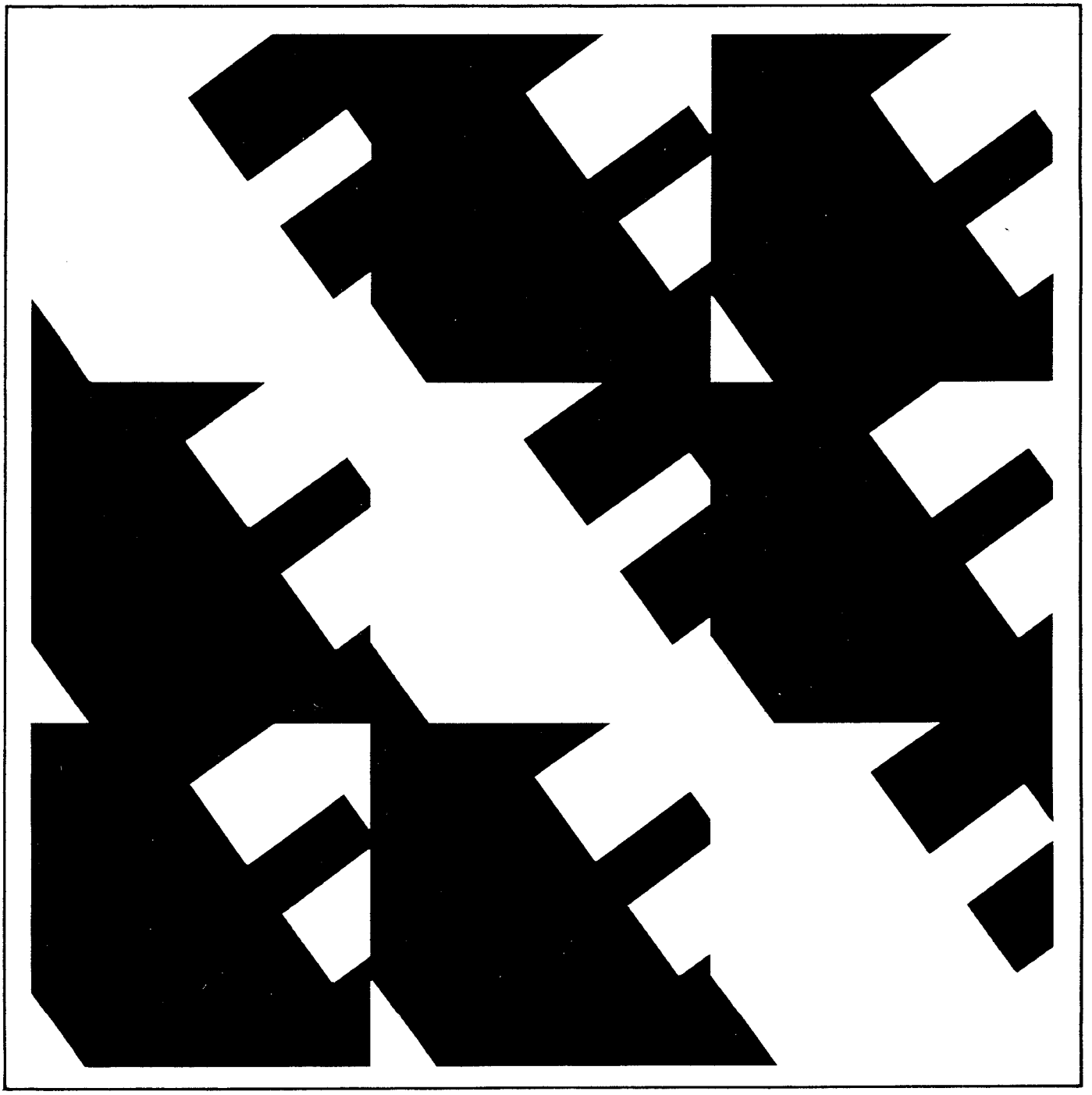
ANSI/IEEE Std 1016-1987
AMERICAN NATIONAL STANDARD
Approved October 6, 1987
American National Standards Institute

IEEE Recommended Practice for Software Design Descriptions



Published by The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017, USA
July 13, 1987 SH11031

Contents

David P. Linssen
Steven Lituintchouk
William Lively
John M. Long
John Lowell
Bill Macre
Andy Mahindru
Kartik C. Majumdar
Henry A. Malec
Paulo C. Marcondes
Stuart Marcotte
Gregory Mark
Phillip C. Mariott
Nicholas L. Marselos
Roger Martin
Paul A. Mauro
Jean I. Maxwell
L. J. Mazlack
Ivano Mazza
John McArdle
J. A. McCall
John McKissick, Jr
Glen A. Meldrum
Belden Menkus
Ben W. Miller
Charles S. Mooney
Gary D. Moorhead
Joyce E. Mortison
Gene T. Morun
David G. Mullens
Myron L. Nack
Hironobu Nagano
Saied Najafi
Gerry Neidhart
Brian A. Nejmeh
Dennis E. Nickle
J. T. L. Obbink
Wilma M. Osborne
D. J. Ostrom
Thomas D. Parrish
William Perry
Donald J. Pfeiffer
Robert M. Poston
Peter Ron Prinzivalli
James E. Quelle
Thomas S. Radi
Edward B. Rawson
Meir Razy
John Reddan
Larry K. Reed
Matthais F. Reese II
T. D. Regulinski
R. T. Reiner
Paul Renaud
Steven M. Rowan
John Rymer
Hom Sack
Julio Gonzalez Samz
Stephen R. Schach
Franz P. Schauer
Max Schindler
Norman Schneidewind
Wolf A. Schnoege
Robert G. Schueppert
David J. Schultz
Gregory D. Schumacher
Leonard W. Seagren
Gary T. Shea
Craig Shermer
Robert W. Shillato
Victor Shtern
David M. Siefert
David J. Simkins
William G. Singer
Jacob Slonim
H. Bennett Smith
Harry M. Sneed
J. G. Snodgrass
A. R. Sorkowitz
Hugh B. Spilhne
G. Wayne Staley
Mary Jane Stoughton
Alan N. Sukert
Robert A. Symes
Michael Taint
Richard H. Thayer
Paul U. Thompson
Michael H. Thursby
Terrence L. Tillmanns
Lawrence F. Tracey
Illeana Trandafir
Henry J. Trochesset
Robert Troy
C. L. Troyanowski
Dana L. Ulery
David Usechok
R. L. Vantilburg
P. M. Vatez
Osmo Vikman
R. Wachter
Dolores R. Wallace
William M. Walsh III
Thomas J. Walsh
Roger D. H. Warburton
Andrew H. Weigel
R. W. Werlwas
Charles J. Wertz
W. L. Whipple
Patrick J. Wilson
T. J. Wojcik
Paul Wolfgang
W. Martin Wong
Dennis L. Wood
Tom Worthington
Charles Wortz
Zhou Zhi Ying
A. W. Yonda
Natalie C. Yoptanka
Michael E. York
Don Zeleny
Marvin Zelkowitz
Hugh Zettel
Peter F. Zoll

When the IEEE Standards Board approved this standard on March 12, 1987, it had the following membership:

Donald C. Fleckenstein, *Chairman* **Marco W. Migliaro,** *Vice-Chairman*
Sava I. Sherr, *Secretary*

James H. Beall
Dennis Bodson
Marshall L. Cain
James M. Daly
Stephen R. Dillon
Eugene P. Fogarty
Jay Forster
Kenneth D. Hendrix
Irvin N. Howell
Leslie I. Kerr
Jack Kinn
Irving Kolodny
Joseph L. Koepfinger*
Edward Lohse
John May
Lawrence V. McCall
L. Bruce McClung
Donald T. Michael*
L. John Rankine
John P. Riganati
Gary S. Robinson
Frank L. Rose
Robert E. Rountree
William R. Tackaberry
William B. Wilkens
Helen M. Wood

* Member emeritus

Foreword

(This Foreword is not a part of IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions.)

Purpose

This recommended practice specifies the necessary information content and recommends an organization for software design descriptions. This document does not explicitly support, nor is it limited to, any particular software design methodology or descriptive technology. It will guide the production of anything from paper design documents to an automated database of design information. For an organization in the process of developing a design description standard, use of this document will help the new standard meet the needs of all of its users. For an organization with a mature design description standard, it should prove useful in evaluating and modifying that standard in light of the informational and organizational needs of the design description user community.

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

Overview

This document consists of six sections. Section 1 defines the scope of the recommended practice and Section 2 references other ANSI/IEEE standards that should be followed when applying this practice. Section 3 provides definitions of terms within the context of the practice. Section 4 places the Software Design Description into the framework of the software development life cycle. Section 5 describes the minimum information that shall be included in a software design description and Section 6 gives a recommended organization for software design descriptions. The Appendix shows a sample table of contents for a software design description.

Audience

This document is intended for those in technical and managerial positions who prepare and use software design descriptions. It will guide a designer in the selection, organization, and presentation of design information. It will help standards developers ensure that a design description is complete, concise, and well organized.

Software design descriptions play a pivotal role in the development and maintenance of software systems. During its lifetime, a given design description is used by project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers. Each of these users has unique needs, both in terms of required design information and optimal organization of that information. Hence, a design description must contain all the design information needed by those users.

Terminology

This recommended practice follows the IEEE Guide to Standards Development. In particular, the word *shall* and the imperative form identify mandatory material within the recommended practice. The words *should*, *might*, and *may* identify advisory material.

History

The project authorization request for development of this recommended practice was approved by the IEEE Standards Board on September 22, 1983. Modification of the authorization request to change the title and scope was approved on March 13, 1986. A series of 10 meetings were held within the United States and internationally between March, 1983 and March, 1986. These meetings produced the draft submitted for balloting in April, 1986.

Suggestions for the improvement of this practice will be welcome. They should be sent to

Secretary
IEEE Standards Board
Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street
New York, NY 10017

Contributors

This document was developed by the Software Design Description Working Group of the Software Engineering Standards Subcommittee of the IEEE Computer Society. The Software Design Description Working Group Steering Committee had the following members:

H. Jack Barnard, *Chairman* **James A. Darling,** *Vice Chairman*
Robert F. Metz, *Secretary*

Chris Beall
Patricia Cox
Leo Endres
H. Gregory Frank
Manoochehr Ghiassi
Daniel E. Klingler
John McArdle
Arthur L. Price
Basil Sherlund

The Software Design Description Working Group had the following members:

A. Frank Ackerman
Jim Anderson
Sandro Bologna
Fletcher Buckley
Lori J. Call
Wan P. Chiang
Francois Coallier
Cliff Cockerham
Patricia W. Daggett
Jim DeLeo
Cammie Donaldson
Euiar Dragstedt
Laurence E. Fishtahler
David Gelperin
Yair Gershkovitch
Tom Gilb
Shirley A. Gloss-Soler
Larry J. Hardouin
Fredrick Ho
David M. Home
William S. Junk
Laurel Kaleda
Tom Kurihara
Jim Lemmon
F. C. Lim
Oyvind Lorentzen
Bert Martin
Lindsay McDermid
Glen Meldrum
Walter Merenda
Randy Peterson
Robert Poston
Ian C. Pyle
Ann S. Ping
Hans Schaefer
David Schultz
David Siefert
Peter Smith
Richard H. Thayer
T. H. Tse
David Weiss
Charles J. Wertz
G. Robert Zambs

The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

A. Frank Ackerman
Jagdish Agrawal
Richard L. Aurbach
James Baldo, Jr
H. Jack Barnard
Leo Beltracchi
Yechiel Ben-Naftali
H. R. Berlack
Michael A. Blackledge
Ron Blair
Kevin W. Bowyer
Kathleen L. Briggs
A. Winsor Brown
F. Buckley
Homer C. Carney
Ronald G. Carter
Richard L. Chilausky
Robert N. Chorette
T. S. Chow
Slucki Jean Christophe
Jung K. Chung
Peter Coad, Jr
François Coallier
Sharon R. Cobb-Pierson
Christopher M. Cooke
A. J. Cote, Jr
Ismael Fuentes Crespo
Patricia W. Daggett
George D. Darling
Taz Daughtry
Peter A. Denny
Harpal S. Dhama
Mike Dotson
David C. Doty
Einar Dragstedt
W. DuBlanica
William P. Dupres
Michael Dutton
Robert E. Dwyer
Mary L. Eads
John D. Earls
Mike Edwards
L. G. Egan
W. D. Ehrenberger
Steven R. Eisen
Caroline L. Evans
John W. Fendrich
Robert G. Ferreol
Glenn S. Fields
Gordon Force
Julian Forster
Deborah L. Franke
C. R. Frederick
Carl Friedlander
Richard Fries
Michael Galinier
Leonard B. Gardner
David Gelperin
Tom Gilb
James L. Gildersleeve
Shirley Gloss-Soler
Ole Golubjatnikov
J. Kaye Grau
Andrej Grebenc
Thomas Griest
Robert S. Grossman
Victor M. Guarnera
Lawrence M. Gunther
David A. Gustafson
Russell Gustin
Michael Haggerty
Howard Hamer
Harry E. Hansen, Jr
Robert M. Haralick
Haus-Ludis Hauser
Clark M. Hay
H. Hect
Terry L. Hengl
Maretta T. Holden
Charles P. Hollocker
Mark Holthouse
John Horch
Cheng Hu
Peter L. Hung
Sheng-Sheng Jeng
Laurel Kaleda
Charles F. Kaminski
Constantine Kaniklidis
Myron S. Karasik
Adi N. Kasad
Ron Kenett
R. A. Kessler
Shaye Koenig
Edward E. Kopicky
Joseph A. Krupinski
H. M. Kudyan
Joan Kundig
Lak-Ming Lam
John B. Lane
Robert Lane
William P. LaPlant
Greg Larsen
John A. Latimer
John A. N. Lee
Leon S. Levy
Paul Liebertz
F. C. Lim
Bertil Lindberg

IEEE
Std 1016-1987

IEEE Recommended Practice for Software Design Descriptions

Sponsor

Software Engineering Subcommittee
of the
Technical Committee on Software Engineering of the
IEEE Computer Society

The Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street, New York, NY 10017, USA

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE which have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. 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 once every five years for revision or reaffirmation. When a document is more than five years old, and has not been reaffirmed, 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.

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.

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 all 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 technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
345 East 47th Street
New York, NY 10017
USA

IEEE Recommended Practice for Software Design Descriptions

1. Scope

This is a recommended practice for describing software designs. It specifies the necessary information content, and recommended organization for a software design description. A software design description is a representation of a software system that is used as a medium for communicating software design information.

The practice may be applied to commercial, scientific, or military software that runs on any digital computer. Applicability is not restricted by the size, complexity, or criticality of the software.

This practice is not limited to specific methodologies for design, configuration management, or quality assurance. It is assumed that the quality design information and changes to the design of description will be managed by other project activities. Finally, this document does not support nor is it limited to any particular descriptive technique. It may be applied to paper documents, automated databases, design description languages, or other means of description.

2. References

This standard shall be used in conjunction with the following publications:

[1] ANSI/IEEE Std 729-1983, IEEE Standard Glossary of Software Engineering Terminology.[1]

[2] ANSI/IEEE Std 730-1984, IEEE Standard for Software Quality Assurance Plans.

[3] ANSI/IEEE Std 828-1983, IEEE Standard for Software Configuration Management Plans.

[4] ANSI/IEEE Std 830-1984, IEEE Guide to Software Requirements Specifications

[5] Freeman, P. and A. I. Wasserman. *Tutorial on Software Design Techniques.* 4th Edition, IEEE Computer Society Press, Annotated Bibliography, pp 715-718, 1983.

3. Definitions

The definitions listed here establish meaning in the context of this recommended practice. Definitions of other terms used in this document can be found in ANSI/IEEE Std 729-1983 [1].[2]

design entity. An element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.

design view. A subset of design entity attribute information that is specifically suited to the needs of a software project activity.

entity attribute. A named characteristic or property of a design entity. It provides a statement of fact about the entity.

software design description (SDD). A representation of a software system created to facilitate analysis, planning, implementation, and decision making. A blueprint or model of the software system. The SDD is used as the primary medium for communicating software design information.

4. Considerations for Producing a Software Design Description (SDD)

This section provides information to be considered before producing an SDD. How the SDD fits

[1] ANSI/IEEE publications can be obtained from the Sales Department, American National Standards Institute, 1430 Broadway, New York, NY 10018, or from the Service Center, The Institute of Electrical and Electronics Engineers, 445 Hoes Lane, Piscataway, NJ 08855-1331.

[2] Numbers in brackets correspond to those of the references in Section 2 of this standard.

into the software life cycle, where it fits, and why it is used are discussed.

4.1 Software Life Cycle. The life cycle of a software system is normally defined as the period of time that starts when a software product is conceived and ends when the product is no longer available for use. The life cycle approach is an effective engineering management tool and provides a model for a context within which to discuss the preparation and use of the SDD. While it is beyond the scope of this document to prescribe a particular standard life cycle, a typical cycle will be used to define such a context for the SDD. This cycle is based on ANSI/IEEE Std 729-1983 [1] and consists of a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and retirement phase.

4.2 Software Design Description (SDD) within the Life Cycle. For both new software systems and existing systems under maintenance, it is important to ensure that the design and implementation used for a software system satisfy the requirements driving that system. The minimum documentation required to do this is defined in ANSI/IEEE Std 730-1984 [2]. The SDD is one of these required products. It records the result of the design processes that are carried out during the design phase.

4.3 Purpose of a Software Design Description (SDD). The SDD shows how the software system will be structured to satisfy the requirements identified in the software requirements specification ANSI/IEEE Std 830-1984 [4]. It is a translation of requirements into a description of the software structure, software components, interfaces, and data necessary for the implementation phase. In essence, the SDD becomes a detailed blueprint for the implementation activity. In a complete SDD, each requirement must be traceable to one or more design entities.

5. Design Description Information Content

5.1 Introduction. A software design description is a representation or model of the software system to be created. The model should provide the precise design information needed for planning, analysis, and implementation of the software system. It should represent a partitioning of the system into design entities and describe the important properties and relationships among those entities.

The design description model used to represent a software system can be expressed as a collection of design entities, each possessing properties and relationships.[3] To simplify the model, the properties and relationships of each design entity are described by a standard set of attributes. The design information needs of project members are satisfied through identification of the entities and their associated attributes. A design description is complete when the attributes have been specified for all the entities.

5.2 Design Entities. A *design entity* is an element (component) of a design that is structurally and functionally distinct from other elements and that is separately named and referenced.

Design entities result from a decomposition of the software system requirements. The objective is to divide the system into separate components that can be considered, implemented, changed, and tested with minimal effect on other entities.

Entities can exist as a system, subsystems, data stores, modules, programs, and processes; see ANSI/IEEE Std 729-1983 [1]. The number and type of entities required to partition a design are dependent on a number of factors, such as the complexity of the system, the design technique used, and the programming environment.

Although entities are different in nature, they possess common characteristics. Each design entity will have a name, purpose, and function. There are common relationships among entities such as interfaces or shared data. The common characteristics of entities are described by design entity attributes.

5.3 Design Entity Attributes. A *design entity attribute* is a named characteristic or property of a design entity. It provides a statement of fact about the entity.

Design entity attributes can be thought of as questions about design entities. 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.

[3] The design description model is similar to an entity-relationship model, a common approach to information modeling.

The list of design entity attributes presented in this section is the minimum set required for all software design descriptions. The selection of these attributes is based on three criteria:

(1) The attribute is necessary for all software projects

(2) An incorrect specification of the attribute value could result in a fault in the software system to be developed

(3) The attribute describes intrinsic design information and not information related to the design process. Examples of attributes that do not meet the second and third criteria are designer names, design status, and revision history. This important process information is maintained by other software project activities as described in ANSI/IEEE Std 730-1984 [2] and ANSI/IEEE Std 828-1983 [3].

All attributes shall be specified for each 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*. When additional attributes are identified for a specific software project, they should be included in the design description. The attributes and associated information items are defined in 5.3.1 through 5.3.10.

5.3.1 Identification. *The name of the entity.* Two entities shall not have the same name. The names for the entities may be selected to characterize their nature. This will simplify referencing and tracking in addition to providing identification.

5.3.2 Type. *A description of the kind of entity.* The type attribute shall describe the nature of the entity. It may simply name the kind of entity, such as subprogram, module, procedure, process, or data store. Alternatively, design entities may be grouped into major classes to assist in locating an entity dealing with a particular type of information. For a given design description, the chosen entity types shall be applied consistently.

5.3.3 Purpose. *A description of why the entity exists.* The purpose attribute shall provide the rationale for the creation of the entity. Therefore, it shall designate the specific functional and performance requirements for which this entity was created; see ANSI/IEEE Std 830-1984 [4]. The purpose attribute shall also describe special requirements that must be met by the entity that are not included in the software requirements specification.

5.3.4 Function. *A statement of what the entity does.* The function attribute shall state the transformation applied by the entity to inputs to produce the desired output. In the case of a data entity, this attribute shall state the type of information stored or transmitted by the entity.

5.3.5 Subordinates. *The identification of all entities composing this entity.* The subordinates attribute shall identify 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 software system decomposition.

5.3.6 Dependencies. *A description of the relationships of this entity with other entities.* The dependencies attribute shall identify the *uses* or *requires the presence of* relationship for an entity. These relationships are often graphically depicted by structure charts, data flow diagrams, and transaction diagrams.

This attribute shall describe the nature of each interaction including such characteristics as timing and conditions for interaction. The interactions may involve the initiation, order of execution, data sharing, creation, duplicating, usage, storage, or destruction of entities.

5.3.7 Interface. *A description of how other entities interact with this entity.* The interface attribute shall describe the *methods* of interaction and the *rules* governing those interactions. The 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 shall provide 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.

5.3.8 Resources. *A description of the elements used by the entity that are external to the design.* The resources attribute shall identify and describe 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 shall 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 shall describe usage characteristics such as the process 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.

5.3.9 Processing. *A description of the rules used by the entity to achieve its function.* The processing attribute shall describe the algorithm used by the entity to perform a specific task and shall include contingencies. This description is a refinement of the function attribute. It is the most detailed level of refinement for this 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.

5.3.10 Data. *A description of data elements internal to the entity.* The data attribute shall describe 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 shall describe everything pertaining to the use of data or internal data structures by this entity. It shall include data specifications such as formats, number of elements, and initial values. It shall 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 shall 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 shall include a description of data validation needed for the process.

6. Design Description Organization

6.1 Introduction. Each design description user may have a different view of what is considered the essential aspects of a software design. All other information is extraneous to that user. The proportion of useful information for a specific user will decrease with the size and complexity of a software project. The needed information then becomes difficult or impractical to extract from the description and impossible to assimilate. Hence, a practical organization of the necessary design information is essential to its use.

This section introduces the notion of *design views* to aid in organizing the design attribute information defined in Section 5. It does not supplement Section 5 by providing additional design information nor does it prescribe the format or documentation practice for design views.

A recommended organization of design entities and their associated attributes are presented in this section to facilitate the access of design information from various technical viewpoints. This recommended organization is flexible and can be implemented through different media such as paper documentation, design languages, or database management systems with automated report generation, and query language access. Since paper documentation is currently the primary design description medium, a sample table of contents is given in the Appendix.

6.2 Design Views. Entity attribute information can be organized in several ways to reveal all of the essential aspects of a design. In so doing, the user is able to focus on design details from a different perspective or viewpoint. A *design view* is a subset of design entity attribute information that is specifically suited to the needs of a software project activity.

Each design view represents a separate concern about a software system. Together, these views provide a comprehensive description of the design in a concise and usable form that simplifies information access and assimilation.

A recommended organization of the SDD into separate design views to facilitate information access and assimilation is given in Table 1. Each of these views, their use, and representation are discussed in detail.

6.2.1 Decomposition Description

6.2.1.1 Scope. The decomposition description records the division of the software system into design entities. It describes the way the system has been structured and the purpose and function of each entity. For each entity, it provides a reference to the detailed description via the identification attribute.

Table 1
Recommended Design Views

Design View	Scope	Entity Attributes	Example Representations
Decomposition Description	Partition of the system into design entities	Identification, type, purpose, function, subordinates	Hierarchical decomposition diagram, natural language
Dependency Description	Description of the relationships among entities and system resources	Identification, type, purpose, dependencies, resources	Structure charts, data flow diagrams, transaction diagrams
Interface Description	List of everything a designer, programmer, or tester needs to know to use the design entities that make up the system	Identification, function, interfaces	Interface files, parameter tables
Detail Description	Description of the internal design details of an entity	Identification, processing, data	Flowcharts, N-S charts, PDL

The attribute descriptions for identification, type, purpose, function, and subordinates should be included in this design view. This attribute information should be provided for all design entities.

6.2.1.2 Use. The decomposition description can be used by designers and maintainers to identify the major design entities of the system for purposes such as determining which entity is responsible for performing specific functions and tracing requirements to design entities. Design entities can be grouped into major classes to assist in locating a particular type of information and to assist in reviewing the decomposition for completeness. For example, a module decomposition may exist separately from a data decomposition.

The information in the decomposition description can be used by project management for planning, monitoring, and control of a software project. They can identify each software component, its purpose, and basic functionality. This design information together with other project information can be used in estimating cost, staff, 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 ANSI/IEEE Std 828-1983 [3]. Metrics developers may also use this information for initial complexity, sizing, staffing, and development time parameters. The software quality assurance staff can use the decomposition description to construct a requirements traceability matrix.

6.2.1.3 Representation. The literature on software engineering describes a number of methods that provide consistent criteria for entity decomposition [5]. These methods provide for designing simple, independent entities and are based on such principles as structured design and information hiding. The primary graphical technique used to describe system decomposition is a hierarchical decomposition diagram. This diagram can be used together with natural language descriptions of purpose and function for each entity.

6.2.2 Dependency Description

6.2.2.1 Scope. The dependency description specifies the relationships among entities. It identifies the dependent entities, describes their coupling, and identifies the required resources.

This design view defines the strategies for interactions among design entities and provides the information needed to easily preceive how, why, where, and at what level system actions occur. It specifies the type of relationships that exist among the entities such as shared information, prescribed order of execution, or well defined parameter interfaces.

The attribute descriptions for identification, type, purpose, dependencies, and resources should be included in this design view. This attribute information should be provided for all design entities.

6.2.2.2 Use. The dependency description provides an overall picture of how the system works in order to assess the impact of requirements and 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.

6.2.2.3 Representation. There are a number of methods that help minimize the relationships among entities by maximizing the relationship among elements in the same entity. These methods emphasize low module coupling and high module cohesion [5].

Formal specification languages provide for the specification of system functions and data, their interrelationships, the inputs and outputs, and other system aspects in a well-defined language. The relationship among design entities is also represented by data flow diagrams, structure charts, or transaction diagrams.

6.2.3 Interface Description

6.2.3.1 Scope. The entity interface description provides everything designers, programmers, and testers need to know to correctly use the functions provided by an entity. This description includes the details of external and internal interfaces not provided in the software requirements specification.

This design view consists of a set of interface specifications for each entity. The attribute descriptions for identification, function, and interfaces should be included in this design view. This attribute information should be provided for all design entities.

6.2.3.2 Use. The interface 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. In addition, the interface description may be used by technical writers to produce customer documentation or may be used directly by customers. In the later case, the interface description could result in the production of a human interface view.

Designers, programmers, and testers may need to use design entities that they did not develop. These entities may be reused from earlier projects, contracted from an external source, or produced by other developers. The interface description settles the 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 multiperson development for smooth integration and ease of maintenance.

6.2.3.3 Representation. The interface description should provide the language for communicating with each entity to include screen formats, valid inputs, and resulting outputs. For those entities that are data driven, 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.

6.2.4 Detailed Design Description

6.2.4.1 Scope. The detailed design description contains the internal details of each design entity. These details include the attribute descriptions for identification, processing, and data. This attribute information should be provided for all design entities.

6.2.4.2 Use. This description contains the details needed by programmers prior to implementation. The detailed design description can also be used to aid in producing unit test plans.

6.2.4.3 Representation. There are many tools used to describe the details of design entities. Program design languages can be used to describe inputs, outputs, local data and the algorithm for an entity. Other common techniques for describing design entity logic include using metacode or structured English, or graphical methods such as Nassi-Schneidermann charts or flowcharts.

Foreword

(This Foreword is not a part of IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions.)

Purpose

This recommended practice specifies the necessary information content and recommends an organization for software design descriptions. This document does not explicitly support, nor is it limited to, any particular software design methodology or descriptive technology. It will guide the production of anything from paper design documents to an automated database of design information. For an organization in the process of developing a design description standard, use of this document will help the new standard meet the needs of all of its users. For an organization with a mature design description standard, it should prove useful in evaluating and modifying that standard in light of the informational and organizational needs of the design description user community.

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

Overview

This document consists of six sections. Section 1 defines the scope of the recommended practice and Section 2 references other ANSI/IEEE standards that should be followed when applying this practice. Section 3 provides definitions of terms within the context of the practice. Section 4 places the Software Design Description into the framework of the software development life cycle. Section 5 describes the minimum information that shall be included in a software design description and Section 6 gives a recommended organization for software design descriptions. The Appendix shows a sample table of contents for a software design description.

Audience

This document is intended for those in technical and managerial positions who prepare and use software design descriptions. It will guide a designer in the selection, organization, and presentation of design information. It will help standards developers ensure that a design description is complete, concise, and well organized.

Software design descriptions play a pivotal role in the development and maintenance of software systems. During its lifetime, a given design description is used by project managers, quality assurance staff, configuration managers, software designers, programmers, testers, and maintainers. Each of these users has unique needs, both in terms of required design information and optimal organization of that information. Hence, a design description must contain all the design information needed by those users.

Terminology

This recommended practice follows the IEEE Guide to Standards Development. In particular, the word *shall* and the imperative form identify mandatory material within the recommended practice. The words *should*, *might*, and *may* identify advisory material.

History

The project authorization request for development of this recommended practice was approved by the IEEE Standards Board on September 22, 1983. Modification of the authorization request to change the title and scope was approved on March 13, 1986. A series of 10 meetings were held within the United States and internationally between March, 1983 and March, 1986. These meetings produced the draft submitted for balloting in April, 1986.

Suggestions for the improvement of this practice will be welcome. They should be sent to

Secretary
IEEE Standards Board
Institute of Electrical and Electronics Engineers, Inc
345 East 47th Street
New York, NY 10017

Contributors

This document was developed by the Software Design Description Working Group of the Software Engineering Standards Subcommittee of the IEEE Computer Society. The Software Design Description Working Group Steering Committee had the following members:

H. Jack Barnard, *Chairman* **James A. Darling,** *Vice Chairman*
Robert F. Metz, *Secretary*

Chris Beall
Patricia Cox
Leo Endres
H. Gregory Frank
Manoochehr Ghiassi
Daniel E. Klingler
John McArdle
Arthur L. Price
Basil Sherlund

The Software Design Description Working Group had the following members:

A. Frank Ackerman
Jim Anderson
Sandro Bologna
Fletcher Buckley
Lori J. Call
Wan P. Chiang
Francois Coallier
Cliff Cockerham
Patricia W. Daggett
Jim DeLeo
Cammie Donaldson
Euiar Dragstedt
Laurence E. Fishtahler
David Gelperin
Yair Gershkovitch
Tom Gilb
Shirley A. Gloss-Soler
Larry J. Hardouin
Fredrick Ho
David M. Home
William S. Junk
Laurel Kaleda
Tom Kurihara
Jim Lemmon
F. C. Lim
Oyvind Lorentzen
Bert Martin
Lindsay McDermid
Glen Meldrum
Walter Merenda
Randy Peterson
Robert Poston
Ian C. Pyle
Ann S. Ping
Hans Schaefer
David Schultz
David Siefert
Peter Smith
Richard H. Thayer
T. H. Tse
David Weiss
Charles J. Wertz
G. Robert Zambs

The following persons were on the balloting committee that approved this document for submission to the IEEE Standards Board:

A. Frank Ackerman
Jagdish Agrawal
Richard L. Aurbach
James Baldo, Jr
H. Jack Barnard
Leo Beltracchi
Yechiel Ben-Naftali
H. R. Berlack
Michael A. Blackledge
Ron Blair
Kevin W. Bowyer
Kathleen L. Briggs
A. Winsor Brown
F. Buckley
Homer C. Carney
Ronald G. Carter
Richard L. Chilausky
Robert N. Chorette
T. S. Chow
Slucki Jean Christophe
Jung K. Chung
Peter Coad, Jr
François Coallier
Sharon R. Cobb-Pierson
Christopher M. Cooke
A. J. Cote, Jr
Ismael Fuentes Crespo
Patricia W. Daggett
George D. Darling
Taz Daughtry
Peter A. Denny
Harpal S. Dhama
Mike Dotson
David C. Doty
Einar Dragstedt
W. DuBlanica
William P. Dupres
Michael Dutton
Robert E. Dwyer
Mary L. Eads
John D. Earls
Mike Edwards
L. G. Egan
W. D. Ehrenberger
Steven R. Eisen
Caroline L. Evans
John W. Fendrich
Robert G. Ferreol
Glenn S. Fields
Gordon Force
Julian Forster
Deborah L. Franke
C. R. Frederick
Carl Friedlander
Richard Fries
Michael Galinier
Leonard B. Gardner
David Gelperin
Tom Gilb
James L. Gildersleeve
Shirley Gloss-Soler
Ole Golubjatnikov
J. Kaye Grau
Andrej Grebenc
Thomas Griest
Robert S. Grossman
Victor M. Guarnera
Lawrence M. Gunther
David A. Gustafson
Russell Gustin
Michael Haggerty
Howard Hamer
Harry E. Hansen, Jr
Robert M. Haralick
Haus-Ludis Hauser
Clark M. Hay
H. Hect
Terry L. Hengl
Maretta T. Holden
Charles P. Hollocker
Mark Holthouse
John Horch
Cheng Hu
Peter L. Hung
Sheng-Sheng Jeng
Laurel Kaleda
Charles F. Kaminski
Constantine Kaniklidis
Myron S. Karasik
Adi N. Kasad
Ron Kenett
R. A. Kessler
Shaye Koenig
Edward E. Kopicky
Joseph A. Krupinski
H. M. Kudyan
Joan Kundig
Lak-Ming Lam
John B. Lane
Robert Lane
William P. LaPlant
Greg Larsen
John A. Latimer
John A. N. Lee
Leon S. Levy
Paul Liebertz
F. C. Lim
Bertil Lindberg