation lead to the principle of information hiding. It refers to the principle that objects and components are fully described and usable according to their externally visible features without access to their internal implementations.

Instantiation

Instantiation refers to the act of turning an abstract entity into a concrete entitiy, that is, turning a component into a component instance, and instantiating a product family core representing a generic system into a concrete product.

Integration Test

This is a test that assesses component interactions according to their provided and required contracts when components are integrated to compose the final system.

Interface Definition Language (IDL)

This is a notation used by many contemporary component platforms to establish syntactic component interactions.

Invocation History

The invocation history refers to the sequence of operation invocations on a component, including their input parameter values, to bring a component into a distinct state.

KobrA Component

This is a component in accordance with the component model of the KobrA method, including a component specification, a component realization, and quality attributes with the required models.

Logical Component

This refers to an abstract component in a modeling hierarchy, in contrast to a concrete physical component in the runtime environment.

Logical State

This is an abstract externally visible description of the concrete internal attribute settings of a component. Logical states represent value domains of concrete physical component states. One logical state represents a number of internal physical states.

Middleware

The middleware refers to what is commonly understood as the collection of all services provided by contemporary component platforms. These services have not yet made it into the operating system environment, so that they reside in the middle between the operating platform and the application level.

Model-Based Testing

This refers to all activities and techniques that deal with the derivation of test artifacts from models.

Model-Driven Architecture

This is one of OMG's proposed approaches to future system engineering in which entire applications are modeled in graphical notations and then transformed automatically into executable entities.

Model-Driven Development

System development according to the principles of the model-driven architecture.

Normal Object Form (NOF)

This is a predefined implementation profile that contains elements of the core features of object-oriented programming languages, and thus can be simply translated into any mainstream object-oriented language.

Object Request Broker (ORB)

This is the part of a CORBA platform responsible for accepting and redirecting component requests. It works pretty much like an Internet proxy.

Physical Component

This is a concrete binary executable version of an abstract component.

Physical State

This is a concrete internal attribute setting of a component. A physical state represents a distinct value setting of an abstract component state that does not define values but only value domains.

Postcondition

This refers to the collection of all constraints on component properties of the server that must be fulfilled after successful completion of an operation invocation on a component. The postcondition is part of a server's component contract.

Precondition

This refers to the collection of all constraints on component properties of the client that must be fulfilled before an operation may be called on a server. The precondition is part of the client's component contract.

Product Family

This refers to all possible similar systems that are based on a common product family core, the component framework.

Product Line

This is the same as a product family.

QoS Contract

A quality-of-service contract quantifies the expected behavior or the component interaction in terms of minimum and maximum response delays, average response, quality of a result, e.g., in terms of precision, result throughput in data streams, and the like.

Quality Assurance Plan

This is a collection of documents defining what quality means for a development project, how it manifests itself in different kinds of products, what quality aspects are important for different kinds of products, and which quality levels are required and how they can be reached.

Refinement

Refinement is the representation of an entity in the same notation at a finer level of detail.

Semantic Map

This describes a mapping between the specification of a desired component and the KobrA specification of the interface offered by a foreign component.

Server

A server of a component instance *CI* is any other component instance whose operations *CI* invokes. A server of a component *C* (as a type) is any other component with instances that are servers of *C's* instances.

Structural Model

This describes classes and the nature of their attributes, operations, and relations in the form of UML diagrams.

Spiral Model

The spiral model represents an iterative approach to organize the phases of the software devlopment life-cycle. Each iteration in the process goes through planning, determining goals, alternatives, and constraints, evaluating alternatives and risks, developing and testing.

Synchronization Contract

The synchronization contract adds another dimension to the behavioral contract, which is the sequence or combination with which the interdependent operations of a component may be invoked.

Test

This term refers to an experiment under controlled conditions that applies a set of test cases or a test framework in order to validate wether a tested entity is consistent with its specification.

Test Case

This refers to an experimental execution of the component under consideration; a test case comprises an operation with parameters, expected and actual pre- and postconditions, a result, and a verdict.

Test Modeling

This refers to all activities and techniques that deal with the specification of test artifacts using models.

Test Stub

This represents fake functionality at a component's required interface with which it can be executed and assessed in a test run.

Testable Component

This refers to a component that provides an additional access mechanism to alleviate testing in built-in contract testing.

Tester Component

This refers to a component that contains test cases in built-in contract testing.

Testing Component

This refers to a component that owns a built-in tester component in built-in contract testing.

Testing Interface

This is an additional access mechanism of a component that facilitates its testing.

Translation

Translation is a transformation from one representation format into another one on the same level of detail.

UML Testing Profile

This is a variant of UML that addresses specifically the requirements of building test systems with UML.

Usage Model

This is a model that describes how a system is used.

Validation

Validation refers to all activities that are applied to assess whether "we build the right system." Typically, this comprises testing technologies that are applied in a translation relation.

Verification

Verification refers to all activities that are applied to assess whether "we build the system right." Typically, this comprises inspection and review techniques that are applied in a transformation relation.

V-Model

The v-model is one of the earliest product models that shows how the artifacts in a software development are related to one another. It aligns with the waterfall model as process model.

Waterfall Model

Software development process that proceeds linearly from requirements analysis through design, coding, and unit testing, subsystem testing and system testing. The waterfall model as process model aligns with the v-model as product model.

Worst-Case Execution Time

This is the longest feasible runtime of a program or a component over all feasible runs according to varying input parameters.

References

1. A. Abdurazik and J. Offutt. Using UML collaboration diagrams for static checking and test generation. In *International Conference on the Unified Modeling Language (UML 2000)*, York, UK, October 2000.
2. P. Allen and F. Frost. *Component-Based Development for Enterprise Systems: Applying the Select Perspective*. Cambridge University Press, 1998.
3. S. Amiri, C. Bunse, H.G. Gross, N. Mayer, and C. Peper. Marmot – Method for object-oriented and component-based embedded real-time system development and testing. http://www.marmot-project.org.
4. H. Apperly. The Component Industry Metaphor. In *Component-Based Software Engineering, Heineman/Councill (Eds)*, Boston, 2001. Addison-Wesley.
5. C. Atkinson, C. Bunse, H.-G. Gross, and T. Kühne. Towards a general component model for Web-based applications. *Annals of Software Engineering*, 13, 2002.
6. C. Atkinson et al. *Component-Based Product-Line Engineering with UML*. Addison-Wesley, London, 2002.
7. F. Barbier, N. Belloir, and J.-M. Bruel. Incorporation of test functionality into software components. In *2nd International Conference on COTS-Based Software Systems*, Volume LNCS 2580, Ottawa, Canada, Feb. 2003. Springer.
8. J. Bayer et al. Pulse – A methodology to develop software product lines. In *Proceedings of the 5th Symposium on Software Reusability (SSR'99)*, Los Angeles, May 21–23, 1999.
9. K. Beck. *Extreme Programming Explained*. Addison-Wesley, 1999.
10. K. Beck and E. Gamma. *Test Infected – Programmers Love Writing Tests*. CSLife and OTI, Zürich (http://members.pingnet.ch/gamma/junit.htm).
11. B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, 1990.
12. B. Beizer. *Black-Box Testing, Techniques for Functional Testing of Software and Systems*. Wiley, New York, 1995.
13. N. Belloir, J.-M. Bruel, and F. Barbier. BIT/J User's Guide. Technical Report, University of Pau, LIUPPA, http://liuppa.univ-pau.fr/themes/aoc/aoc/bitj.php, 2003.
14. A. Bertolino and P. Inverardi. Architecture-based software testing. In *SIGSOFT'96 Workshop on Software Architectures*, San Francisco, CA, USA, 1996.

15. A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Software*, 32(7):38–44, 1999.
16. R. Binder. *Testing Object-Oriented Systems: Models, Patterns and Tools*. Addison-Wesley, 2000.
17. J. Bloch. *Effective Java Programming Language Guide*. Sun Microsystems, 2001.
18. J. Bøegh. Quality evaluation of software products. *Software Quality Professional*, 1(2), 1999.
19. L. Bokhorst (Ed.). Requirements Specification Description Template. Technical Report, DESS Project (Software Development Process for Real-Time Embedded Software Systems), November 2001.
20. G. Booch. *Software Components with Ada: Structures, Tools and Subsystems*. Benjamin-Cummings, Redwood, CA, 1987.
21. M. Born, A. Hoffmann, A. Rennoch, and J. Reznik. The European CORBA Components open source initiative. *European Research Consortium for Informatics and Mathematics – News*, 55:33–34, October 2003.
22. J. Bosch (Ed.). *Generative and Component-Based Software Engineering*, Volume 2186 of *Lecture Notes in Computer Science*. Springer, Berlin, 2001.
23. F. Brooks. *The Mythical Man Month*. Addison-Wesley, 1995.
24. A.W. Brown. *Large-Scale, Component-Based Development*. Prentice Hall, 2000.
25. C. Bunse. *Pattern-Based Refinement and Translation of Object-Oriented Models to Code*, Volume 2 of *PhD Theses in Experimental Software Engineering*. Fraunhofer, Stuttgart, 2001.
26. C. Bunse and C. Atkinson. Improving quality in object-oriented software: Systematic refinement and translation from models to code. In *12^{th} International Conference on Software & Systems Engineering and Their Applications*, Paris, France, 1999.
27. C. Bunse and C. Atkinson. The normal object form: Bridging the gap from models to code. In *2^{nd} International Conference on the Unified Modeling Language*, Fort Collins, USA, 1999.
28. J. Cheesman and J. Daniels. *UML Components, A Simple Process for Specifying Component-Based Systems*. Addison-Wesley, 2000.
29. S. Chung et al. Testing of concurrent programms based on message sequence charts. In *IEEE International Symposium on Software Engineering for Parallel and Distributed Systems*, Los Angeles, CA, May 17–18, 1999.
30. J. Clark, C. Clarke, S. DePanfilis, G. Granatella, P. Predonzani, A. Sillitti, G. Succi, and T. Vernazza. Selecting components in large COTS repositories. *Journal of Systems and Software*, 2004.
31. A. Cockburn. Basic Use Case Template. Technical Report TR.96.03a, Human and Technology (http://alistair.cockburn.us), Salt Lake City, 1996.
32. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, Boston, 2001.
33. D. Coleman et al. *Object-Oriented Development. The Fusion Method*. Prentice Hall, 1994.
34. Component+ Consortium, www.component-plus.org. *Built-in Test Vade Mecum – Part 1: A common BIT architecture*, 2003.
35. Component+ Consortium, www.component-plus.org. *Built-in Test Vade Mecum – Part 2: Interface specifications, types, syntax and semantics*, 2003.
36. Component+ Consortium, www.component-plus.org. *Built-in Test Vade Mecum – Part 3: Quality of service testing*, 2003.

37. Component+ Consortium. Built-in Testing for Component-Based Development. Technical Report, Component+ Project, www.component-plus.org, 2001.
38. Component+ Consortium. Built-in Testing for Component-Based Development. Technical Report, Deliverable D3, www.component-plus.org, 2001.
39. J.R. Corbin. *The Art of Distributed Applications: Programming Techniques for Remote Procedure Calls*. Springer, 1991.
40. K. Czarnecki and U.W. Eisenecker. *Generative Programming*. Addison-Wesley, 2000.
41. W. Dröschel and M. Wiemers. *Das V-Model 1997*. Oldenbourg, 1999.
42. D.F. D'Souza and A.C. Willis. *Objects, Components and Frameworks*. Addison-Wesley, 1998.
43. A. Eberhart and S. Fischer. *Web Services*. Hanser, München, 2003.
44. H.-E. Eriksson and M. Penker. *UML Toolkit*. Wiley, 1998.
45. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 1: TTCN-3 Core Language. Technical Report, ETSI ES 201 873-1, 2003.
46. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 2: TTCN-3 Tabular Presentation Format. Technical Report, ETSI ES 201 873-2, 2003.
47. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 3: TTCN-3 Graphical Presentation Format. Technical Report, ETSI ES 201 873-3, 2003.
48. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 4: TTCN-3 Operational Semantics. Technical Report, ETSI ES 201 873-4, 2003.
49. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 5: TTCN-3 Runtime Interface. Technical Report, ETSI ES 201 873-5, 2003.
50. European Telecommunications Standards Institute (ETSI). The Testing and Test Control Notation – Part 6: TTCN-3 Control Interface. Technical Report, ETSI ES 201 873-6, 2003.
51. Cpp Test Framework for C++. http://cppunit.sourceforge.net.
52. Perl Unit Test Framework for C++. http://perlunit.sourceforge.net.
53. International Organization for Standardization (ISO). Information Technology - Open System Interconnection, Conformance Testing Methodology and Framework. Technical Report, ISO/IEC 9646:1998, 1998.
54. International Organization for Standardization (ISO). Information Technology - Open Distributed Processing - Interface Definition Language. Technical Report, ISO/IEC 14750:1999, 1999.
55. International Organization for Standardization (ISO). Software Engieering – Product Evaluation – Part 3: Process for Developers. Technical Report, ISO/IEC 14598-3:2000, 2000.
56. International Organization for Standardization (ISO). Software Engieering – Product Quality – Part 1: Quality Model. Technical Report, ISO/IEC 9126-1:2001, 2001.
57. International Organization for Standardization (ISO). Software Engieering – Product Quality – Part 2: External Metrics. Technical Report, ISO/IEC TR 9126-2:2003, 2003.

58. International Organization for Standardization (ISO). Software Engieering – Product Quality – Part 2: Internal Metrics. Technical Report, ISO/IEC TR 9126-3:2003, 2003.
59. M. Fowler. A UML Testing Framework. *Software Development*, April 1999.
60. M. Fowler and K. Scott. *UML Distilled*. Addison-Wesley, 1997.
61. C Unit Test Framework. http://cunit.sourceforge.net.
62. JUnit Test Framework. http://www.junit.org.
63. XUnit Test Framework. http://c2.com/cgi/wiki?testingframework.
64. S. Frolund and J. Koisten. QML: A language for quality of service specification. Technical Report HPL-98-10 980210, Hewlett-Packard, 1998.
65. J. Gao. Challenges and problems in testing software components. In *Workshop on Component-Based Software Engineering (ICSE 2000)*, Limerick, June 2000.
66. J.Z. Gao, H.-S.J. Tsao, and Y. Wu. *Testing and Quality Assurance for Component-Based Software*. Artech House, 2003.
67. S. Ghosh and A.P. Mathur. Issues in testing distributed component-based systems. In *Workshop on Testing Distributed Component-Based Systems (ICSE 1999)*, Los Angeles, May 1999.
68. D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
69. J. Grabowski and M. Schmitt. TTCN-3 - A language for the specification and implementation of test cases. *'at – Automatisierungstechnik*, 3, 2002.
70. L. Graham, B. Henderson-Sellers, and H. Younessi. *The OPEN Process Specification*. Addison-Wesley, 1997.
71. H.-G. Gross. *Measuring Evolutionary Testability of Real-Time Software*. PhD thesis, University of Glamorgan, Pontypridd, Wales, UK, June 2000.
72. H.-G. Gross. An evaluation of dynamic, optimisation-based worst-case execution time analysis. In *International Conference on Information Technology*, Kathmandu, Nepal, 2003.
73. H.-G. Gross, C. Atkinson, F. Barbier, N. Belloir, and J.-M. Bruel. Business Component-Based Software Engineering, Barbier (Ed.), Chapter Built-in Contract Testing for Component-Based Development (Chapter. 4). Kluwer, 2003.
74. H.-G. Gross, C. Atkinson, and F. Barbier. *Component-Based Software Quality, Cechich, Piattini, Vallecillo (eds.)*, Volume 2693 of *Lecture Notes in Computer Science (LNCS)*, chapter Component Integration through Built-in Contract Testing. Springer, Berlin, 2003.
75. H.-G. Gross and N. Mayer. Search-based execution-time verification in object-oriented and component-based real-time system development. In 8^{th} *IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, Guadalajara, Mexico, January, 15–17 2003.
76. H.-G. Gross, I. Schieferdecker, and G. Din. *Quality in Component-based Development – Testing and Debugging, Beydeda (Ed.)*, Chapter Modeling and Implementation of Built-in Contract Tests. Springer, Berlin, 2004.
77. M. Grossman. Component testing: An extended abstract. In *Workshop on Component-Oriented Programming (ECOOP 1998)*, Brussels, July 1998.
78. Object Management Group. History of CORBA. Technical Report, http://www.omg.org, 1997–2004.
79. Object Management Group. Model driven architecture - resource page. Technical Report, http://www.omg.org, 1997–2004.

80. Object Management Group. Object management architecture - resource page. Technical Report, http://www.omg.org, 1997–2004.
81. Object Management Group. UML testing profile. Technical Report, http://www.omg.org, 1997-2004.
82. Object Management Group. CORBA - core specification. Technical Report, Version 3.0, December 2002.
83. D.S. Guindi, W.B. Ligon, W.M. McCracken, and S. Rugaber. The impact of verification and validation of reusable components on software productivity. In *22nd Annual Hawaii Intl Conference on System Sciences*, Pages 1016–1024, 1989.
84. D. Hamlet, D. Mason, and D. Woit. Theory of software reliability based on components. In *23rd International Conference on Software Engineering (ICSE-01)*, Los Alamitos, California, 2001. IEEE Computer Society.
85. D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
86. M.J. Harrold, D. Liang, and S. Sinha. An approach to analyzing and testing component-based systems. In *Workshop on Testing Distributed Component-Based Systems (ICSE 1999)*, Los Angeles, May 1999.
87. J. Hartmann, C. Imoberdorf, and M. Meisinger. UML-based integration testing. In *International Symposium on Software Testing and Analysis (ISSTA 2000)*, Portland, USA, August 2000.
88. R. Heckel and M. Lohmann. Towards model-driven testing. *Electronic Notes in Theoretical Computer Science*, 82(6), 2003.
89. G.T. Heineman and W.T. Councill (Eds). *Component-Based Software Engineering*. Addison-Wesley, Boston, 2001.
90. J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, 1975.
91. IEEE. *Standard Glossary of Software Engineering Terminology*, Volume IEEE Std. 610.12-1990. IEEE, 1999.
92. Fraunhofer IGD. RIN system specification. Technical Report, Fraunhofer Institute for Graphical Data Processing, Darmstadt, Germany (www.igd.fraunhofer.de), 2002.
93. J.A. Illik. *Programmierung in C unter UNIX*. Sybex, Düsseldorf, 1990.
94. European Telecommunications Standard Institute. www.etsi.org.
95. I. Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.
96. I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
97. J.M. Jézéquel and B. Meyer. Design by contract: The Lessons of ARIANE. *IEEE Computer*, 30(1):129–130, Jan. 1997.
98. J.-M. Jézéquel, D. Deveaux, and Y. Le Traon. Reliable objects: Lightweight testing for oo languages. *IEEE Software*, July/August 2001.
99. K.C. Kang, S.G. Cohen, W.E Novak, and E.S. Petersen. Feature-oriented domain analysis (FODA) feasibility study. Technical Report, Software Engineering Institute, November 1990.
100. Y.G. Kim, H.S. Hong, D.H. Bae, and S.D. Cha. Test cases generation from UML state diagrams. *IEE Proceedings Software*, 146(4), 1999.
101. P.B. Kruchten. *The Rational Unified Process – An Introduction*. Addison-Wesley, 2000.
102. D. Lane. *JUnit – The Definitive Guide*. O'Reilly, 2004.

103. Y. Le Traon, D. Deveaux, and J.-M. Jézéquel. Self-testable components: from pragmatic tests to design for testability methodology. In *Technology of Object-Oriented Languages and Systems*, Nancy, France, June 7-11 1999.
104. J.T.L. Lions and ARIANE 501 Inquiry Board. ARIANE 5, flight 501 failure. Technical Report, European Space Agency, Paris, July 1996.
105. Testing Technologies IST Ltd. *TTanalyze, http://www.testingtech.de/products/TTanalyze.*
106. Testing Technologies IST Ltd. *TTspec, http://www.testingtech.de/products/TTspec.*
107. Testing Technologies IST Ltd. *TTthree, http://www.testingtech.de/products/TTthree.*
108. Testing Technologies IST Ltd. *TTtwo2three, http://www.testingtech.de/products/TTtwo2three.*
109. B. Marick. *The Craft of Software Testing.* Englewood-Cliffs, New Jersey, 1995.
110. R.C. Martin. Java vs. C++. Technical Report, Object Mentor, Inc., www.objectmentor.com, March 1997.
111. J.D. McGregor. Testing a software product line. Technical Report CMU/SEI-2001-TR-022, Software Engineering Institute, 2001.
112. B. Meyer. *Object-oriented Software Construction.* Prentice Hall, 1997.
113. Microsoft. *Microsoft .NET.* http://www.microsoft.com/net.
114. Sun Microsystems. Enterprise JavaBeans technology specification, version 2.1 – final release. Technical Report, java.sun.com, 1995–2003.
115. Sun Microsystems. JavaBeans component architecture documentation. Technical Report, java.sun.com, 1995–2003.
116. H.D. Mills, R.C. Linger, and R.A. Hevner. Box structured information systems. *IBM Systems Journal*, 26(4), 1987.
117. K.D. Nilsen and B. Rygg. Worst-case execution time analysis on modern processors. *ACM SIGPLAN Notices*, 30(11):61–64, 1995.
118. Object Management Group. *Unified Modeling Language Specification*, 2000.
119. J. Offutt and A. Abdurazik. Generating tests from UML specifications. In *International Conference on the Unified Modeling Language (UML 1999)*, Fort Collins, USA, October 1999.
120. OMG. *UML 2.0 Testing Profile Specification.* Object Management Group, www.omg.org, 2003.
121. OpenTTCN. *http://www.openttcn.com.*
122. C.Y. Park. Predicting program execution times by analyzing static and dynamic program paths. *Real-Time Systems*, 5:31–62, 1993.
123. P. Predonzani, G. Succi, and T. Vernazza. *Strategic Software Production with Domain-Oriented Reuse.* Artech House, 2000.
124. R.S. Pressman. *Software Engineering: A Practioner's Approach.* McGraw-Hill, New York, 1997.
125. CLARiFi Project. http://www.clarify.eng.it.
126. EU ITEA Empress Project. http://www.empress-itea.org.
127. P. Puschner. A tool for high-level language analysis of worst-case execution times. In *10^{th} Euromicro Workshop on Real-Time Systems*, Berlin, 1998.
128. P. Puschner. Worst-case execution-time analysis at low cost. In *Control Engineering Practice*, Volume 6, Pages 129–135, 1998.
129. P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. In *Real-Time Systems*, Volume 1, Pages 159–176, 1989.

130. P. Puschner and R. Nossal. Testing the results of static worst-case execution time analysis. In *19th IEEE Real-Time Systems Symposium*, Madrid, Dec. 1998.
131. P. Puschner and A. Schedl. Computing maximum task execution times – a graph-based approach. In *Real-Time Systems*, Volume 13, Pages 67–91, 1997.
132. T. Reenskaug, P. Wold, and O. Lehne. *Working with Objects: The OORAM Software Development Method*. Manning/Prentice Hall, 1996.
133. B. Regnell and P. Runeson. Combining scenario-based requirements with static verification and dynamic testing. In *4th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ98)*, Pisa, June 8–9, 1998.
134. B. Regnell, P. Runeson, and C. Wohlin. Towards the integration of use case modeling and usage-based testing. In *4th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ98)*, Pisa, June 8-9 1998.
135. R.H. Reussner. *Parametrisierte Verträge zur Protokolladaption bei Software-Komponenten*. Logos, Berlin, 2001.
136. R.H. Reussner. The use of parameterised contracts for architecting systems with software components. In *6th Intl Workshop on Component-Oriented Programming*, Budapest, Hungary, 2001.
137. R.H. Reussner and H.W. Schmidt. Using parameterised contracts to predict properties of component based software architectures. In *9th International Workshop on Component-Based Software Engineering*, Lund, Sweden, 2002.
138. R.H. Reussner, H.W. Schmidt, and I.H. Poernomo. Reliability prediction for component-based software architectures. *Systems and Software*, 66(3), 2002.
139. D.J. Richardson and A.L. Wolf. Software testing at the architectural level. In *SIGSOFT 1996 Workshop on Software Architectures*, San Francisco, CA, USA, 1996.
140. P.J. Robinson. *Hierarchical Object-Oriented Design*. Prentice Hall, 1992.
141. D.S. Rosenblum. A practical approach to programming with assertions. *IEEE Transactions on Software Engineering*, 21(1):19–31, Jan. 1995.
142. D.S. Rosenblum. Adequate testing of component-based software. Technical Report, Dept. of Computer Science, University of California, TR 97-34, 1997.
143. J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.
144. J. Ryser and M. Glinz. A practical approach to validating and testing software systems using scenarios. In *Quality Week Europe*, Brussels, 1999.
145. J. Ryser and M. Glinz. SCENT: A method employing scenarios to systematically deriving test cases for system test. Technical Report, University of Zürich, 2000.
146. J. Ryser and M. Glinz. Using dependency charts to improve scenario-based testing. In *17th International Conference on Testing Computer Software (TCS2000)*, Washington, 2000.
147. P. Santos, T. Ritter, and M. Born. Rapid engineering of collaborative and adaptive multimedia systems on top of CORBA Components, K. Irmscher (Ed.). *Kommunikation in Verteilten Systemen, VDE*, 2003.
148. I. Schieferdecker, Z.R. Dai, J. Grabowski, and A. Rennoch. The UML 2.0 testing profile and its relation to TTCN-3. In Wiles Hogrefe (Ed.), *Proceedings of the 15th International Conference on Testing Communicating Systems*, Springer LNCS Volume 2644, Heidelberg, 2003.

149. I. Schieferdecker and J. Grabowski. The graphical format of TTCN-3 in the context of MSC and UML. In *International Workshop on SDL and MSC*, Springer LNCS Volume 2599, Heidelberg, 2003.
150. M. Schünemann, I. Schieferdecker, A. Rennoch, L. Mang, and C. Desroches. Improving test software using TTCN-3. Technical Report, GMD Forschungszentrum Informationstechnik (now Fraunhofer FOKUS), 2001.
151. H.P. Schwefel and R. Männer. *Parallel Problem Solving from Nature*. Springer, 1990.
152. B. Selic, G. Gullekson, and P. Ward. *Real-Time Object-Oriented Modeling*. Wiley, 1994.
153. A. Sillitti, G. Granatella, P. Predonzani, G. Succi, and T. Vernazza. Ranking and selecting components to build systems. In *International Conference on Enterprise Information Systems (ICEIS 2003)*, Angers, France, 2003.
154. IBM Rational Software. Purifyplus; http://www-306.ibm.com/software/awdtools/purifyplus.
155. G. Succi, W. Pedrycz, and R. Wong. Dynamic composition of components using Web-CODs. *International Journal of Computers and Applications*, 2002.
156. SunSoft. *Java 2 Enterprise Edition (J2EE)*. http://java.sun.com/j2ee/.
157. C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, Harlow, England, 1999.
158. C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, London, England, second edition, 2002.
159. J. Udell. Componentware. *Byte Magazine*, 14(5):46–56, May 1994.
160. A. van der Hoek and H. Muccini. Towards testing product line architectures. In *ETAPS 2003 – Workshop on Test and Analysis of Component-Based Systems*, 2003.
161. J. Vincent and G. King. Built-in-test for run-time-testability in software components: Testing architecture. In *BCS Software Quality Management Conference*, 2002.
162. J. Vincent, G. King, P. Lay, and J. Kinghorn. Principles of built-in-test for run-time-testability in component based software systems. *Software Quality Journal*, 10(2), 2002.
163. J. Vincent, G. King, P. Lay, and J. Kinghorn. *Business Component-based Software Engineering, Franck Barbier (Ed.)*, chapter Built-In-Test for Run-Time-Testability in Software Components: Testing Architecture. Kluwer, Boston, 2003.
164. P.J.M. Von Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer, 1997.
165. Y. Wang, G. King, I. Court, M. Ross, and G. Staples. On testable object-oriented programming. *ACM Software Engineering Notes*, 22(4), 1997.
166. Y. Wang, G. King, D. Patel, S. Patel, and A. Dorling. On coping with real-time software dynamic inconsistency by built-in tests. *Annals of Software Engineering*, 7, 1999.
167. Y. Wang, G. King, and H. Wickburg. A method for built-in tests in component-based software maintenance. In *IEEE International Conference on Software Maintenance and Reengineering (CSMR-99)*, Pages 186–189, 1999.
168. Y. Wang, D. Patel, G. King, and S. Patel. *BIT: A Method for Built-in Tests in Object-Oriented Programming*, chapter 47 in Handbook of Object Technology, Zamir (Ed.). CRC Press, 1998.

169. C.D. Warner. Evaluation of program testing. Technical Report, IBM Data Systems Division Development Laboratories, Poughkeepsie, NY, July 1964.
170. B.F. Webster. *Pitfalls of Object-Oriented Development.* M&T Books, 1995.
171. J. Wegener and M. Grochtmann. Verifying timing constraints by means of evolutionary testing. *Real-Time Systems*, 3(15), 1998.
172. J. Wegener, R. Pitschinetz, and H. Sthamer. Automated testing of real-time tasks. In *1st International Workshop on Automated Program Analysis, Testing and Verification*, Limerick, Ireland, June 2000.
173. J. Wegener, H. Sthamer, and A. Baresel. Application fields for evolutionary testing. In *EUROStar, Testing and Verification*, Stockholm, Sweden, November 2001.
174. D.M. Weiss and C.T.R. Lai. *Software Product Line Engineering – A Family-Based Software Engineering Process.* Addison-Wesley, 1999.
175. E.J. Weyuker. Testing component-based software: A cautionary tale. *IEEE Software*, 15(5), 1998.
176. E.J. Weyuker. The trouble with testing components,. In *Component-Based Software Engineering, Heineman/Councill (Eds).* Addison-Wesley, 2001.

Index

.NET, 203

abstract state
  specification, 267
abstract test case, 136
abstract test software, 148
abstraction dimension, 187
ACET, 269
acquires
  stereotype, 146
ActiveX, 203, 204
activity diagram, 102
  concept, 104
  testing, 104
Activity Model, 35
activity model, 106
adapter component, 56, 59, 175
algorithmic model, 44, 48
application
  non-testable version, 138
  testable version, 138
application engineering, 9, 62, 184
architectural reuse, 242
Ariane 5 failure, 14, 121
assertion, 123, 144
  object invariant, 123
  parts, 124
  postcondition, 123
  precondition, 123
assertion combination, 145
average case execution time, 269

banking application, 211
basic contract, 256
BCET, 269, 272
behavioral contract, 256
behavioral model, 30, 42, 135
  testing, 99
behavioral modeling, 98, 102
best-case execution time, 269
best-case execution-time
  analysis, 272
BIT/J Library, 199
BIT/J Library structure, 199
black box coverage, 77
black box testing, 77
bottom-up composition, 229
bottom-up development, 21
bottom-up integration, 176
built-in
  documentation, 234
  platform test, 233
  quality of service testing, 280
  self test, 125
  testing artifacts, 231
built-in contract testing, 122
  architecture, 130, 164
  C, 188
  component technologies, 206
  C++, 191
  development, 185
  documentation, 165
  embodiment, 183
  implementation, 179, 185
  implementation technologies, 214
  integration, 174
  Java, 191

JUnit, 216
motivation, 127
objective, 123, 127
permanent artifacts, 163
process, 123, 157
product line engineering, 183
programming languages, 187
removable artifacts, 163
return on investment, 127
step-by-step guide, 159
support framework, 199
test architecture, 135
tested interactions, 163
tester component, 123
testing interface, 123
TTCN-3, 223
web services, 209, 210
built-in testing, 123, 124
integration testing, 144
permanent checking, 144

Catalysis, 23
category partitioning, 78
CCM, 204, 205
certification, 238
CLARiFi, 238
broker platform, 239
certification model, 239
class association, 92
multiplicity, 93
class diagram
concepts, 92
testing, 96
Cleanroom, 23
client, 129
client/server model, 129
clientship, 5
cloning, 212
cohesion, 43
collaboration diagram
concepts, 109
COM, 203
component, 2
acceptance testing, 122
acquisition, 122
adapter, 13, 56, 59, 175
allocation, 207
assembly, 121
association, 152
binding, 181
certification, 238
clientship, 5
cloning, 212
cohesion, 43
composability, 3
composition, 4
configuration, 146, 154
containment, 4
context, 146
contract, 5, 92, 123, 129, 256
customer test, 233
data cohesion, 43
deallocation, 207
definition, 2
degradation, 126
deployment environment, 131
different usage patterns, 233
documentation, 69
engineering, 123
expectation, 127
expected environment, 126
final state, 134
framework, 62, 67, 184
functional cohesion, 43
glue code, 128
identification, 122
implementation, 7
independent deployment, 2
initial state, 134
instance, 6
integration, 58, 60, 179
interaction, 3, 129
interaction model, 53
interface, 5
internal state, 134
invariant, 145
meta-model, 7
modularity, 3
nesting, 4, 28, 91, 129
physical component, 9
platform, 181
postcondition, 8
precondition, 8
procurement, 238
properties, 3
provided interface, 5, 6
provider, 13
provider test, 232

quality assurance plan, 69
quality attributes, 6
realization, 9, 25, 37, 44, 47, 53, 147
required interface, 5, 6
reusability, 3
reuse, 56, 229
runtime environment, 128
runtime reuse, 235
self test, 126
specification, 9, 25, 37
stakeholder, 13
state, 6
state change, 135
stereotype, 29
usage profile, 130
user, 13
variant, 183
wiring standards, 200
wrapper, 56, 59
component diagram
concepts, 93
testing, 95
component engineering, 9
component hook, 244
component integration, 174
adapter, 175
Component Interface, 5
component market, 121
component meta model
extended, 157
component specification, 38, 40
component technology, 200
component-based development, 2, 21
bottom-up development, 21
challenges, 121
clientship, 5
composition, 4
dimensions, 24
expectation, 122
principles, 4, 22
savings, 122
top-down development, 21
vision, 121
component-based software testing, 11
composability, 3
composition, 4, 10, 27
bottom-up, 177
top-down, 177
composition dimension, 187
COM+, 203
concretization dimension, 179, 187
configuration, 154
configuration mechanism, 231
conformance map, 59
constructive timing analysis, 260
constructor, 280
containment, 129
hierarchy, 57
rules, 27
tree, 4
containment diagram
concepts, 90
containment tree
testing, 98
context realization, 29, 31
contract, 5, 78, 129
containment, 129
ownership, 129
parameterized, 129
contract testing
exception, 146
control flow test, 104
CORBA, 204
CORBA component model, 205
CORBA Components, 205
core testing interface, 140
COTS component, 58
integration, 58
coverage
state-transition, 151
customer self-certification, 240
customer test, 233

data cohesion, 43
DCOM, 203
decision
resolution model, 68
decision model, 62
decision modeling, 67
decomposition, 10, 24, 180
decomposition dimension, 187
dependency, 92
deployment, 60
deployment diagram
concepts, 94
testing, 96
deployment environment, 121, 131
development

for reuse, 3
with reuse, 3
development environment, 121, 255
development for reuse, 214
development with reuse, 214
development-time
component reuse, 232
environment, 258
evolution, 235
different usage patterns, 233
documentation, 234
domain analysis, 78
domain knowledge, 121
duality of components, 76
dynamic tester component, 138
dynamic timing analysis, 274
dynamic update, 279
problems, 280

embodiment, 25, 26, 50, 68, 179, 180, 183, 186, 229
C, 188
C++, 191
Java, 191
encapsulation, 3, 42, 124, 134
Enterprise JavaBeans, 201
enterprise model, 30
environment
expectation, 127
equivalence partitioning, 77, 78
error classification, 74
evolution strategies, 272
evolutionary testing, 272
execution environment, 122
execution time analysis, 259
explicit client/server relationship, 165
explicit client/server-relationship, 166
explicit invariant, 145
Explicit Server, 130
explicit server, 131, 133
export interface, 7
extended model
tester component, 268
testing interface, 267
extensible markup language, 209
externally visible state, 135
extreme programming, 215

FAST, 23
final state, 41, 43, 134
fitness function, 272
flow graph coverage, 104
FODA, 23
framework, 64
engineering, 64, 67, 68
framework engineering, 62, 184
functional
cohesion, 43
component, 42
model, 41, 184
testing, 78
Fusion, 23

generalization, 92
generic framework, 67
genetic algorithms, 272
glue code, 179
good tester component, 148
good testing interface, 140

handler, 280
hard real-time, 264
heavy weight test, 149
high-level tester components, 84
HOOD, 23

IDL, 204
implementation, 26, 179
implementation technologies, 179
implicit client/server relationship, 165
implicit client/server-relationship, 166
implicit design decisions, 52
implicit invariant, 145
implicit runtime system, 259
Implicit Server, 130
implicit server, 131, 133
import interface, 7
independently deployable, 3
information hiding, 3, 124, 134
inheritance, 92
initial state, 41, 43, 134
instantiation, 184, 186
integration test, 75
interaction diagram
concepts, 107
testing, 109
Interaction Model, 35
interaction model, 44, 48, 52, 53
interaction modeling, 106

interaction test, 105
interface, 5
interface definition language, 204
internal state, 100, 134
invariant, 145
invocation
  history, 143, 144, 271
  interface, 204

Java
  beans, 201
  component, 201
  ORB, 207
  self test, 125
JavaBeans, 201
JUnit, 215

KobrA Method, 22
KobrA method, 22, 23
  abstraction, 24
  abstraction dimension, 187
  activity model, 35, 36
  algorithmic model, 44, 48
  behavioral model, 30, 38, 42
  component realization, 37, 44, 47
  component specification, 37, 38, 40
  composition, 24, 27
  composition dimension, 187
  concretization, 24
  concretization dimension, 187
  conformance map, 59
  containment rules, 27
  context realization, 29, 30
  decision model, 62
  decision modeling, 67
  decomposition, 24
  decomposition dimension, 187
  deployment, 60
  dimensions, 24, 180
  embodiment, 24–26, 50, 68
  encapsulation, 23
  enterprise model, 30, 33
  framework engineering, 64, 68
  functional model, 38, 41
  genericity, 24
  incremental development, 23
  information hiding, 23
  interaction model, 35, 37, 44, 48
  model-based development, 23
  modularity, 23
  operation specification template, 41, 42, 87
  principles, 22
  process model, 33
  reuse, 23
  semantic map, 59
  separation of concerns, 23
  specialization, 24
  state model, 42
  structural model, 30, 33, 38, 39, 44, 46
  system construction, 60
  top-level component, 37
  unified functions, 23
  usage model, 30
  use case definition, 31
  use case template, 32, 34, 89
  validation, 27

late integration, 121
light weight test, 149
live update, 279, 280
  test of, 235
logical component, 60
logical state, 100, 135
low testability, 134

mapping
  semantic, 127
  syntactic, 127
Marmot method, 30
message flow coverage, 105
message sequence-based testing, 79
method sequence-based testing, 79
middleware, 181
middleware platform, 200
  organization, 202
model-based testing, 74, 80
model-driven
  architecture, 23, 56
  development, 21
modularity, 3
mutation, 272

n-transition coverage, 100
natural evolution, 272
nesting
  tree, 4
node coverage, 76

NOF model, 56
non-functional properties, 255
non-functional requirements, 255
normal object form, 55

object diagram
  concepts, 92
  testing, 96
object-oriented principles, 3
  encapsulation, 3
  modularity, 3
  unique identity, 3
observability, 152
OLE, 204
OMA, 204
OMG IDL, 204
OMT, 23
OORAM, 23
OPEN, 23
operation specification
  template, 41
  testing, 84
operation specification template, 42, 87
optimization-based timing analysis, 272
ORB platform, 200
output state, 134
ownership, 92, 129

package diagram
  concept, 93
  testing, 96
parameterized contract, 129
partition testing, 78
partitioning testing, 77
permanent testing artifacts, 163
persistent state, 6
physical component, 9, 60
physical state, 100, 135
piecewise coverage, 99
platform test, 233
population, 272
postcondition, 41
precondition, 41
predefined state, 138
procurement, 238
product family, 24, 61, 183
  framework, 184
  instantiation, 65
  test of, 242, 244
product line, 24, 63
  engineering, 62, 183
  test of, 242, 244
Provided Interface, 5
provided interface, 7, 25, 38, 41
provider test, 232
public stereotype, 7
PuLSE, 23

QML, 38, 260
QoS, 255
  contract, 256
  contract testing, 279
  requirements, 265
  testing interface, 267
QoS Modeling Language, 38, 260
quality assurance
  techniques, 70
quality assurance plan, 69
quality attributes, 6
  dependability, 6
  documentation, 6
  performance, 6
quality-of-service, 255
quality-of-service contract, 256
quantitative contract, 256

random testing, 272
Rational Unified Process, 23
real-time
  cohesion, 262
  requirement, 257
refinement, 53–55, 92, 181, 193
remote procedure call, 201
remote testing, 267
removable testing artifacts, 163
reproduction, 272
Required Interface, 5
required interface, 7, 25, 38
resolution model, 69
resource information network, 159
  behavioral model, 168, 170
  built-in testing, 167
  containment, 160
  context, 159
  test cases, 171
  testing interface, 168
  testing profile, 174
resource information system

class diagram, 162
response-time, 255
requirement, 257
specification, 265
responsibility-based testing, 78
reusability, 3
reuse, 229
architectural level, 62
architecture, 242
at development-time, 232
runtime, 235
RIN system
scenarios, 274
sequence diagram, 277
timing measurement, 278
ROOM, 23
round-trip path coverage, 100
runtime
evolution, 235
monitoring, 280
reuse, 235
support software, 133

search space, 273
exploitation, 273
exploration, 273
search-based timing analysis, 272
selection, 272
SelectPerspective, 23
self test, 125
advantage, 125
self-certification, 240
semantic map, 59
semantic mapping, 127, 174
separation of concerns, 23, 52
sequence diagram
concepts, 107
sequence model, 108
server, 129
explicit, 130
implicit, 130
server-client test, 235
set-to-state, 141
signal flow coverage, 105
simulated annealing, 272
soft real-time, 264
software crisis, 11
software development method, 22
software engineering, 1
cost dimension, 1
quality dimension, 1
software testing, 73
spiral model, 27
state
definition, 44
model, 42
transition, 9
state setup
criteria, 143
state transition, 131
state-based testing, 79, 99
criteria, 99
state-of-the-practice development, 21
state-transition
coverage, 151
statechart diagram, 98
concepts, 98
testing, 99
statement coverage, 76
stereotype
acquires, 146
komponent, 47
public, 7
subject, 46
testing, 185
variant, 29, 65, 185
stratification, 211
structural diagram
concepts, 90
testing, 95
structural model, 30, 44, 46, 113
structural modeling, 88
structural testing, 75, 77
subject stereotype, 46
synchronization contract, 256
syntactic mapping, 127, 174
system
construction, 60
design, 123
implementation, 179

test
adequacy criteria, 70
architecture, 135
as product line, 253
behavior, 172
defined, 124
invocation sequence, 142

method, 148
model, 86
modeling, 112
objective, 172
observation, 172
process, 74
server-client, 235
stub, 27
target definition, 95
two stages, 232
verdict, 172
weight, 149
test case, 172
behavior, 113
implementation, 269
ingredients, 265
pseudocode, 271
test of
code integrity, 283
data integrity, 283
residual defects, 283
Testability, 135
testability, 152
testable component, 131
tester component, 123, 268, 280
alternatives, 156
C, 190
criteria, 153
customization, 231
C++, 193
design, 146
heavy test, 149
invocation sequence, 176
Java, 193
lighter test, 149
optimal design, 148
realization, 169
specification, 169
test weight, 149
variation, 152
volume, 152
testing
architecture, 164
artifact, 74
component, 131, 169
concepts, 74
documentation, 165
model, 184
stereotype, 185
technique, 74
testing and test control notation, 219
testing criteria
black box, 77
white box, 75
testing interface, 123, 131, 134, 267, 280
alternatives, 136, 140
assert all, 145
C, 188
C++, 191
design, 141
IBITError, 281
IBITErrorNotify, 281
IBITQuery, 281
IBITRegister, 281
ingredients, 138
inheritance, 155
is-in-state, 144
Java, 191
nested components, 155
optimal design, 140
realization, 167
set-to-state, 141
specification, 167
state setup, 131
state validation, 131
usage, 140
testing profile, 112, 172, 224
arbiter, 117
behavior, 113
behavioral concepts, 118
concepts, 112, 113
mapping to UML, 115
structural concepts, 113
structure, 112
SUT, 117
test architecture, 115
test behavior, 172
test case, 117, 172
test component, 116
test configuration, 116
test context, 116
test objective, 118, 172
test observation, 172
UML structure, 115
validation action, 172
verdict, 172
throughput, 255
timing analysis, 260

approaches, 263
constructive, 260
dynamic, 263
problems, 261
static, 263
validation, 260
timing behavior, 258
timing contract test, 280
timing measurement, 267
top-down decomposition, 229
top-down development, 21
transition, 9
transition coverage, 100
translation, 53–55, 181, 193
TTCN-3, 219, 224

UML, 73
activity diagram, 35
activity model, 36
anchor symbol, 164
class diagram, 39
implementation, 55
interaction diagram, 35
interaction model, 37
meta-model, 7
normal object form, 55
object diagram, 39
profile, 55
structural model, 33, 39
testing profile, 112, 172, 215, 224
tools, 73
usage model, 67
use case model, 30, 31, 33
UML Components, 23
Unified Modeling Language, 7, 73
usage model, 30
usage modeling, 80
Usage Profile, 130
usage profile, 121, 130
use case definition, 31
use case diagram, 80
concepts, 80
coverage, 84
coverage criteria, 83
testing, 81
use case template, 32, 34, 85, 89
testing, 84
user requirement, 257

v-model, 27, 50
validation, 27, 73
validation action, 172
validation timing analysis, 260
variability identification, 67
variant stereotype, 29, 65, 185
variant system, 183
vending machine, 24
usage model, 33, 81
activity diagram, 106
activity model, 36
algorithmic model, 48, 50, 51
behavioral model, 45, 46, 101
cash component, 29
collaboration diagram, 110
containment, 195
containment diagram, 86
containment hierarchy, 28, 57
containment model testing, 96
containment testing, 96
containment tree, 91
context realization, 29, 30
cooling component, 29
decision model, 64
decomposition, 26
dispenser component, 26
display component, 29
embodiment, 189, 192
functional model, 41
initial state, 141
insert coins, 141
interaction model, 38, 48, 52, 53
key pad component, 29
operation specification, 43
realization, 147
refinement, 194
sequence diagram, 108
state model, 45
state table, 46
state transition table, 102
structural model, 35, 39, 41, 49, 94
structure, 94
testing containment, 86
testing interface, 144
testing model, 86
translation, 195
usage model, 67
use case, 34
use case diagram, 33, 81

use case model, 31, 33
verification, 73

waterfall model, 27, 50
WCET, 269, 272
web services, 209
white box coverage, 75
white box testing, 75
worst-case execution time, 269
worst-case execution-time analysis, 272
wrapper component, 59

XML, 209
XUnit framework, 215

Printing: Strauss GmbH, Mörlenbach
Binding: Schäffer, Grünstadt