IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

IEEE Computer Society

Sponsored by the
Software & Systems Engineering Standards Committee

1016™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

20 July 2009

IEEE Std 1016™-2009
(Revision of
IEEE Std 1016-1998)

IEEE Std 1016™-2009
(Revision of
IEEE Std 1016-1998)

IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

Sponsor

Software & Systems Engineering Standards Committee
of the
IEEE Computer Society

Approved 19 March 2009

IEEE-SA Standards Board

Abstract: The required information content and organization for software design descriptions (SDDs) are described. An SDD is a representation of a software design to be used for communicating design information to its stakeholders. The requirements for the design languages (notations and other representational schemes) to be used for conformant SDDs are specified. This standard is applicable to automated databases and design description languages but can be used for paper documents and other means of descriptions.
Keywords: design concern, design subject, design view, design viewpoint, diagram, software design, software design description

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-0-7381-5925-6 STD95917
Print: ISBN 978-0-7381-5926-3 STDPD95917

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation, or every ten years for stabilization. When a document is more than five years old and has not been reaffirmed, or more than ten years old and has not been stabilized, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Recommendations to change the status of a stabilized standard should include a rationale as to why a revision or withdrawal is required. Comments and recommendations on standards, and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Introduction

This introduction is not part of IEEE Std 1016-2009, IEEE Standard for Information Technology—Systems Design—Software Design Descriptions.

This standard specifies requirements on the information content and organization for software design descriptions (SDDs). An SDD is a representation of a software design that is to be used for recording design information, addressing various design concerns, and communicating that information to the design's stakeholders.

SDDs play a pivotal role in the development and maintenance of software systems. During its lifetime, an SDD is used by acquirers, project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers. Each of these stakeholders has unique needs, both in terms of required design information and optimal organization of that information. Hence, a design description contains the design information needed by those stakeholders.

The standard also specifies requirements on the design languages to be used when producing SDDs conforming to these requirements on content and organization.

The standard specifies that an SDD be organized into a number of design views. Each view addresses a specific set of design concerns of the stakeholders. Each design view is prescribed by a design viewpoint. A viewpoint identifies the design concerns to be focused upon within its view and selects the design languages used to record that design view. The standard establishes a common set of viewpoints for design views, as a starting point for the preparation of an SDD, and a generic capability for defining new design viewpoints thereby expanding the expressiveness of an SDD for its stakeholders.

This standard is intended for use in design situations in which an explicit SDD is to be prepared. These situations include traditional software design and construction activities leading to an implementation as well as "reverse engineering" situations where a design description is to be recovered from an existing implementation.

This standard can be applied to commercial, scientific, military, and other types of software. Applicability is not restricted by size, complexity, or criticality of the software. This standard considers both the software and its system context, including the developmental and operational environment. It can be used where software comprises the system or where software is part of a larger system characterized by hardware, software, and human components and their interfaces.

This standard is applicable whether the SDD is captured using paper documents, automated databases, software development tools, or other media. This standard does not explicitly support, nor is it limited to, use with any particular software design methodology or particular design languages, although it establishes minimum requirements on the selection of those design languages.

This standard can be used with IEEE Std 12207™-2008 [B21];[a] it can also be used in other life cycle contexts.

This standard consists of five clauses, as follows:

Clause 1 defines the scope and purpose of the standard.

Clause 2 provides definitions of terms used within the context of the standard.

[a] The numbers in brackets correspond to those of the Bibliography in Annex A.

Clause 3 provides a framework for understanding SDDs in the context of their preparation and use.

Clause 4 describes the required content and organization of an SDD.

Clause 5 defines several design viewpoints for use in producing SDDs.

Annex A provides a bibliography.

Annex B defines how a design language to be used in an SDD is to be described in a uniform manner.

Annex C contains a template for organizing an SDD conforming to the requirements of this standard.

This standard follows the *IEEE Standards Style Manual*.[b] In particular, the word *shall* identifies requirements that must be satisfied in order to claim conformance with this standard. The verb *should* identifies recommendations, and the verb *may* is used to denote that particular courses of action are permissible.

This revision of the standard is modeled after IEEE Std 1471™-2000 [B20], extending the concepts of view, viewpoint, stakeholder, and concern from that standard to support high-level and detailed design and construction for software. The demarcation between architecture, high-level and detailed design varies from system to system and is beyond the scope of this standard.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

[b] The *IEEE Standards Style Manual* can be found at http://standards.ieee.org/guides/style/index.html.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the Software Design Descriptions Working Group had the following membership:

Keith R. Middleton, *Chair (acting)*
Vladan V. Jovanovic, *Chair (emeritus)*
Basil A. Sherlund, *Chair (emeritus)*
Rich Hilliard, *Secretary and Technical Editor*

William Bartholomew
Edward Byrne
Bob Cook
Ed Corlett
Philippe Kruchten
Kathy Land
James Moore
Ira Sachs

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Butch Anton
Angela Anuszewski
Chris Bagge
Pieter Botman
Juan Carreon
Lawrence Catchpole
Danila Chernetsov
Keith Chow
S. Claassen
Daniel Conte
David Cornejo
Geoffrey Darnton
Terry Dietz
Antonio Doria
Timothy Ehrler
Kameshwar Eranki
Andre Fournier
Eva Freund
David Friscia
Lewis Gray
Michael Grimley
Randall Groves
John Harauz
Mark Henley
Rutger A. Heunks
Ian Hilliard
Werner Hoelzl
Peter Hung
Atsushi Ito
Mark Jaeger
Piotr Karocki
Dwayne Knirk
George Kyle
Susan Land
Dewitt Latimer
David J. Leciston
Daniel Lindberg
Vincent Lipsio
Edward Mccall
Keith R. Middleton
William Milam
James Moore
Rajesh Murthy
Michael S. Newman
William Petit
Ulrich Pohl
Robert Robinson
Randall Safier
James Sanders
Bartien Sayogo
Robert Schaaf
David J. Schultz
Stephen Schwarm
Raymond Senechal
Luca Spotorno
Thomas Starai
Walter Struppler
Gerald Stueve
Marcy Stutzman
K. Subrahmanyam
Richard Thayer
John Thywissen
Thomas Tullia
Vincent Tume
John Walz
P. Wolfgang
Forrest Wright
Janusz Zalewski
Wenhao Zhu

When the IEEE-SA Standards Board approved this standard on 19 March 2009, it had the following membership:

Robert M. Grow, *Chair*
Thomas Prevost, *Vice Chair*
Steve M. Mills, *Past Chair*
Judith Gorman, *Secretary*

John Barr
Karen Bartleson
Victor Berman
Ted Burse
Richard DeBlasio
Andy Drozd
Mark Epstein
Alexander Gelman
Jim Hughes
Rich Hulett
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David Law
Ted Olsen
Glenn Parsons
Ron Petersen
Narayanan Ramachandran
Jon Rosdahl
Sam Sciacca

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Howard Wolfman, *TAB Representative*
Michael Janezic, *NIST Representative*
Satish Aggarwal, *NRC Representative*

Lisa Perry
IEEE Standards Program Manager, Document Development

Malia Zaman
IEEE Standards Program Manager, Technical Program Development

Contents

IEEE Standard for Information Technology—Systems Design—Software Design Descriptions

1. Overview

1.1 Scope

This standard describes software designs and establishes the information content and organization of a software design description (SDD). An SDD is a representation of a software design to be used for recording design information and communicating that design information to key design stakeholders. This standard is intended for use in design situations in which an explicit SDD is to be prepared. These situations include traditional software construction activities, when design leads to code, and "reverse engineering" situations when a design description is recovered from an existing implementation.

This standard can be applied to commercial, scientific, or military software that runs on digital computers. Applicability is not restricted by the size, complexity, or criticality of the software. This standard can be applied to the description of high-level and detailed designs.

This standard does not prescribe specific methodologies for design, configuration management, or quality assurance. This standard does not require the use of any particular design languages, but establishes requirements on the selection of design languages for use in an SDD. This standard can be applied to the preparation of SDDs captured as paper documents, automated databases, software development tools, or other media.

1.2 Purpose

This standard specifies requirements on the information content and organization of SDDs. The standard specifies requirements for the selection of design languages to be used for SDD, and requirements for documenting design viewpoints to be used in organizing an SDD.

1.3 Intended audience

This standard is intended for technical and managerial stakeholders who prepare and use SDDs. It guides a designer in the selection, organization, and presentation of design information. For an organization developing its own SDD practices, the use of this standard can help to ensure that design descriptions are complete, concise, consistent, interchangeable, appropriate for recording design experiences and lessons learned, well organized, and easy to communicate.

1.4 Conformance

An SDD conforms to this standard if it satisfies all of the requirements in Clause 4 and Clause 5 of this standard. Requirements are denoted by the verb *shall*.

2. Definitions

For the purposes of this standard, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B13][3] and IEEE Std 12207™-2008 [B21] should be referenced for terms not defined in this clause.

3.1 design attribute: An element of a design view that names a characteristic or property of a design entity, design relationship, or design constraint. *See also:* **design constraint, design entity**, **design relationship**.

3.2 design concern: An area of interest with respect to a software design.

3.3 design constraint: An element of a design view that names and specifies a rule or restriction on a design entity, design attribute, or design relationship. *See also:* **design attribute, design entity**, **design relationship**.

3.4 design element: An item occurring in a design view that may be any of the following: design entity, design relationship, design attribute, or design constraint.

3.5 design entity: An element of a design view that is structurally, functionally, or otherwise distinct from other elements, or plays a different role relative to other design entities. *See also:* **design view**.

3.6 design overlay: A representation of additional, detailed, or derived design information organized with reference to an existing design view.

3.7 design rationale: Information capturing the reasoning of the designer that led to the system as designed, including design options, trade-offs considered, decisions made, and the justifications of those decisions.

[3] The numbers in brackets correspond to those of the Bibliography in Annex A.

3.8 design relationship: Element of a design view that names a connection or correspondence between design entities. *See also:* **design entity**.

3.9 design stakeholder: An individual, organization, or group (or classes thereof playing the same role) having an interest in, or design concerns relative to, the design of some software item. *See also:* **design concern**.

3.10 design subject: A software item or system for which an SDD will be prepared. *Syn:* **software under design**, **system under design**.

3.11 designer: The stakeholder responsible for devising and documenting the software design.

3.12 design view: A representation comprised of one or more design elements to address a set of design concerns from a specified design viewpoint. *See also:* **design concern, design element, design viewpoint.**

3.13 design viewpoint: The specification of the elements and conventions available for constructing and using a design view. *See also:* **design view**.

3.14 diagram (type): A logically coherent fragment of a design view, using selected graphical icons and conventions for visual representation from an associated design language, to be used for representing selected design elements of interest for a system under design from a single viewpoint *See also:* **design subject**.

3. Conceptual model for software design descriptions

This clause establishes a conceptual model for SDDs. The conceptual model includes basic terms and concepts of SDD, the context in which SDDs are prepared and used, the stakeholders who use them, and how they are used.

3.1 Software design in context

A design is a conceptualization of a design subject (the system under design or software under design) that embodies its essential characteristics; demonstrates a means to fulfill its requirements; serves as a basis for analysis and evaluation and can be used to guide its implementation. (See Abran and Moore [B1] for further discussion.)

NOTE 1—This standard does not establish what a design subject may be. A design subject can be any software item to be constructed or which already exists and is to be analyzed. Examples of possible design subjects include, but are not limited to, systems, subsystems, applications, components, libraries, application frameworks, application program interfaces (APIs), and design pattern catalogs.[4]

An SDD is a work product depicting some design subject of interest. An SDD can be produced to capture one or more levels of concern with respect to its design subject. These levels are usually determined by the design methods in use or the life cycle context; they have names such as “architectural design,” “logical design,” or “physical design.”

An SDD can be prepared and used in a variety of design situations. Typically, an SDD is prepared to support the development of a software item to solve a problem, where this problem has been expressed in terms of a set of requirements. The contents of the SDD can then be traced to these requirements. In other

[4] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

cases, an SDD is prepared to understand an existing system lacking design documentation. In such cases, an SDD is prepared such that information of interest is captured, organized, presented and disseminated to all interested parties. This information of interest can be used for planning, analysis, implementation and evolution of the software system, by identifying and addressing essential design concerns.

NOTE 2—The generic term *software design description* is used in this standard (1) to retain compatibility with the terminology of its predecessor, IEEE Std 1016-1998, and (2) to refer to a range of work products typically defined in use. For example, software design description covers the following information items identified in ISO/IEC 15289:2006 [B25]:[5] database design description (10.14), database detailed design description (10.15), high-level software design description (10.22), interface description (10.27), low-level software design description (10.29), system description (10.71), and system element description (10.72).

A design concern names any area of interest in the design, pertaining to its development, implementation, or operation. Design concerns are expressed by design stakeholders. Frequently, design concerns arise from specific requirements on the software, others arise from contextual constraints. Typical design concerns include functionality, reliability, performance, and maintainability. Typical design stakeholders include users, developers, software designers, system integrators, maintainers, acquirers, and project managers.

NOTE 3—From a system-theoretic standpoint, an SDD captures the information content of the design space with convenient inputs (design diagrams and specifications produced by designers) and outputs (results of transformations produced by software tools). The design space typically contains alternative designs and design rationale in addition to the minimal information of the current version of design. An interesting property of a design description as a system is that its configuration is subject to dynamic evolution and the respective state space, based on its design elements, is not given in advance but created iteratively in a manner of system analysis by synthesis. The final design synthesis is obtained via successive analysis of intermediate designs. Therefore, an SDD can be considered an open, goal-directed system whose end state is a detailed model of the system under design.

An SDD is organized using design views. A design view addresses one or more of the design concerns.

NOTE 4—The use of multiple views to achieve separation of concerns has a long history in software engineering (see Ross, Goodenough, and Irvine [B33] and Ross [B31]), recently in viewpoint-oriented requirements engineering (see Nuseibeh, Kramer, and Finkelstein [B27]), and particularly relevant to this standard, the use of views to rationally present the results of a design process (see Parnas and Clements [B30]) and their use during design (see Gomma [B11]). The particular formulation here is derived from IEEE Std 1471™-2000 [B20].

Each design view is governed by a design viewpoint. Each design viewpoint focuses on a set of the design concerns and introduces a set of descriptive resources called *design elements* that are used to construct and interpret the design view.

Example:

A viewpoint can introduce familiar design elements such as functions, input, and outputs; these elements are used to construct a functional view.

There are four kinds of design elements: design entities, design relationships, design attributes, and design constraints. A design viewpoint determines the types of design elements to be used in any design views it governs. Each design view is expressed as a collection of instances of design entities, design attributes, design relationships among design entities, and design constraints on those elements. The design information needs of stakeholders of the system under design are to be satisfied through use of these elements.

NOTE 5—Although a view need not be a graph, its content is frequently described using diagrams that may be formalized as extensions or specializations of conceptual graphs of design elements (see ISO draft *Conceptual Graphs* [B22]).

[5] Expressions in parentheses in the remainder of this NOTE refer to the subclauses in ISO/IEC 15289:2006 [B25], where these information items are defined.

It is sometimes useful to gather and present information that does not strictly follow the partitioning of information by design viewpoints. A design overlay is a mechanism to organize and present such additional design information.

Design involves the consideration and evaluation of alternatives and trade-offs among alternatives leading to decisions (see Abran and Moore [B1]). Design decisions and the design rationale for decisions are captured both to aid the understanding of current design stakeholders and to support future decision-making (see IEEE Std 12207-2008 [B21]).

The key concepts of SDD are depicted in Figure 1a and Figure 1b.

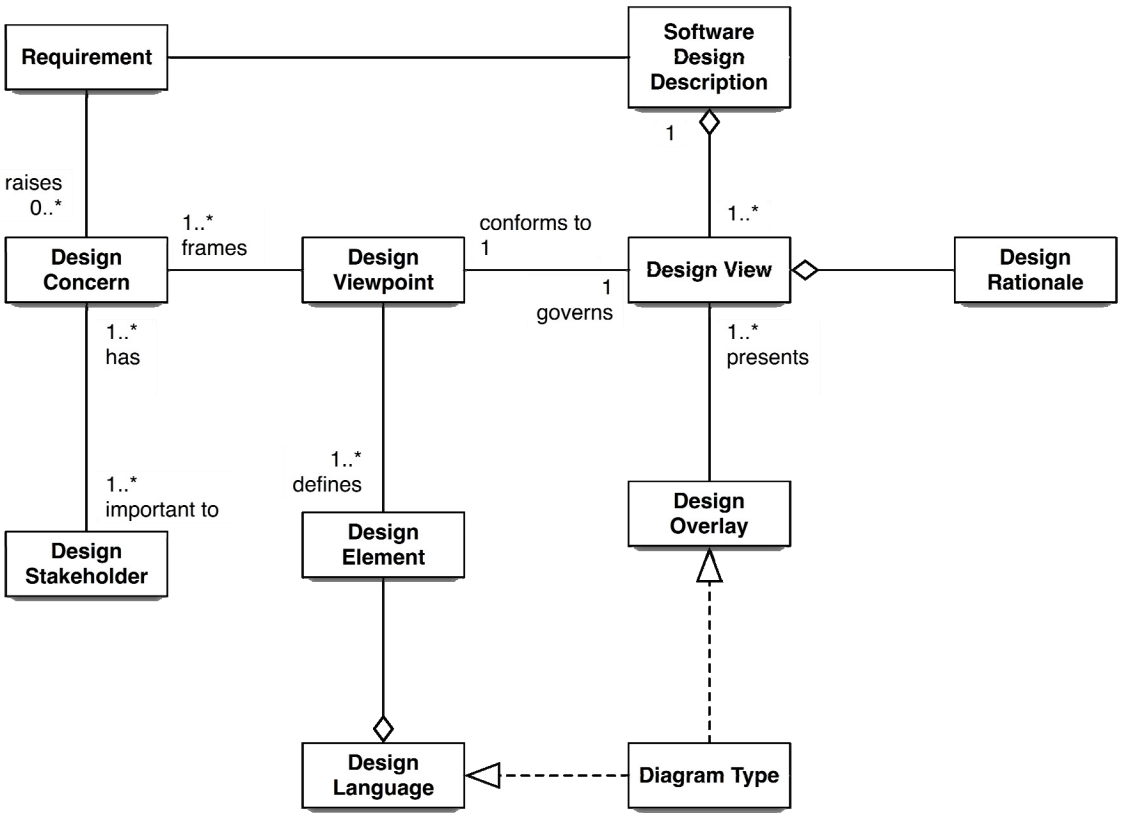

Figure 1a—Conceptual model: top view

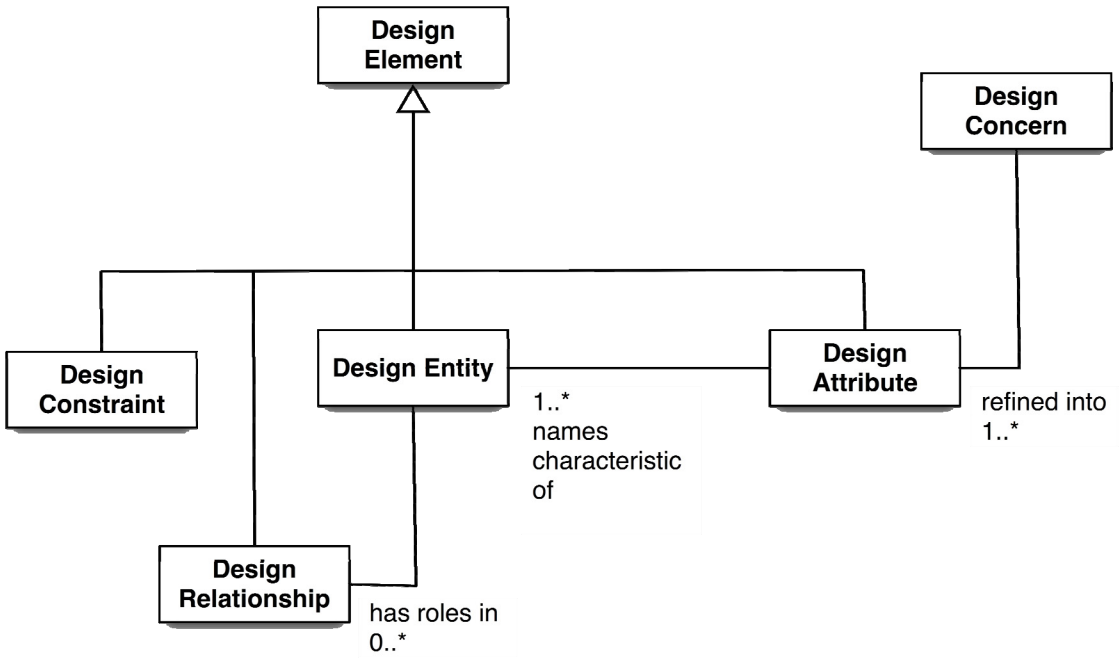

Figure 1b—Conceptual model: design elements

NOTE 6—Figure 1a and Figure 1b provide a summary of the key concepts used in this standard and their relationships. The figure presents these concepts and their relationships from the perspective of a single SDD used to depict a single design subject. An SDD includes one or more design views employing design viewpoints selected to cover stakeholders' design concerns raised by requirements. The contents of each design view is expressed in terms of design entities, their design attributes and design relationships, using selected design languages. In the figure, boxes represent classes of things. Lines connecting boxes represent associations between classes of things. Each class participating in an association has a role in that association. A role is optionally named with a label appearing close to where the association is attached to the class. For example, in the association between Design Viewpoint and Design Element, the role of Design Element is labeled *defines*. Each role may have a multiplicity, denoting a number or set of numbers such as a numeric range. A diamond at the end of an association denotes a part-of relationship. For example, the association between Design View and Software Design Description is interpreted as "one or more Design Views are part of a Software Design Description." This notation is defined in OMG formal/2007-11-2 and 2007-11-4, Unified Modeling Language [B28], [B29].

3.2 Software design descriptions within the life cycle

In this standard, a typical cycle will be used to describe the various design situations in which an SDD can be created and used. This life cycle is based on IEEE Std 12207-2008 [B21].

3.2.1 Influences on SDD preparation

The key software life cycle product that drives a software design is typically the software requirements specification (SRS). An SRS captures the software requirements that will drive the design and design constraints to be considered or observed (see IEEE Std 830™-1998 [B15] and ISO/IEC 15289:2006 [B25]).

3.2.2 influences on software life cycle products

The SDD influences the content of several major software life cycle work products. Developers of these products will be recognized among the SDD's intended audience.

— *Software requirements specification.* Design decisions, or design constraints discovered during the preparation of the SDD, can lead to requirements changes. Maintaining traceability between requirements and design is one way to record the relationship between the two (see Abran and Moore [B1]).

— *Test documentation.* Test plans can be influenced by the SDD. The development of test cases and test procedures can take into consideration the contents of the SDD.

3.2.3 Design verification and design role in validation

Verification is the determination whether a software work product fulfills specified requirements (see IEEE Std 12207-2008 [B21]). An SDD is subject to design verification to ascertain whether the design: addresses stakeholders' design concerns; is consistent with stated requirements; implements intended design decisions (such as those pertaining to interfaces, inputs, outputs, algorithms, resource allocation, and error handling); achieves intended qualities (such as safety, security, or maintainability); and conforms to an imposed architecture. Verification therefore can introduce design concerns to be dealt with in an SDD. Its design verification will usually be performed via review, inspection, or analysis (see IEEE Std 1028™-2008 [B17]).

Validation is the determination that the requirements for a specific intended use of a software work product are fulfilled (see IEEE Std 12207-2008 [B21]). The SDD plays a role in validation by providing: an overview necessary for understanding the implementation; the rationale justifying design decisions made; and traceability back to the requirements on the software under design. (See IEEE Std 1012™-2004 [B16] on verification and validation.)

4. Design description information content

4.1 Introduction

The required contents of an SDD are as follows:

— Identification of the SDD

— Identified design stakeholders

— Identified design concerns

— Selected design viewpoints, each with type definitions of its allowed design elements and design languages

— Design views

— Design overlays

— Design rationale

These are described in the remainder of this clause.

4.2 SDD identification

An SDD shall include the following descriptive information:

- Date of issue and status
- Scope
- Issuing organization
- Authorship (responsibility or copyright information)
- References
- Context
- One or more design languages for each design viewpoint used
- Body
- Summary
- Glossary
- Change history

NOTE—This requirement enables an SDD to meet the guidelines on a description in 7.3 of ISO/IEC 15289:2006 [B25]. The description of design languages is a proper part of the design viewpoint declarations (per 4.5). The body of the SDD is organized into design views (per 4.4), possibly with associated overlays (per 4.7) and design rationale (per 4.8).

4.3 Design stakeholders and their concerns

An SDD shall identify the design stakeholders for the design subject.

An SDD shall identify the design concerns of each identified design stakeholder.

An SDD shall address each identified design concern.

NOTE—An SDD can be used to satisfy the content guidelines for several types of design description as defined in ISO/IEC 15289:2006 [B25], by identifying their content guidelines as design concerns. The types of design descriptions are as follows: database design description (10.14),[6] database detailed design description (10.15), high-level software design description (10.22), interface description (10.27), low-level software design description (10.29), system description (10.71), and system element description (10.72).

4.4 Design views

An SDD shall be organized into one or more design views.

Each design view in an SDD shall conform to its governing design viewpoint.

The purpose of a design view is to address design concerns pertaining to the design subject, to allow a design stakeholder to focus on design details from a specific perspective and effectively address relevant requirements.

[6] Expressions in parentheses in the remainder of this NOTE refer to the subclauses in ISO/IEC 15289:2006 [B25], where the content guidelines for each information item are defined.

Each design view shall address the design concerns specified by its governing design viewpoint.

An SDD is *complete* when each identified design concern is the topic of at least one design view; all design attributes refined from each design concern by some viewpoint have been specified for all of the design entities and relationships in its associated view; and all design constraints have been applied.

An SDD is *consistent* if there are no known conflicts between the design elements of its design views.

NOTE—Users of this standard may wish to state delivery requirements on an SDD in terms of the above notions of completeness and consistency.

4.5 Design viewpoints

For each design view in an SDD, there shall be exactly one design viewpoint governing it.

Each design viewpoint shall be specified by:

- Viewpoint name;
- Design concerns that are the topics of the viewpoint;
- Design elements, defined by that viewpoint, specifically the types of design entities, attributes, relationships, and constraints introduced by that viewpoint or used by that viewpoint (which may have been defined elsewhere). These elements may be realized by one or more design languages;
- Analytical methods or other operations to be used in constructing a design view based upon the viewpoint, and criteria for interpreting and evaluating the design; and
- Viewpoint source (e.g., authorship or citation), when applicable.

In addition, a design viewpoint specification may provide the following information on using the viewpoint:

- Formal or informal consistency and completeness tests to be applied to the view;
- Evaluation or analysis techniques to be applied to a view; and
- Heuristics, patterns, or other guidelines to assist in construction or synthesis of a view.

An SDD shall include a rationale for the selection of each selected viewpoint.

Each design concern identified in an SDD (4.3) shall be framed by at least one design viewpoint selected for use. A design concern may be the focus of more than one viewpoint in an SDD.

NOTE—A design viewpoint specification may be included in the SDD or incorporated by reference.

4.6 Design elements

A design element is any item occurring in a design view. A design element may be any of the following subcases: design entity, design relationship, design attribute, or design constraint.

Each design element in the SDD shall have a name (4.6.2.1), a type (4.6.2.2), and any contents.

NOTE 1—This requirement is “inherited” by the four subcases: design entities, design relationships, design attributes, and design constraints.

The type of each design element shall be introduced within exactly one design viewpoint definition.

A design element may be used in one or more design views.

NOTE 2—A design element is introduced and "owned" by exactly one design view, in accordance with its type definition within the associated viewpoint. It may be shared or referenced within other design views. Sharing of design elements permits the expression of design aspects as in aspect-oriented design.

4.6.1 Design entities

Design entities capture key elements of a software design.

Each design entity shall have a name (4.6.2.1), a type (4.6.2.2), and purpose (4.6.2.3).

Examples of design entities include, but are not limited to, the following: systems, subsystems, libraries, frameworks, abstract collaboration patterns, generic templates, components, classes, data stores, modules, program units, programs, and processes.

NOTE—The number and types of entities needed to express a design view are dependent on a number of factors, such as the complexity of the system, the design technique used, and the tool support environment.

4.6.2 Design attributes

A design attribute names a characteristic or property of a design element (which may be a design entity, design constraint, or a design relationship) and provides a statement of fact about that design element.

All design attributes declared by a design viewpoint shall be specified.

NOTE 1—Design attributes can be thought of as questions about design elements. The answers to those questions are the values of the attributes. All the questions can be answered, but the content of the answer will depend upon the nature of the entity. The collection of answers provides a complete description of an entity. Attribute descriptions should include references and design considerations such as tradeoffs and assumptions when appropriate. In some cases, attribute descriptions may have the value *none*.

NOTE 2—Design attributes have been generalized from the concept of *design entity attribute* (which appeared in IEEE Std 1016-1998 and applied only to design entities) to apply to design entities, design relationships, and design constraints.

NOTE 3—Use of the design attributes in 4.6.2.1 through 4.6.2.3 ensures compatibility with IEEE Std 1016-1998. Other design attributes required as a part of specific design viewpoints are defined with those viewpoints in Clause 5. Some design attributes [such as subordinates (see 5.3.2.2)] can be more usefully represented as design relationships. This was not possible in IEEE Std 1016-1998.

4.6.2.1 Name attribute

The name of the element. Each design element shall have an unambiguous reference name. The names for the elements may be selected to characterize their nature. This will simplify referencing and tracking in addition to providing identification.

4.6.2.2 Type attribute

A description of the kind of element. The type attribute shall describe the nature of the element. It may simply name the kind of element, such as subsystem, component, framework, library, class, subprogram,

module, program unit, function, procedure, process, object, persistent object, class, or data store. Alternatively, design elements may be grouped into major classes to assist in locating an element dealing with a particular type of information.

Within an SDD, the chosen element types shall be applied consistently.

4.6.2.3 Purpose attribute

A description of why the element exists. The purpose attribute shall provide the rationale for the creation of the element.

4.6.2.4 Author attribute

Identification of designer. The author attribute shall identify the designer of the element.

4.6.3 Design relationships

A design relationship names an association or correspondence among two or more design entities. It provides a statement of fact about those design entities.

Each design relationship in an SDD shall have a name (4.6.2.1) and a type (4.6.2.2). A design relationship shall identify the design entities participating in the relationship.

NOTE—There are no design relationships defined in this standard. Most design techniques use design relationships extensively. Normally these design relationships will be defined as a part of a design viewpoint. For example, structured design methods are built around design relationships including input (datum *I* is an **input** to process *A*), output (datum *O* is an **output** of process *A*) and decompose (process *A* **decomposes** into processes *A1*, *A2,* and *A3*) relationships. Object-oriented design methods use design relationships including encapsulation, generalization, specialization, composition, aggregation, realization, and instantiation.

4.6.4 Design constraints

A design constraint is an element of a design view that names a rule or restriction imposed by one design element (the *source*) upon another design element (the *target*), which may be a design entity, design attribute, or design relationship.

Each design constraint in an SDD shall have a name (4.6.2.1) and a type (4.6.2.2). A design constraint shall identify its source and target design entities.

NOTE—There are no design constraints predefined in this standard. Many design techniques introduce design constraints.

4.7 Design overlays

A design overlay is used for presenting additional information with respect to an already-defined design view.

Each design overlay shall be uniquely named and marked as an overlay.

Each design overlay shall be clearly associated with a single viewpoint.

NOTE—Reasons to utilize a design overlay as a part of an SDD include: to provide an extension mechanism for design information to be presented conveniently on top of some view without a requirement for existing external standardization of languages and notations for such representation; to extend expressive power of representation with additional details while reusing information from existing views (i.e., without a need to define additional views or persistently store derivable design information); and to relate design information with facts from the system environment for the convenience of the designer (or other stakeholders).

4.8 Design rationale

Design rationale captures the reasoning of the designer that led to the system as designed and the justifications of those decisions.

Design rationale may take the form of commentary, made throughout the decision process and associated with collections of design elements. Design rationale may include, but is not limited to: design issues raised and addressed in response to design concerns; design options considered; trade-offs evaluated; decisions made; criteria used to guide design decisions; and arguments and justifications made to reach decisions.

NOTE—The only required design rationale is use of the purpose attribute (4.6.2.3).

4.9 Design languages

Design languages are selected as a part of design viewpoint specification (4.5).

A design language may be selected for a design viewpoint only if it supports all design elements defined by that viewpoint.

Design languages shall be selected that have:

- A well-defined syntax and semantics; and
- The status of an available standard or equivalent defining document.

Only standardized and well-established (i.e., previously defined and conveniently available) design languages shall be used in an SDD. In the case of a newly invented design language, the language definition must be provided as a part of the viewpoint declaration.

NOTE 1—Standardized design languages that are in common use are preferable to established languages without a formal definition. Examples of standardized languages include: IDEF0 (IEEE Std 1320.1™-1998 [B18]); IDEF1X (IEEE Std 1320.2™-1998 [B19]); Unified Modeling Language (UML) (OMG [B28] and [B29]); Vienna Definition Method (VDM) (ISO/IEC 13817-1:1996 [B24]); and Z (ISO/IEC 13568:2002 [B23]). Examples of established languages include: state machines, automata, decision tables, Warnier diagrams, Jackson Structured Design (JSD), program design languages (PDL), structure charts, Hierarchy plus Input-Process-Output (HIPO), reliability models, and queuing models.

NOTE 2—It is acceptable to use a design language in more than one view. It is also acceptable to use more than one design language within any number of views when each design language to be used is declared by the viewpoint. This is acceptable even for a portion of the design; for example, when used as a basis for interchange; due to organizational considerations such as development by non-collocated team members; subcontracting of partial design responsibility; or taking advantage of particular design tools or designer expertise.

NOTE 3—Annex B establishes a uniform format for describing design languages to be used in SDDs.

NOTE 4—In case that no adequate design language is readily available for a specified viewpoint, it is the designer's responsibility to provide an adaptation of an existing language or the definition of an appropriate new design language. This design language definition would be provided by the designer to be included in the SDD in accordance with the requirements for viewpoints in 4.5.

5. Design viewpoints

5.1 Introduction

This clause defines several design viewpoints for use in SDDs. It illustrates the realization of these design viewpoints in terms of design language selections, relates design concerns with viewpoints, and establishes language (notation and method) neutral names for these viewpoints. Table 1 summarizes these design viewpoints. For each viewpoint, its name, design concerns, and appropriate design languages are listed. Short descriptions relating a minimal set of design entities, design relationships, design entity attributes, and design constraints are provided for each viewpoint. Additional references pertinent to the use of each viewpoint are also listed.

A design viewpoint defined in this clause shall be used in the SDD whenever applicable to the design subject, based on identified design concerns (4.3).

Table 1—Summary of design viewpoints

Design viewpoint	Design concerns	Example design languages
Context (5.2)	Systems services and users	IDEF0, UML use case diagram, Structured Analysis context diagram
Composition (5.3) Can be refined into new viewpoints, such as: functional (logical) decomposition, and run-time (physical) decomposition.	Composition and modular assembly of systems in terms of subsystems and (pluggable) components, buy vs. build, reuse of components	Logical: UML package diagram, UML component diagram, Architecture Description Languages, IDEF0, Structure chart, HIPO Physical: UML deployment diagram
Logical (5.4)	Static structure (classes, interfaces, and their relationships) Reuse of types and implementations (classes, data types)	UML class diagram, UML object diagram
Dependency (5.5)	Interconnection, sharing, and parameterization	UML package diagram and component diagram
Information (5.6) with data distribution overlay and physical volumetric overlay	Persistent information	IDEF1X, entity-relation diagram, UML class diagram
Patterns (5.7)	Reuse of patterns and available Framework template	UML composite structure diagram
Interface (5.8)	Service definition, service access	Interface definition languages (IDL), UML component diagram
Structure (5.9)	Internal constituents and organization of design subjects, components and classes	UML structure diagram, class diagram
Interaction (5.10)	Object communication, messaging	UML sequence diagram, UML communication diagram
State dynamics (5.11)	Dynamic state transformation	UML state machine diagram, statechart (Harel's), state transition table (matrix), automata, Petri net
Algorithm (5.12)	Procedural logic	Decision table, Warnier diagram, JSP, PDL
Resources (5.13) May be refined into resource based viewpoints with possible overlays	Resource utilization	UML Real-time Profile, UML class diagram, UML Object Constraint Language (OCL)

5.2 Context viewpoint

The Context viewpoint depicts services provided by a design subject with reference to an explicit context. That context is defined by reference to actors that include users and other stakeholders, which interact with the design subject in its environment. The Context viewpoint provides a "black box" perspective on the design subject.

Services depict an inherently functional aspect or anticipated cases of use of the design subject (hence "use cases" in UML). Stratification of services and their descriptions in the form of scenarios of actors' interactions with the system provide a mechanism for adding detail. Services may also be associated with actors through information flows. The content and manner of information exchange with the environment implies additional design information and the need for additional viewpoints (see 5.10).

A Deployment overlay to a Context view can be transformed into a Deployment view whenever the execution hardware platform is part of the design subject; for stand-alone software design, a Deployment overlay maps software entities onto externally available entities not subject of the current design effort. Similarly, work allocation to teams and other management perspectives are overlays in the design.

5.2.1 Design concerns

The purpose of the Context viewpoint is to identify a design subject's offered services, its actors (users and other interacting stakeholders), to establish the system boundary and to effectively delineate the design subject's scope of use and operation.

Drawing a boundary separating a design subject from its environment, determining a set of services to be provided, and the information flows between design subject and its environment, is typically a key design decision. That makes this viewpoint applicable to most design efforts.

When the system is portrayed as a black box, with internal decisions hidden, the Context view is often a starting point of design, showing what is to be designed functionally as the only available information about the design subject: a name and an associated set of externally identifiable services. Requirements analysis identifies these services with the specification of quality of service attributes, henceforth invoking many non-functional requirements. Frequently incomplete, a Context view is begun in requirements analysis. Work to complete this view continues during design.

5.2.2 Design elements

Design entities: actors—external active elements interacting with the design subject, including users, other stakeholders and external systems, or other items; services—also called *use cases;* and directed information flows between the design subject, treated as a black box, and its actors associating actors with services. Flows capture the expected information content exchanged.

Design relationships: receive output and provide input (between actors and the design subject).

Design constraints: qualities of service; form and medium of interaction (provided to and received from) with environment.

5.2.3 Example languages

Any black-box type diagrams can be used to realize the Context viewpoint. Appropriate languages include Structured Analysis [e.g., IDEF0 (IEEE Std 1320.1-1998 [B18]), Structured Analysis and Design

Technique (SADT) (Ross [B32]) or those of the DeMarco or Gane-Sarson variety], the Cleanroom's black box diagrams, and UML use cases (OMG [B28]).

5.3 Composition viewpoint

The Composition viewpoint describes the way the design subject is (recursively) structured into constituent parts and establishes the roles of those parts.

5.3.1 Design concerns

Software developers and maintainers use this viewpoint to identify the major design constituents of the design subject, to localize and allocate functionality, responsibilities, or other design roles to these constituents. In maintenance, it can be used to conduct impact analysis and localize the efforts of making changes. Reuse, on the level of existing subsystems and large-grained components, can be addressed as well. The information in a Composition view can be used by acquisition management and in project management for specification and assignment of work packages, and for planning, monitoring, and control of a software project. This information, together with other project information, can be used in estimating cost, staffing, and schedule for the development effort. Configuration management may use the information to establish the organization, tracking, and change management of emerging work products (see IEEE Std 12207-2008 [B21]).

5.3.2 Design elements

Design entities: types of constituents of a system: subsystems, components, modules; ports and (provided and required) interfaces; also libraries, frameworks, software repositories, catalogs, and templates.

Design relationships: composition, use, and generalization. The Composition viewpoint supports the recording of the part-whole relationships between design entities using realization, dependency, aggregation, composition, and generalization relationships. Additional design relationships are required and provided (interfaces), and the attachment of ports to components.

Design attributes: For each design entity, the viewpoint provides a reference to a detailed description via the identification attribute. The attribute descriptions for identification, type, purpose, function, and definition attribute should be utilized.

5.3.2.1 Function attribute

A statement of what the entity does. The function attribute states the transformation applied by the entity to its inputs to produce the output. In the case of a data entity, this attribute states the type of information stored or transmitted by the entity.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998.

5.3.2.2 Subordinates attribute

The identification of all entities composing this entity. The subordinates attribute identifies the "composed of" relationship for an entity. This information is used to trace requirements to design entities and to identify parent/child structural relationships through a design subject.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998. An equivalent capability is available through the composition relationship.

5.3.3 Example languages

UML component diagrams (see OMG [B28]) cover this viewpoint. The simplest graphical technique used to describe functional system decomposition is a hierarchical decomposition diagram; such diagram can be used together with natural language descriptions of purpose and function for each entity, such as is provided by IDEF0 (IEEE Std 1320.1-1998 [B18]), the Structure Chart (Yourdon and Constantine [B38], and the HIPO Diagram. Run-time composition can also use structured diagrams (Page-Jones [B29]).

5.4 Logical viewpoint

The purpose of the Logical viewpoint is to elaborate existing and designed types and their implementations as classes and interfaces with their structural static relationships. This viewpoint also uses examples of instances of types in outlining design ideas.

5.4.1 Design concerns

The Logical viewpoint is used to address the development and reuse of adequate abstractions and their implementations. For any implementation platform, a set of types is readily available for the domain abstractions of interest in a design subject, and a number of new types is to be designed, some of which may be considered for reuse. The main concern is the proper choice of abstractions and their expression in terms of existing types (some of which may had been specific to the design subject).

5.4.2 Design elements

Design entities: class, interface, power type, data type, object, attribute, method, association class, template, and namespace.

Design relationships: association, generalization, dependency, realization, implementation, instance of, composition, and aggregation.

Design attributes: name, role name, visibility, cardinality, type, stereotype, redefinition, tagged value, parameter, and navigation efficiency.

Design constraints: value constraints, relationships exclusivity constraints, navigability, generalization sets, multiplicity, derivation, changeability, initial value, qualifier, ordering, static, pre-condition, post-condition, and generalization set constraints.

5.4.3 Example languages

UML class diagrams and UML object diagrams (showing objects as instances of their respective classes) (OMG [B28]). Lattices of types and references to implemented types are commonly used as supplementary information.

5.5 Dependency viewpoint

The Dependency viewpoint specifies the relationships of interconnection and access among entities. These relationships include shared information, order of execution, or parameterization of interfaces.

5.5.1 Design concerns

A Dependency view provides an overall picture of the design subject in order to assess the impact of requirements or design changes. It can help maintainers to isolate entities causing system failures or resource bottlenecks. It can aid in producing the system integration plan by identifying the entities that are needed by other entities and that must be developed first. This description can also be used by integration testing to aid in the production of integration test cases.

5.5.2 Design elements

Design entities: subsystem, component, and module.

Design relationships: uses, provides, and requires.

Design attribute: name (4.6.2.1), type (4.6.2.2), purpose (4.6.2.3), dependencies (5.5.2.1), and resources. These attributes should be provided for all design entities.

5.5.2.1 Dependencies attribute

A description of the relationships of this entity with other entities. The dependencies attribute identifies the uses or requires the presence of relationship for an entity. This attribute is used to describe the nature of each interaction including such characteristics as timing and conditions for interaction. The interactions involve the initiation, order of execution, data sharing, creation, duplicating, usage, storage, or destruction of entities.

NOTE—This design entity attribute is retained for compatibility with IEEE Std 1016-1998.

5.5.3 Example languages

UML component diagrams and UML package diagrams showing dependencies among subsystems (OMG [B28]).

5.6 Information viewpoint

The Information viewpoint is applicable when there is a substantial persistent data content expected with the design subject.

5.6.1 Design concerns

Key concerns include persistent data structure, data content, data management strategies, data access schemes, and definition of metadata.

5.6.2 Design elements

Design entities: data items, data types and classes, data stores, and access mechanisms.

Design relationships: association, uses, implements. Data attributes, their constraints and static relationships among data entities, aggregates of attributes, and relationships.

Design attributes: persistence and quality properties.

5.6.2.1 Data attribute

A description of data elements internal to the entity. The data attribute describes the method of representation, initial values, use, semantics, format, and acceptable values of internal data. The description of data may be in the form of a data dictionary that describes the content, structure, and use of all data elements. Data information should describe everything pertaining to the use of data or internal data structures by this entity. It should include data specifications such as formats, number of elements, and initial values. It should also include the structures to be used for representing data such as file structures, arrays, stacks, queues, and memory partitions.

The meaning and use of data elements should be specified. This description includes such things as static versus dynamic, whether it is to be shared by transactions, used as a control parameter, or used as a value, loop iteration count, pointer, or link field. In addition, data information should include a description of data validation needed for the process.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998.

5.6.3 Example languages

IDEF1X (IEEE Std 1320.2™-1998 [B19]), UML class diagrams (OMG [B28]).

5.7 Patterns use viewpoint

This viewpoint addresses design ideas (emergent concepts) as collaboration patterns involving abstracted roles and connectors.

5.7.1 Design concerns

Key concerns include reuse at the level of design ideas (design patterns), architectural styles, and framework templates.

5.7.2 Design elements

Design entities: collaboration, class, connector, role, framework template, and pattern.

Design relationships: association, collaboration use, and connector.

Design attributes: name.

Design constraints: collaboration constraints.

5.7.3 Example languages

UML composite structure diagram and a combination of the UML class diagram and the UML package diagram (OMG [B28]).

5.8 Interface viewpoint

The Interface viewpoint provides information designers, programmers, and testers the means to know how to correctly use the services provided by a design subject. This description includes the details of external and internal interfaces not provided in the SRS. This viewpoint consists of a set of interface specifications for each entity.

NOTE—User interfaces are addressed separately.

5.8.1 Design concerns

An Interface view description serves as a binding contract among designers, programmers, customers, and testers. It provides them with an agreement needed before proceeding with the detailed design of entities. The interface description is used by technical writers to produce customer documentation or may be used directly by customers. In the latter case, the interface description could result in the production of a human interface view.

Designers, programmers, and testers often use design entities that they did not develop. These entities can be reused from earlier projects, contracted from an external source, or produced by other developers. The interface description establishes an agreement among designers, programmers, and testers about how cooperating entities will interact. Each entity interface description should contain everything another designer or programmer needs to know to develop software that interacts with that entity. A clear description of entity interfaces is essential on a multi-person development for smooth integration and ease of maintenance.

5.8.2 Design elements

The attributes for identification (4.6.2.1), function (5.3.2.1), and interface (5.8.2.1) should be provided for all design entities.

5.8.2.1 Interface attribute

A description of how other entities interact with this entity. The interface attribute describes the methods of interaction and the rules governing those interactions. Methods of interaction include the mechanisms for invoking or interrupting the entity, for communicating through parameters, common data areas or messages, and for direct access to internal data. The rules governing the interaction include the communications protocol, data format, acceptable values, and the meaning of each value.

This attribute provides a description of the input ranges, the meaning of inputs and outputs, the type and format of each input or output, and output error codes. For information systems, it should include inputs, screen formats, and a complete description of the interactive language.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998

5.8.3 Example languages

Interface definition languages (IDL), UML component diagram (OMG [B28]). In case of user interfaces the Interface view should include screen formats, valid inputs, and resulting outputs. For data-driven entities, a data dictionary should be used to describe the data characteristics. Those entities that are highly visible to a user and involve the details of how the customer should perceive the system should include a functional model, scenarios for use, detailed feature sets, and the interaction language.

5.9 Structure viewpoint

The Structure viewpoint is used to document the internal constituents and organization of the design subject in terms of like elements (recursively).

5.9.1 Design concerns

Compositional structure of coarse-grained components and reuse of fine-grained components.

5.9.2 Design elements

Design entities: port, connector, interface, part, and class.

Design relationships: connected, part of, enclosed, provided, and required.

Design attributes: name, type, purpose, and definition.

Design constraints: interface constraints, reusability constraints, and dependency constraints.

5.9.3 Example languages

UML composite structure diagram, UML class diagram, and UML package diagram (OMG [B28]).

5.10 Interaction viewpoint

The Interaction viewpoint defines strategies for interaction among entities, regarding why, where, how, and at what level actions occur.

5.10.1 Design concerns

For designers. this includes evaluating allocation of responsibilities in collaborations, especially when adapting and applying design patterns; discovery or description of interactions in terms of messages among affected objects in fulfilling required actions; and state transition logic and concurrency for reactive, interactive, distributed, real-time, and similar systems.

5.10.2 Design elements

Classes, methods. states, events, signals, hierarchy, concurrency, timing, and synchronization.

5.10.3 Examples

UML composite structure diagram, UML interaction diagram (OMG [B28]).

5.11 State dynamics viewpoint

Reactive systems and systems whose internal behavior is of interest use this viewpoint.

5.11.1 Design concerns

System dynamics including modes, states, transitions, and reactions to events.

5.11.2 Design elements

Design entities: event, condition, state, transition, activity, composite state, submachine state, critical region, and trigger.

Design relationships: part-of, internal, effect, entry, exit, and attached to.

Design attributes: name, completion, active, initial, and final.

Design constraints: guard conditions, concurrency, synchronization, state invariant, transition constraint, and protocol.

5.11.3 Example languages

UML state machine diagram (OMG [B28]), Harel statechart, state transition table (matrix), automata, Petri net.

5.12 Algorithm viewpoint

The detailed design description of operations (such as methods and functions), the internal details and logic of each design entity.

5.12.1 Design concerns

The Algorithm viewpoint provides details needed by programmers, analysts of algorithms in regard to time-space performance and processing logic prior to implementation, and to aid in producing unit test plans.

5.12.2 Design elements

These should include the attribute descriptions for identification, processing (5.12.1), and data for all design entities.

5.12.3 Processing attribute

A description of the rules used by the entity to achieve its function. The processing attribute describes the algorithm used by the entity to perform a specific task and its contingencies. This description is a refinement of the function attribute and is the most detailed level of refinement for the entity.

This description should include timing, sequencing of events or processes, prerequisites for process initiation, priority of events, processing level, actual process steps, path conditions, and loop back or loop termination criteria. The handling of contingencies should describe the action to be taken in the case of overflow conditions or in the case of a validation check failure.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998.

5.12.4 Examples

Decision tables and flowcharts; program design languages, "pseudo-code," and (actual) programming languages may also be used.

5.13 Resource viewpoint

The purpose of the Resource viewpoint is to model the characteristics and utilization of resources in a design subject.

5.13.1 Design concerns

Key concerns include resource utilization, resource contention, availability, and performance.

5.13.2 Design elements

Design entities: resources, usage policies.

Design relationships: allocation and uses.

Design attributes: identification (4.6.2.1), resource (5.13.2.1), performance measures (such as throughput, rate of consumption).

Design constraints: priorities, locks, resource constraints.

5.13.2.1 Resources attribute

A description of the elements used by the entity that are external to the design. The resources attribute identifies and describes all of the resources external to the design that are needed by this entity to perform its function. The interaction rules and methods for using the resource are to be specified by this attribute.

This attribute provides information about items such as physical devices (printers, disc-partitions, memory banks), software services (math libraries, operating system services), and processing resources (CPU cycles, memory allocation, buffers).

The resources attribute should describe usage characteristics such as the processing time at which resources are to be acquired and sizing to include quantity, and physical sizes of buffer usage. It should also include the identification of potential race and deadlock conditions as well as resource management facilities.

NOTE—This design attribute is retained for compatibility with IEEE Std 1016-1998.

5.13.3 Examples

Woodside [B37], UML class diagram, UML Object Constraint Language (OMG [B28]).

Annex A

(informative)

Bibliography

[B1] Abran, A., and J. W. Moore (editors), *Guide to the Software Engineering Body of Knowledge,* 2004 Edition. IEEE Press, 2005.

[B2] Arlow, J., and I. Neustadt, *Enterprise Patterns and MDA: Building better software with archetype patterns and UML.* Addison-Wesley, 2004.

[B3] Coplien, J., *Multi-Paradigm Design for C++.* Addison-Wesley, 1998.

[B4] D'Souza, D., and A. Wills, *Objects, Components, and Frameworks with UML.* Addison-Wesley 1999.

[B5] Douglass, B. P., *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns.* Addison-Wesley, 1999.

[B6] Douglass, B. P., *Real-Time UML.* Third edition, Addison-Wesley, 2004.

[B7] Evitts, P., A *UML Pattern Language.* Sams, 2000.

[B8] Fayad, M. E., and R. E. Johnson (editors), *Domain-Specific Application Frameworks.* Wiley, 2000.

[B9] Feldmann, C. G., *The Practical Guide to Business Process Reengineering Using IDEF0.* Dorset House Publishing, 1998.

[B10] Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[B11] Gomma, H., *Designing Software Product Lines with UML.* Addison-Wesley, 2005.

[B12] Graham, I., *Object-Oriented Methods.* Addison-Wesley, 2001.

[B13] IEEE 100™, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition. New York: Institute of Electrical and Electronics Engineers, Inc.[7, 8]

[B14] IEEE Std 610.12™-1990, IEEE Standard Glossary of Software Engineering Terminology.

[B15] IEEE Std 830™-1998, IEEE Recommended Practice for Software Requirements Specifications.

[B16] IEEE Std 1012™-2004, IEEE Standard for Software Verification and Validation.

[B17] IEEE Std 1028™-2008, IEEE Standard for Software Reviews and Audits.

[B18] IEEE Std 1320.1™-1998, IEEE Standard for Functional Modeling Language—Syntax and Semantics for IDEF0.

[B19] IEEE Std 1320.2™-1998, IEEE Standard Conceptual Modeling Language—Syntax and Semantics for IDEF1X97 (IDEFobject).

[B20] IEEE Std 1471™-2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.[9]

[B21] IEEE Std 12207™-2008 Systems and software engineering—Software life cycle processes.[10]

[7] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
[8] The IEEE standards or products referred to in Annex A are trademarks owned by the Institute of Electrical and Electronics Engineers, Incorporated.
[9] IEEE Std 1471-2000 is also known as ISO/IEC 42010:2007.
[10] IEEE Std 12207-2008 is also known as ISO/IEC 12207:2008.

[B22] ISO draft standard, Conceptual Graphs, 2001.[11]

[B23] ISO/IEC 13568:2002, Information technology—Z formal specification notation—Syntax, type system and semantics.[12]

[B24] ISO/IEC 13817–1:1996, Information technology—Programming languages, their environments and system software interfaces—Vienna Development Method—Specification Language—Part 1: Base language.

[B25] ISO/IEC 15289:2006, Systems and software engineering—Content of systems and software life cycle process information products (Documentation).

[B26] Kruchten, P., *The Rational Unified Process.* Addison-Wesley, 2000.

[B27] Nuseibeh, B., J. Kramer, and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Transactions on Software Engineering,* 20(10) pp. 760–773, 1994.

[B28] OMG formal/2007-11-02, Unified Modeling Language (OMG UML), Superstructure, version 2.1.2, November 2007.

[B29] OMG formal/2007-11-04, Unified Modeling Language (OMG UML), Infrastructure, version 2.1.2, November 2007.

[B30] Page-Jones, M., *The Practical Guide to Structured Systems Design,* Second Edition. Prentice Hall, 1988.

[B31] Parnas, D. L., and P. C. Clements, "A rational design process: how and why to fake it," *IEEE Transactions on Software Engineering,* 12(7), 1986.

[B32] Ross, D. T., "Structured Analysis: a language for communicating ideas," *IEEE Transactions on Software Engineering,* 1977.

[B33] Ross, D. T., J. B. Goodenough, and C.A. Irvine, "Software engineering: Process, principles, and goals," *COMPUTER* 8(5) (May 1975): 17–27

[B34] Rumbaugh, J., I. Jacobson, and G. Booch, *UML Reference Manual,* Second Edition, Addison-Wesley, 2005.

[B35] Shon, D., *The Reflective Practitioner.* Basic Books, 1983.

[B36] Weiss, D., and C. Lai, *Software Product Line Engineering.* Addison-Wesley, 1999.

[B37] Woodside, C. M., "A three-view model for performance engineering of concurrent software," *IEEE Transactions on Software Engineering,* 21(9), 1995.

[B38] Yourdon, E., and L. Constantine, *Structured Design.* Prentice Hall, 1979.

[11] Available at http://www.jfsowa.com/cg/cgstand.htm.

[12] ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

Annex B

(informative)

Conforming design language description

This annex defines a uniform format for describing design languages. Any design language can be documented in terms of a number of characteristics. These characteristics have been chosen to facilitate the selection of design languages in the selection and definition of viewpoints (Clause 4). The format has a structured textual format intended to allow both human and machine-readable application.

It is envisioned that the providers of design languages (whether commercial, industrial, research or experimental) will be able to document the intended usage of the design language using this format. This documentation will allow designers to more readily review the properties of that design language for use in an SDD because the attributes captured here match the considerations to be made when defining a design viewpoint in accordance with Clause 4 and Clause 5 of this standard.

Every well-formed design language description should specify the design language name.

Examples of design language names:

"IDEF0" "UML state machine"

Every well-formed design language description must have a reference to the definition of the design language. This may be a reference to a standard or other defining document.

Example of design language reference:

IEEE Std 1302.1, OMG-Unified Modeling Language, v1.4 September 2001

Every well-formed design language description should contain an identification of one or more design concerns that are capable of being expressed using this design language. This information can be used by designers to choose appropriate design languages to implement selected design viewpoints within an SDD.

Examples of design concerns:

Functionality, Reliability (for additional examples see Clause 5).

Every well-formed design language description must identify each design entity type defined by the design language.

Examples of design elements:

State, Transition, Event (for additional examples see Clause 5).

Every well-formed design language description must identify each design entity attribute type, and the design entity that define it.

Examples of design attributes:

transitionLabel *defined by*: Transition; guardCondition *defined by*: Transition (for additional examples see Clause 4 and Clause 5).

Every well-formed design language description must identify the design entity relationship types that are a part of the design language. First, the relationship type is named. Then the design entity types participating in the relationship are identified.

Examples of design relationships:

subActivities participants: 2 or more Activities (for additional examples see Clause 5).

Annex C

(informative)

Templates for an SDD

The template in Figure C.1 shows one possible way to organize and format an SDD conforming to the requirements of Clause 4.

Frontspiece
- Date of issue and status
- Issuing organization
- Authorship
- Change history

Introduction
- Purpose
- Scope
- Context
- Summary

References

Glossary

Body
- Identified stakeholders and design concerns
- Design viewpoint 1
- Design view 1
- …
- Design viewpoint n
- Design view n
- Design rationale

Figure C.1—Table of contents for an SDD