Knowledge Area Description for Software Design (version 0.60)

Guy Tremblay
Dépt. d'informatique, UQAM
Montréal, Québec

1. Introduction

Software design, from (software) requirements typically stated in terms relevant to the problem domain, produces a description of a solution that will solve the software-related aspects of the problem. Software design describes how the system is decomposed and organized into components and describes the interfaces between these components (architectural design). Software design also refines the description of these components into a level of detail suitable for allowing their construction (detailed design).

In a classical software development life cycle, e.g., ISO/IEC 12207 [ISO95b], software design fits between software requirements analysis and software coding and testing (software construction). Software design encompasses both software architectural design (sometime called top-level design) and software detailed design. Software design plays an important role in the development of a software system in that it allows the developer to produce a model, a blueprint of the solution to be implemented. Such a model can then be analyzed and evaluated to determine if it will allow the various requirements to be fulfilled. This model can also be used to plan the subsequent development activities, in addition to being used as input and starting point of the coding and testing activities.

This document presents a breakdown of the Software Design Knowledge Area for the SWEBOK (stone man version). It has been developed in accordance with the “Knowledge Area Description Specifications for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge” (version 0.25). Thus, various constraints had to be satisfied by the resulting Knowledge Area (KA) description, e.g., it must describe “generally accepted” knowledge not specific to any application domains or development methods, it must propose a list of pointers to reference material with a limited number of entries, etc.

It must be noted that certain topics – e.g., User Interface Design, Real-time Design – were specifically excluded from the Software Design KA (SWEBOK Straw Man Version), thus are not explicitly discussed in the present document, although some of the topics in the KA might also apply to these specialized areas. Furthermore, some additional “Design” topics were also excluded from the present description, as they were considered to be outside of “Software Design” in the sense mentioned above; we come back to this issue in the Breakdown Rationale section.

2. Outline of the KA breakdown and rationale

In this section, we briefly go through the various requirements described in the “Knowledge Area Description Specifications for the Stone Man Version of the Guide to the SWEBOK” (version 0.25) and describe how most of these requirements are satisfied by the present KA description. Before doing so, however, let us first present the top two levels of the KA description.
### I. Software Design Basic Concepts
- General design concepts
- The context of software design
- The software design process
- Basic software design concepts
- Key issues in software design

### II. Software Architecture
- Architectural structures and viewpoints
- Architectural styles and patterns (macro-architecture)
- Design patterns (micro-architecture)
- Design of families of programs and frameworks

### III. Software Design Quality Analysis and Evaluation
- Quality attributes
- Quality analysis and evaluation tools
- Metrics

### IV. Software Design Notations
- Structural descriptions (static view)
- Behavioural descriptions (dynamic view)

### V. Software Design Strategies and Methods
- General strategies
- Function-oriented design
- Object-oriented design
- Data-structure centered design
- Other methods

### VI. Software Design Tools
- Mathematical tools
- CASE tools
- Tools for metrics

### VII. Standards relevant to Software Design

First and foremost, the breakdown of topics must describe “generally accepted” knowledge, that is, knowledge for which there is a “widespread consensus”. Furthermore, and this is clearly where this becomes difficult, such knowledge must be “generally accepted” today and expected to be so in a 3 to 5 years timeframe. In this spirit, the following topics have been included: i) elements related with software
architecture (e.g., “Software Architecture in Practice”, Bass, Clements and Kazman, 1998; “Pattern-oriented software architecture”, Buschmann et al., 1996), including notions related with architectural styles; ii) the approach to OOD recently presented by the UML group (“The Unified Software Development Process”, Jacobson, Booch and Rumbaugh, 1999); iii) component-based design (“Objects, Components, and Frameworks with UML – The Catalysis Approach.”, D’Souza and A.C. Wills, 1999). Note that although “UML” (Unified Modeling Language) is not explicitly mentioned in the Design Notations section, many of its elements are indeed present, for example: class and object diagrams, collaboration diagrams, deployment diagrams, sequence diagrams, statecharts.

The need for the breakdown to be independent of specific application domains, life cycle models, technologies, development methods, etc., and to be compatible with the various schools (churches?) within software engineering, is particularly apparent within the “Software Design Strategies and Methods” section, where numerous approaches and methods have been included and references given. This is also the case in the “Software Design Notations”, which incorporates pointers to many of the existing notations and description techniques for software design artifacts. Although many of the design methods use specific design notations and description techniques, most of these notations are generally useful independently of the particular methods which use them. Note that this is also the approach used in many software engineering books, including the recent UML series of books by the three amigos, which describe “The Unified Modeling Language” apart from “The Unified Software Development Process”.

The specifications document also specifically asked that the breakdown be as inclusive as possible and that it includes topics related with quality, measurements, tools and standards. Thus, a certain number of topics have been included in the list of topics even though they may not yet be fully considered as generally accepted. For example, although there are a number of books on metrics, design metrics per se is rarely discussed in detail and few “mainstream” software engineering books formally discuss this topic. But it is indeed discussed in some books and may become more mainstream in the coming years. Note that although those metrics can sometimes be categorized into high-level (architectural) design vs. component-level (detailed) design, the use of such metrics generally depend on the approach used for producing the design, for example, structured vs. object-oriented design. Thus, the latter way of defining the metrics sub-topics has been used.

Concerning the topic of Standards, there seems to be few standards that directly pertain to the design task per se. But there does exist standards which have some relationships with various aspects of Software Design, e.g., OMG standards for UML or CORBA, or “IEEE Recommended Practice for Software Design Descriptions”. An independent section on “Existing Standards Relevant with Software Design” has been created to collect together all such standards. As for Tools, a similar independent section has been created. Again, there are tools that can be used in the design process. However, there seems to be few generally used tools that pertain directly and exclusively to Software Design, except those for developing various kinds of diagrams that can be found in general CASE tools. The previous version of the KA Description (0.5) did not include any specific mention of tools, and this was noted by many reviewers. However, because hardly any reviewer suggested specific tools or references to be included in the Tools topic of Software Design KA Description, few references have been provided for this particular topic.

As required by the KA Description Specifications, the breakdown is at most three levels deep and use topic names which, from our survey of the existing literature and after having made a number of modifications suggested by the reviewers, should be meaningful when cited outside SWEBOK.
By contrast with the previous version (0.50) of the Software Design KA Description, and following suggestions made by a number of reviewers, the Software Design Basic Concepts has been expanded to include topics related with design in general and topics introducing the context and process of software design. A totally new subsection has also been recently added: Key Issues in Software Design. The reason for this new subsection is that a number of reviewers suggested that certain topics, not explicitly mentioned in the previous version, be added, e.g., concurrency and multi-threading, exception handling. Although some of these aspects are addressed by some of the existing design methods, it seemed appropriate that these key issues be explicitly identified and that more specific references be given for them, thus the addition of this new subsection. (Important note: this is a first attempt at such a description of this topic and the author of the Software Design KA Description would gladly welcome any suggestions that could improve and/or refine the content of this subsection.)

In the KA breakdown, as mentioned earlier, an explicit Software Architecture section has been included. Here, the notion of “architecture” is to be understood in the large sense of defining the structure, organization and interfaces of the components of a software system, by opposition to producing the “detailed design” of the specific components. This is what really is at the heart of Software Design. Thus, the Software Architecture section includes topics which pertain to the macro-architecture of a system – what is now becoming known as “Architecture per se, including notions such as “architectural styles” and “family of programs” – as well as topics related with the micro-architecture of the smaller subsystems – for example, lower-level design patterns. Although some of these topics are relatively new, they should become much more generally accepted within the 3-5 years timeframe expected from the SWEBOK specifications. By contrast, note that no explicit “Detailed Design” section has been included: topics relevant to detailed design can implicitly be found in the Software Design Notations and Software Design Strategies and Methods sections, as well as in “The Software Design Process” section.

The Software Design Strategies and Methods has been divided, as is done in many books discussing software design, in a first section that presents general strategies, followed by subsequent sections that present the various classes of approaches (data-, function-, object-oriented or other approaches). For each of these approaches, numerous methods have been proposed and can be found in the software engineering literature. Because of the limit on the number of references, we have mostly given general references, which can then be used as starting point for more specific references. In the particular case of Object-Oriented Design (OOD), the Unified Software Development Process recently proposed by the UML group, which can be considered a kind of synthesis of many earlier well-known approaches (Booch, OMT, OOSE), was a must, even though it is quite recent (1999). For similar reasons, the Software Design Notations section mentions most of the elements that can be found in UML.

A last point worth explaining is the omission of a number of topics which contain “Design” in their name and which, indeed, pertain to the development of software systems. Among these are the followings: User Interface Design, Real-time Design, Database Design, Participatory Design, Collaborative Design. The first two were specifically excluded, in the Straw Man document, from the Software Design KA. The third one, Database Design, can be considered as specialized knowledge, part of the Computer Science Related Discipline. As for the last two, they are related more to the Requirements Analysis KA, rather than Software Design. In the terminology of DeMarco (DeM99), these topics belong more appropriately to I-Design (invention design, done by system analysts) rather than D-design (decomposition design, done by designers and coders) or FP-design (family pattern design, done by architecture groups). It is mainly the latter two, with a major emphasis on D-design, which can be considered as generally accepted knowledge related with Software Design.

3. Description of the topics of the Software Design KA

In this section, we give brief descriptions of each of the major topics of the Software Design Knowledge Area. These brief descriptions should be sufficient to guide the reader, in section 4, to the appropriate recommended references.
I. Software Design Basic Concepts

- General design concepts: Notions and concepts relevant to design in general, e.g., design as wicked problem solving; prominent concepts of design: goals, constraints, alternatives, representations, and solutions.

- The context of software design: The context (software development life cycle) in which software design fits: software requirements analysis vs. software design; software design vs. software construction. Traceability between the work products of the various phases.

- The software design process: The general process by which software is designed: Architectural vs. detailed design = the two classical phases of software design: whereas architectural design describes how the system is decomposed and organized into components, detailed design describes the specific behaviour of these components. The distinction made by some authors between software architecture and architectural design should also be mentioned: architectural design is often considered as defining the software architecture of a specific system, whereas the process of defining a software architecture is considered more generic.

- Basic software design concepts: Key notions generally considered fundamental to software design, as they form kind of a foundation for understanding many of the proposed approaches to software design: abstraction, modularity (including notions like cohesion and coupling), encapsulation and information hiding, hierarchy, interface vs. implementation, separation of concerns and locality.

- Key Issues in Software Design: The key issues which must be dealt with when designing a software system:
  - Modularity and partitioning: how to ensure the software is constructed in a modular way, in order to make it understandable and modifiable.
  - Concurrency considerations: how to decompose the systems into processes, tasks and threads and deal with related atomicity, synchronization and scheduling issues.
  - Control issues and handling events: how to organize the flow of control, how to handle reactive and temporal events through various mechanisms, e.g., implicit invocation and call-backs, etc.
  - Distribution: how the software is distributed on the hardware, the role of middleware when dealing with heterogeneous systems, etc.
  - Handling of faults and exceptions: how to prevent and tolerate faults and deal with exceptional conditions.
  - Interactive systems and dialogue independence: how to separate the details of the user-interface from the business logic. (Note: User Interface design per se is not discussed in the current KA.)
  - Platform independence: how to ensure the software is relatively independent of the platform (hardware, OS, prog. lang.) on which it will run.

II. Software Architecture

This section on software architecture includes topics dealing both with “generic” software architecture and architectural design of a “specific” software system, the frontier between the two not always being clear-cut.

- Architectural structures and viewpoints: The different high-level facets of a software design that should be described and documented. For some authors, these views pertain to different issues associated with the design of software, for example, the logical view (satisfying the functional requirements) vs. the process view (concurrency issues) vs. the physical view (distribution issues) vs. the development view (how the design is implemented). Other authors use different terminologies, e.g., behavioural vs. functional vs. structural vs. data modeling views.

- Architectural styles and patterns (macro-architecture): The notion of architectural style – an architectural style is a paradigmatic architectural pattern that can be used to develop the high-level organization of a software system – is becoming an important notion of the field of software architecture. This section presents some of the major styles that have been identified by various authors. These styles are (tentatively) organized as follows:
  - General structure (e.g., layers, pipes and filters, blackboards);
- Distributed systems (e.g., client-server, three-tiers, broker);
- Interactive systems (e.g., Model-View-Controller, Presentation-Abstraction-Control)
- Adaptable systems (e.g., micro-kernel, reflection);
- Other styles (e.g., batch, interpreters, process control, rule-based).

**Design patterns (micro-architecture):** In the last few years, the field of software design patterns has emerged as an important approach to describing, and thus reusing, design knowledge. Whereas architectural styles can be seen as patterns describing the high-level organization of software systems, its macro-architecture, other design patterns can be used to describe details at a lower-level, at a micro-architecture level. Such design patterns can (tentatively) be categorized as follows:
- Creational patterns: builder, factory, prototype, singleton, etc.
- Structural patterns: adapter, bridge, composite, decorator, facade, flyweight, proxy, etc.
- Behavioural patterns: command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor, etc.

**Design of families of programs and frameworks:** One approach to allow the reuse of software design is to design **families** of systems; this can be done by identifying exploitable commonalities among members of such families. Particularly in the field of OO programming, this has been made possible by the notion of framework: a framework is a partially complete software subsystem which can be extended by appropriately instantiating some specific plug-ins (hot points).

### III. Software Design Quality Analysis and Evaluation

**Quality attributes:** Various attributes are generally considered important for obtaining a design of good quality, e.g., various “ilities” (e.g., maintainability, testability, traceability, plus many others), various “nesses” (e.g., correctness, robustness), including “fitness of purpose”. Because there are so many of them, no specific list is given.

**Quality analysis and evaluation tools:** Conceptual or technical tools and techniques that can help ensure the quality of a design:
- Software design reviews: informal, often group-based, techniques to verify and ensure the quality of design documents.
- Static analysis: formal, design-time thus non-executable (static), analysis that can be used to evaluate a design.
- Simulation and prototyping: dynamic techniques to evaluate a design.

**Metrics:** Formal metrics that can be used to estimate various aspects of the size, structure or quality of a design. Most such metrics generally depend on the approach used for producing the design:
- Functional (structured) design metrics: e.g., structural complexity, morphology metrics, etc.
- Object-oriented design metrics: weighted methods per class, depth of inheritance tree, etc.

### IV. Software Design Notations

A large number of notations and languages exist to represent software design artifacts. Some are used mainly to describe the structural organization of a design, whereas others are used to represent the behaviour of such software systems.

**Structural descriptions (static view):** Notations, mostly graphical, that can be used to describe and represent the structural aspects (static view) of a software design, that is, to describe what the major components are and how they are interconnected. Such notations can be used to describe the logical (e.g., Architecture Description Languages (ADL), class diagrams, subsystems and packages), process (active objects and classes) or physical (e.g., deployment diagrams) views of a software design.

**Behavioural descriptions (dynamic view):** Notations and languages used to describe the dynamic behaviour of systems and components. These include various graphical notations (e.g., activity diagrams, Data Flow Diagrams (DFD), sequence diagrams, state transition diagrams) and various textual notations (e.g., formal specification languages, pseudo-code and Program Design Languages (PDL)).
V. Software Design Strategies and Methods

- General strategies: General strategies that can be used to design a system, e.g., divide-and-conquer, information hiding, use of patterns and pattern languages, iterative and incremental approach to design, etc. Methods, in contrast with general strategies, are more specific in that they generally provide i) a set of notations to be used with the method; ii) a description of the process to be used when following the method; iii) a set of heuristics that provide guidance in using the method. A number of methods are described in the following sections.

- Function-oriented (structured) design: One of the classical approach to software design, where the decomposition is centered around the identification of the major systems functionalities and their elaboration and refinement in a top-down manner. Structured design is generally used after a structured analysis (viz., using DFDs and Entity-Relationship Diagrams (ERDs)) has been produced; various strategies (e.g., transformation analysis, transaction analysis) and heuristics (fan-in/fan-out, scope of effect vs. scope of control, etc.) have thus been proposed to transform a DFD into a software architecture generally represented by a structure chart (identifying which modules uses/calls which other).

- Object-oriented design: This is probably the most (still?) flourishing field of software design in the last 10-15 years, as numerous software design methods based on objects have been proposed. The field evolved from the early object-based design of the mid-1980’s (noun = object; verb = method; adjective = attribute) through object-oriented design, where inheritance and polymorphism play a key role, and to the now emerging field of component-based design, where various meta-information can be defined and accessed (e.g., through reflection). Although object-oriented design’s deep roots stem from the concept data abstraction, the notion of responsibility-driven design has also become an important approach to object-oriented design.

- Data-structure centered design: Although less popular in North America than in Europe, there has been some interesting work (e.g., M. Jackson, Warnier-Orr) on designing a program starting from the data structures it manipulates rather than from the function it performs. The structures of the input and output data are first described (e.g., using Jackson structure diagrams) and then the program is developed based on these data structure diagrams. Various heuristics have been proposed to deal with special cases, for example, when there is mismatch between the input and output structures.

- Other methods: Although software design based on functional decomposition or on object-oriented design are probably the most well-known approaches to software design, other interesting approaches, although probably less “mainstream”, do exist, e.g., formal and rigorous methods (e.g., VDM and Cleanroom), knowledge-based approaches, transformational methods, etc.

VII. Software Design Tools

Various tools exist that can be used and applied when designing software. These range from mathematical tools, e.g., predicate logic and set theory as used in specification languages, to CASE tools that can be used throughout the software development life cycle, to tools that can compute various metrics associated with a design.

VII. Standards relevant to Software Design

This section simply lists the existing standards that seem relevant to the software design KA. This list includes standards for various design representations (e.g., UML, Ada as a PDL), standards related with distributed processing (e.g., CORBA) or standards defining the major quality characteristics of software product.

4. Recommended reference material

In what follows, we suggest reference material for the various topics presented in the proposed breakdown of topics. First, a brief presentation of each of the recommended reference is given. Then, references are given for each of the major topics of the breakdown. A matrix of the reference material vs. each topic is
presented in appendix B. Note that, for some topics, a number of global references are given for a non-leaf topic, rather a specific reference for each particular leaf topic. This seemed preferable because some of these topics were discussed in a number of interesting references.

It should be noted that the number of recommended references (18) is only slightly above the required 15 references asked for in the specification document for the knowledge area descriptions. But note that references to standards have not been included in this list. It must also be stressed that the number of recommended references has not gone (largely) over the limit of 15 mainly because almost no specific references have been given for the various software design methods except very general ones. For example, only to reflect the state of the OO design methods field, an additional 5-10 references should probably have been indicated, for example: Boo94, Cop99, CY91, JBP+91, JCJO92, Pre95, Zsy98, WBWWW90.

4.1. Brief description of the recommended references

[BCK98]
A recent and major work on software architecture. It covers all the major topics associated with software architecture: what software architecture is, quality attributes, architectural styles, enabling concepts and techniques (called unit operations), architecture description languages, development of product lines, etc. Furthermore, it presents a number of case studies illustrating major architectural concepts, including a chapter on CORBA and one on the WWW.

[BMR+96]
According to the Software Design KA Description author’s humble opinion, this is probably the best and clearest introduction to the notions of software architecture and patterns (both architectural and lower-level ones). Distinct chapters are dedicated to architectural patterns, design patterns and lower-level idioms. Another chapter discusses the relationships between patterns, software architecture, methods, frameworks, etc. This chapter also includes an interesting presentation of so-called “enabling techniques for software architecture”, containing many of the elements of the Basic software design concepts, e.g., abstraction, encapsulation, information hiding, coupling and cohesion, etc.

[BRJ99]
A comprehensive and thorough presentation of UML, which incorporates many of the notations mentioned in the Software Design Notations section.

[DT97]
M. Dorfman and R.H. Thayer (eds.). *Software Engineering.*
A collection of papers on software engineering in general. Two chapters deal with software design. One of them contains a general introduction to software design, briefly presenting the software design process and the notions of software design methods and design viewpoints. The other chapter contains an introduction to object-oriented design and a comparison of some existing OO methods.

[DW99]
A thorough presentation of a specific OO approach with an emphasis on component design. The development of static, dynamic and interaction models is discussed. The notions of components and connectors are presented and illustrated with various approaches (Java Beans, COM, Corba); how to use such components in the development of frameworks is also discussed. Another chapter discusses various aspects of software architecture. The last chapter introduces a pattern system for dealing with both high-level and detailed design, the latter level touching on many key issues of design such as concurrent, distribution, middleware, dialogue independence, etc.
[FW83]
Although this is an old book, it is a very interesting one because it allows to better understand the evolution of the software design field. This book is a collection of papers where each paper presents a software design technique. The techniques range from basic strategies like stepwise refinement to, at the time, more refined method such as structured design à la Yourdon and Constantine. Historically important reference.

[FP97]
A detailed presentation of numerous software metrics. Although the metrics are not necessarily presented based on the software development life cycle, many of those metrics, especially in chapter 7 and 8, are applicable to software design.

[GHJV95]
E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software.
The seminal work of design patterns. A detailed catalogue of patterns related mostly with the micro-architecture level.

[Hut94]
Describe and compare, in a very outlined manner, a large number of OO analysis and design methods. Useful as a starting point for obtaining additional pointers and references to OOD methods, not so much as a detailed presentation of those methods.

[Jal97]
P. Jalote. An integrated approach to software engineering, 2nd ed.
A general software engineering textbook with a good coverage of software design, as three chapters are dedicated to design: one on function-oriented design, one on object-oriented design, and the other on detailed design. Another interesting point is that all these chapters have a metrics section.

[JBR99]
A detailed and thorough presentation of the Unified Software Development Process proposed by the Rational amigos. The notion of architecture plays a central role in this development process, the process being said to be architecture-centric. However, the associated notion of architecture is slightly different from the traditional purely design-based one: an architecture description is supposed to contain views not only from the design model but also from the use-case, deployment and implementation models. A whole chapter is devoted to the presentation of the iterative and incremental approach to software development. Another chapter is devoted to design per se, whose goal is to produce the design model, which includes the logical (e.g., class diagrams, collaborations, etc.) and process (active objects) views, and also the deployment model (physical view).

[Mar94]
A general encyclopedia that contains (at least) three interesting articles discussing software design. The first one, “Design” (K. Shumate), is a general overview of design discussing alternative development processes (e.g., waterfall, spiral, prototyping), design methods (structured, data-centered, modular, object-oriented). Some issues related with concurrency are also mentioned. The second one discuss the “Design of distributed systems” (R.M. Adler): communication models, client-server and services models. The third one, “Design representation” (J. Ebert), presents a number of approaches to the representation of design. It is clearly not a detailed presentation of any method; however, it is
interesting in that it tries to explicitly identify, for each such method, the kinds of components and
connectors used within the representation.

[Mey97]
A detailed presentation of the Eiffel OO language and its associated Design-By-Contract approach,
which is based on the use of formal assertions (pre/post-conditions, invariants, etc). It introduces the
basic concepts of OO design, along with discussion of other key issues of design, e.g., concurrency,
persistence.

[MNS96]
H.A. Muller, R.J. Norman, and J. Slonim (eds.). Computer Aided Software Engineering.
A collection of papers on CASE tools. Three of the papers deal with design related tools, including a
knowledge-based one.

[Pfl98]
A general software engineering book with a single chapter devoted to design. Briefly presents and
discusses some of the major architectural styles and strategies and some of the concepts associated
with the issue of concurrency. Another section presents the notions of coupling and cohesion and also
deals with the issue of exception handling. Techniques to improve and to evaluate a design are also
presented: design by contract, prototyping, reviews. Although this chapter does not go very deep into
any topic, it can be a starting point for a number of issues not discussed in other general software
engineering textbooks.

[Pre97]
Probably the classic among all the general software engineering textbooks (4th edition!) Contains over
10 chapters that deal with notions associated with software design in one way or another. The basic
concepts and the design methods are presented in two distinct chapters. Furthermore, the topics
pertaining to the functional-based (structured) approach are separated (part III) from those pertaining to
the object-oriented approach (part IV). Independent chapters are also devoted to metrics applicable to
each of those approaches, a specific section addressing those metrics specific to design. A chapter
discusses formal methods and another presents the Cleanroom approach. Finally, a chapter discusses
client-server systems and distribution issues, whereas another one discusses CASE tools.

[Rie96]
This book, targeted mainly towards OO design, presents a large number of heuristics that can be used
in software design. Those heuristics address a wide range of issues, both at the architectural level and
at the detailed design level.

[SB93]
An interesting paper that discusses what is design in general. More specifically, it presents the five
basic concepts of design: goals, constraints, alternatives, representations, and solutions. The
bibliography is a good starting point for obtaining additional references on design in general.

4.2. Recommended references for each of the KA topic

Note: The numbers after the reference key indicate the appropriate chapter. In the case of Mar94, the
appropriate section are indicated as follows: “D” = Design; “DR” = Design Representation; “DD” = Design
of Distributed systems"
I. Software Design Basic Concepts

General design concepts
[SB93]

The context of software design
[DT97: 1, 11]
[Ja97: 2]
[BR99: 6-10]
[Pr98: 2]
[Pre97: 2]

The software design process
[BCK98: 2]
[BMR+96: 6]
[DW99: 12]
[Ja97: 7]
[BR99: 9]
[Mar94]
[Pr98: 5]
[Pre97: 2]

Basic software design concepts
[BMR+96: 6]
[Ja97: 5]
[Pr98: 5]
[Pre97: 13, 23]

Key Issues in Software Design
Concurrent considerations
[Mey97: 30][DW99: 16][BR99: 9]
Control and handling of events
[BCK98: 5][BMR+96: 2][GHJV95: 3]
Distribution
[BMR+96: 2][BCK98: 8][DW99: 16][Mar94]
[Mey97: 30][Pre97: 28]
Exception handling
[Mey97: 12][Pr98: 5]
Interactive systems and dialogue independence
[BMR+96: 2.4][DW99: 12, 16][GHJV95: 3]
[Mey97: 32][Rie96: 3]
Platform independence
[DW99: 16][Mey97: 32][Rie96: 9]
Modularity and partitioning
[BMR+96: 6][DW99: 16][Mey97: 3][Pr98: 5][Rie96: 2-4]

II. Software Architecture

Architectural structures and viewpoints
[BCK98: 2]
[BR99: 31]
[BR99: 4]
[DT97: 4]
[BMR+96: 6]
[Pr98: 5]
Architectural styles and patterns (macro-architecture)
[BCK98: 5]
[BMR+96: 2]
[BRJ99: 28]
[Pfl98: 5]

Design patterns (micro-architecture)
[BCK98: 13]
[BMR+96: 3]
[BRJ99: 28]
[GHJV95: 3-5]

Design of families of programs and frameworks
[BCK98: 15]
[BMR+96: 6]
[BRJ99: 28]
[DW99: 9, 11]

III. Software Design Quality Analysis and Evaluation

Quality attributes
[BCK98: 4]
[BMR+96: 6]
[DT97: 4]
[Jal97: 5]
[Mar94]
[Mey97: 3-4]
[Pfl98: 5]
[IEE91]
[ISO91]

Quality analysis and evaluation tools
[BCK98: 9-10]
[Jal97: 5, 7]
[Pfl98: 5]

Metrics
[FP97: 7-9]
[Jal97: 5-7]
[Pre97: 18, 23]

IV. Software Design Notations

Structural descriptions (static view)
ADL (Architecture Description Languages)
[BCK98: 12]
Class and objects diagrams
[BRJ99: 4-10, 14][DW99: 2][Jal97: 6][Mey97: 7]
CRC (Class-Responsibilities-Collaborators) Cards
[BRJ99: 4][BMR+96][Hut94: 21]
Deployment diagrams
[BRJ99: 30]
ERD (Entity-Relationship Diagrams)
[DT97: 4][Mar94: DR]
IDL (Interface Description Languages)
[BCK98: 8][BJR99: 11][BMR+96: 2]
Jackson structure diagrams
[DT97: 4][Mar94: DR]
Structure charts
[DT97: 4-5][Jal97: 5][Mar94: DR][Pre97: 12, 14]
Subsystems (packages) diagrams
[BRJ99: 12, 31][DW99: 7]

Behavioural descriptions (dynamic view)
Activity diagrams
[BRJ99: 19]
Collaboration diagrams
[BRJ99: 18]
Data flow diagrams
[Jal97: 5][Mar94: DR][Pre97: 14]
Decision tables and diagrams
[Pre97: 14]
Flowcharts and structured (Nassi-Schneiderman) flowcharts
[FW83: VII][Mar94: DR][Pre97: 14]
Formal specification languages: VDM, Z, CCS/CSP, SDL, etc.
[DT97: 5][DW99: 2][Mey97: 11][Pfl98: 5][Pre97: 24]
Pseudo-code and PDL (Program Design Language)
[FW83: VII][Jal97: 7][Pre97: 14]
Sequence diagrams
[BRJ99: 18]
State transition diagrams and statecharts

V. Software Design Strategies and Methods

General strategies
Data abstraction and information hiding [FW83: V][Mey97: 6]
Heuristics-based design [Pre97: 13][Rie96]
Iterative and incremental design [JBR99: 5][Pfl98: 2]
Pattern-based design and pattern languages [DW99: 16][GHJV95]
Divide-and-conquer and stepwise refinement [FW83: VII]

Function-oriented design
[DT97: 5][FW83: V][Jal97: 5][Pre97: 13-14]

Object-oriented design
[JBR99: 9][Mar94: DI][Mey97][Pfl98: 5][Pre97: 19-21][Rie96]

Data-structure centered design
[DT97: 4][FW83: III, VII][Mar94: D]

Other methods
Formal and rigorous methods, e.g., Cleanroom, VDM, Design by Contract
[DT97: 5][Mey97: 11][Pfl98: 5][Pre97: 24]
Knowledge-based approaches [MNS96]
Transformational methods [Pfl98: 2]

VI. Software Design Tools

Mathematical tools [See: Formal methods, Related Disciplines]
CASE tools [MNS96][DT97: 12][Pre97: 29]
Metrics calculators [FP97: 8, 13][Jal97: 5]
VII. Standards relevant to Software Design

[OMG99]
[IEE88]
[IEE88b]
[IEE91]
[IEE98]
[IEE98b]
[ISO91]
[ISO95]
[ISO95b]
[OMG98]
[otSESC98]

5. Relevant Knowledge Areas of the Related Disciplines

In this section, we briefly present the Knowledge Areas’ topics from the Related Disciplines that can be considered relevant to the Software Design KA. More precisely, these are the topics that can/should be assumed to be known by any Software Designer as they might be relevant, directly or indirectly, to Software Design.

A. Computer Science
1. Foundations: complexity analysis; discrete mathematics; automata; formal specifications.
2. Algorithms and Data Structures: basic data structures; abstract data types; sorting and searching.
3. Computer architecture: memory system organization and architecture; interfacing and communication.
4. Information Management: database models; transaction processing; data compression.
5. Operating Systems: tasks, processes and threads; process coordination and synchronization; file systems; networking fundamentals; security; protection; distributed systems.
7. Net-Centric Computing: distributed objects computing (DOC/CORBA/DCOM/JVM); enterprise computing; network-level security.
8. Social, Ethical, Legal and Professional Issues.

B. Mathematics
• Discrete mathematics.
• Probability.
• Mathematical logic.

C. Project Management
• Project integration management.
• Project scope management.
• Project time management.
• Project cost management.
• Project quality management.
• Project risk management.

D. Computer Engineering
• Systems Design.

E. Systems Engineering
• Process: behavioural analysis; prototyping; system breakdown structure; design; component specification.

F. Cognitive Sciences and Human Factors
• Computer Systems and Interface Architecture: dialogue architecture.
• Development Process: design approaches; evaluation techniques.
Appendices

A. Detailed breakdown of the Software Design KA

I. Software Design Basic Concepts

General design concepts
The context of software design
The software design process

Basic software design concepts
- Abstraction
- Cohesion/coupling and modularity
- Encapsulation and information hiding
- Hierarchy and refinement
- Interface vs. Implementation
- Separation of concerns and locality

Key Issues in Software Design
- Concurrency considerations
- Control and handling of events
- Distribution
- Exception handling
- Interactive systems and dialogue independence
- Platform independence
- Modularity and partitioning

II. Software Architecture

Architectural structures and viewpoints

Architectural styles and patterns (macro-architecture)
- General structure (e.g., layers, pipes and filters, blackboards)
- Distributed systems (e.g., client-server, broker)
- Interactive systems (e.g., Model-View-Control, Presentation-Abstraction-Control)
- Adaptable systems (e.g., micro-kernel, reflection)
- Other styles (e.g., batch, interpreters, process control, rule-based)

Design patterns (micro-architecture)
- Creational patterns (e.g., builder, factory, prototype, singleton)
- Structural patterns (e.g., adapter, bridge, composite, decorator, facade, flyweight, proxy)
- Behavioural patterns (e.g., command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor)

Design of families of programs and frameworks

III. Software Design Quality Analysis and Evaluation

Quality attributes
- Fitness of purpose, “ilities”, and “ness”

Quality analysis and evaluation tools
- Software design reviews
- Static analysis
- Simulation and prototyping

Metrics
- Functional (structured) design metrics
- Object-oriented design metrics
IV. Software Design Notations

Structural descriptions (static view)
- ADL (Architecture Description Languages)
- Class and objects diagrams
- CRC (Class-Responsibilities-Collaborators) Cards
- Deployment diagrams
- ERD (Entity-Relationship Diagrams)
- IDL (Interface Description Languages)
- Jackson structure diagrams
- Structure charts
- Subsystems (packages) diagrams

Behavioural descriptions (dynamic view)
- Activity diagrams
- Collaboration diagrams
- Data flow diagrams
- Decision tables and diagrams
- Flowcharts and structured (Nassi-Schneiderman) flowcharts
- Formal specification languages: VDM, Z, CCS/CSP, SDL, Petri nets, etc.
- Pseudo-code and PDL (Program Design Language)
- Sequence diagrams
- State transition diagrams and statecharts

V. Software Design Strategies and Methods

General strategies
- Data abstraction and information hiding
- Heuristics-based design
- Iterative and incremental design
- Pattern-based design and pattern languages
- Divide-and-conquer and stepwise refinement

Function-oriented design

Object-oriented design

Data-structure centered design

Other methods
- Formal and rigorous methods, e.g., Cleanroom, VDM
- Knowledge-based approaches
- Transformational methods

VI. Software Design Tools

Mathematical tools

CASE tools

Metrics calculators
VII. Standards relevant to Software Design

OMG Unified Modeling Language specification, v. 1.3. [OMG99]

IEEE Standard Glossary of Software Engineering Terminology. [IEE91]

IEEE recommended practice for software design descriptions. [IEE98]

Information technology – Software product evaluation – Quality characteristics and guidelines for their use. [ISO91]

Open distributed processing – Reference model. [ISO95]

Information technology – Software life cycle processes. [ISO95b]

The Common Object Request Broker: Architecture and Specification. [OMG98]

Draft recommended practice for information technology – system design – architectural description. [otSESC98]

B. Matrix of Software Design topics vs. recommended reference material

Note: A number in the matrix entry indicates the appropriate chapter number. A “**” indicates a general reference (no specific chapter). A “—” indicates a reference to an appendix. For Mar94, see the note in section 4.2.

Note: Except for the Key Issues in Software Design section, only the top two level of the breakdown have been indicated in the matrix. Otherwise, especially in the Software Design Notations subsections, this lead to very sparse lines (in an already quite sparse matrix).

| B | C | K | R | D | T | J | W | F | G | H | J | A | I | M | M | P | P | R | S | B |
| 1 | 11 | 6 | 9 | 7 | 9 | 9 | 7 | 9 | 9 | 7 | 9 | 9 | 7 | 9 | 9 | 7 | 9 | 6 | 8 | 7 | 9 | 3 |

<table>
<thead>
<tr>
<th>I. Software Design Basic Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>General design concepts</td>
</tr>
<tr>
<td>The context of software design</td>
</tr>
<tr>
<td>The software design process</td>
</tr>
<tr>
<td>Basic software design concepts</td>
</tr>
<tr>
<td>Key issues in software design</td>
</tr>
<tr>
<td>Concurrency</td>
</tr>
<tr>
<td>Control and events</td>
</tr>
<tr>
<td>Distribution</td>
</tr>
<tr>
<td>Exceptions</td>
</tr>
<tr>
<td>Interaction independence</td>
</tr>
<tr>
<td>Platform independence</td>
</tr>
<tr>
<td>Modularity and partitioning</td>
</tr>
</tbody>
</table>

KA_Description_for_Software_Design(version_0_6).doc 17
**II. Software architecture**

<table>
<thead>
<tr>
<th>Architectural structures and viewpoints</th>
<th>B</th>
<th>C</th>
<th>K</th>
<th>R</th>
<th>J</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural styles and patterns (macro-arch.)</td>
<td>2</td>
<td>6</td>
<td>31</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design patterns (micro-arch.)</td>
<td>13</td>
<td>3</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Families of programs and frameworks</td>
<td>15</td>
<td>6</td>
<td>28</td>
<td>9</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**III. Software design quality analysis and evaluation**

| Quality attributes | 4 | 6 | 4 | | | | |
| Quality analysis and evaluation | 9-10 | 5 | 7 | | | | |
| Metrics | 7-9 | 5-7 | | | | | |

**IV. Software design notations**

<table>
<thead>
<tr>
<th>Structural descriptions (static)</th>
<th>8</th>
<th>12</th>
<th>4-12, 14, 30-31</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioural descriptions (dynamic)</td>
<td>18, 19, 24</td>
<td>5</td>
<td>2, 3</td>
<td>VII</td>
<td>5, 7</td>
<td>DR</td>
<td>11</td>
</tr>
</tbody>
</table>

**V. Software design strategies and methods**

| General strategies | 4 | 1 | V-VII | * | 5 | 6 | 2 | 13 | * |
| Function-oriented design | 5 | V | 5 | | | | | | |
| OO design | 5 | II | VI | 4, 7, 19, 21 | 6 | 9 | D | * | 5 | 19-21 | * |
| Data-oriented design | 4 | III, VII | | | | D | | | | |
| Other methods | 5 | | | | | | | | | |

**VI. Software design tools**

| Mathematical tools | 5 | | | | | | 5 | 24 |
| CASE Tools | 12 | | | | | | | |
| Metrics calculator | 8, 13 | 5 | | | | | | | |
### C. Classification of the Software Design topics based on Vincenti’s categories

Legend: FC = Fundamental Concepts; CaS = Criteria and Specifications; TT = Theoretical Tools; QD = Quantitative Data; PC = Practical Considerations; DI = Design Instrumentalities

<table>
<thead>
<tr>
<th>Software Design Topic</th>
<th>FC</th>
<th>CaS</th>
<th>TT</th>
<th>QD</th>
<th>PC</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I. Software Design Basic Concepts</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General design concepts</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The context of software design</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The software design process</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic software design concepts</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key issues in software design</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>II. Software Architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural structures and viewpoints</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural styles and patterns (macro-architecture)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design patterns (micro-architecture)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design of families of programs and frameworks</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>III. Software Design Quality Analysis and Evaluation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality attributes</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality analysis and evaluation tools</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metrics</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IV. Software Design Notations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural descriptions (static view)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behavioural descriptions (dynamic view)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>V. Software Design Strategies and Methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General strategies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Function-oriented design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Object-oriented design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Data-structure centered design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Other methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>VI. Software Design Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical tools</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CASE tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Tools for metrics</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>VII. Standards relevant to Software Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** The above is a first tentative classification of the Software Design KA topics based on Vincenti’s categories. The correspondence with these categories is not obvious to establish (and the author would welcome any suggestions or alternative classification). It can be seen that the categories are not mutually exclusive. It can also be seen that, probably contrary to more established fields of engineering, few elements belong to the Quantitative Data category.
### D. Rating of the Software Design KA topics based on Bloom’s taxonomy

<table>
<thead>
<tr>
<th>Software Design Topic</th>
<th>Knowledge</th>
<th>Comprehension</th>
<th>Application</th>
<th>Analysis</th>
<th>Synthesis</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I. Software Design Basic Concepts</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General design concepts</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The context of software design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The software design process</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basic software design concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Key issues in software design</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>II. Software Architecture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architectural structures and viewpoints</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Architectural styles and patterns (macro-architecture)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Design patterns (micro-architecture)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Design of families of programs and frameworks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>III. Software Design Quality Analysis and Evaluation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Quality analysis and evaluation tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>IV. Software Design Notations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structural descriptions (static view)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Behavioural descriptions (dynamic view)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>V. Software Design Strategies and Methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General strategies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Function-oriented design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Object-oriented design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Data-structure centered design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Other methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>VI. Software Design Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematical tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CASE tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Tools for metrics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>VII. Standards relevant to Software Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

**Note:** As mentioned in the URL used as reference for “Bloom’s et al.’s Taxonomy of the Cognitive Domain”, we considered Evaluation to be at the same level as Synthesis, but using different cognitive processes.
References

[BCK98]
L. Bass, P. Clements, and R. Kazman.
*Software Architecture in Practice.*

[BDA+98]
Guide to the software engineering body of knowledge – a straw man version.

[BMR+96]
*Pattern-oriented software architecture – A System of Patterns.*

[Boo94]
G. Booch.
*Object Oriented Analysis and Design with Applications, 2nd ed.*

[BRJ99]
Addison-Wesley, Reading, MA, 1999.

[Cop99]
J. Coplien.
*Multi-Paradigm Design for C++.*
Addison-Wesley, 1999.

[Cro84]
N. Cross (ed.).
*Developments in Design Methodology.*

[CY91]
P. Coad and E. Yourdon.
*Object-Oriented Design.*

[DeM99]
T. DeMarco.
*The Paradox of Software Architecture and Design.*
Stevens Prize Lecture, August 1999.

[DT97]
M. Dorfman and R.H. Thayer.
*Software Engineering.*

[DW99]
D.F. D’Souza and A.C. Wills.
*Objects, Components, and Frameworks with UML – The Catalysis Approach.*
Addison-Wesley, Reading, MA, 1999.
[FP97]  
N.E. Fenton and S.L. Pfleeger.  
*Software Metrics – A Rigorous & Practical Approach (Second Edition).*  

[FW83]  
P. Freeman and A.I. Wasserman.  

[GHJV95]  
E. Gamma, R. Helm, R. Johnson, and J. Vlissides.  
*Design Patterns – Elements of Reusable Object-Oriented Software.*  

[Hut94]  
A.T.F. Hutt.  
Object Analysis and Design – Comparison of Methods.  
Object Analysis and Design – Description of Methods.  

[IEE88]  
IEEE.  
IEEE Standard Dictionary of Measures to Produce Reliable Software.  

[IEE88b]  
IEEE.  
IEEE Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.  

[IEE91]  
IEEE.  

[IEE98]  
IEEE.  
IEEE Recommended Practice for Software Design Descriptions.  

[IEE98b]  
IEEE.  

[ISO91]  
ISO/IEC.  
Information technology – Software product evaluation – Quality characteristics and guidelines for their use.  

[ISO95]  
ISO/IEC.  
Open distributed processing – Reference model.  
[ISO95b]
ISO/IEC.
Information technology – Software life cycle processes.

[Jal97]
P. Jalote.
An integrated approach to software engineering, 2nd ed.

[JB+91]
Rumbaugh J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen.
Object-Oriented Modeling and Design.

[JBR99]
I. Jacobson, G. Booch, and J. Rumbaugh.
The Unified Software Development Process.
Addison-Wesley, Reading, Ma, 1999.

[JCJO92]
Object-Oriented Software Engineering – A Use Case Driven Approach.
Addison-Wesley, 1992.

[Kru95]
P.B. Kruchten.
The 4+1 view model of architecture.

[Mar94]
J.J. Marciniak.
Encyclopedia of Software Engineering.

[Mey97]
B. Meyer.
Object-Oriented Software Construction (Second Edition).

[MNS96]
H.A. Muller, R.J. Norman, and J. Slonim (eds.).
Computer Aided Software Engineering.

[OMG98]
OMG.
The common object request broker: Architecture and specification.

[OMG99]
UML Revision Task Force.
OMG Unified Modeling Language specification, v. 1.3. document ad/99-06-08, Object Management Group,
June 1999.


