TECHNICAL REPORT

ISO/IEC TR 19759

Second edition
2015-10-01

Corrected version
2016-06-01

Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)

Ingénierie du logiciel — Guide du corps de connaissance de l'ingénierie du logiciel (SWEBOK)

Reference number
ISO/IEC TR 19759:2016(E)

Documents
Document Sharing Hub
© ISO/IEC 2016

Full document details: ISO_IEC TR 19759_2015

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2016, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

Contents

Page

Foreword iv
Introduction v
1 Scope 1
2 Recommendations 1
3 Revision of the IEEE Computer Society Publication 1
4 Publication 1

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

This corrected version of ISO/IEC TR 19759:2015 incorporates the following corrections:

- corrupted paragraphs of text within the document have been replaced.

ISO/IEC TR 19759 was prepared by the IEEE Computer Society and was adopted, under a special “fast-track procedure”, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

Introduction

The purpose of the Guide to the Software Engineering Body of Knowledge is to provide a consensually validated characterization of the bounds of the software engineering discipline and to provide a topical access to the Body of Knowledge supporting that discipline. The Body of Knowledge is subdivided into fifteen software engineering Knowledge Areas (KA) providing an outline of topics. The descriptions in the KAs are designed to discriminate among the various important concepts, permitting readers to find their way quickly to subjects of interest. Upon finding a subject, readers are referred to key papers or books selected because they succinctly present the knowledge.

Publication of the 2004 version of this Guide to the Software Engineering Body of Knowledge (SWEBOK 2004)—adopted as ISO/IEC TR 19759:2005—was a major milestone in establishing software engineering as a recognized engineering discipline. The goal in developing this update to SWEBOK is to improve the currency, readability, consistency, and usability of the Guide. All knowledge areas (KAs) have been updated to reflect changes in software engineering since publication of SWEBOK 2004. Four new foundation KAs and a Software Engineering Professional Practices KA have been added. The Software Engineering Tools and Methods KA has been revised as Software Engineering Models and Methods. Software engineering tools is now a topic in each of the KAs. Three appendices provide the specifications for the KA description, an annotated set of relevant standards for each KA, and a listing of the references cited in the Guide.

An emphasis on engineering practice leads the Guide toward a strong relationship with the normative literature. Most of the computer science, information technology and software engineering literature provides information useful to software engineers, but a relatively small portion is normative. A normative document prescribes what an engineer should do in a specified situation rather than providing information that might be helpful. The normative literature is validated by consensus formed among practitioners and is concentrated in standards and related documents. From the beginning, the SWEBOK project was conceived as having a strong relationship to the normative literature of software engineering. The two major standards bodies for software engineering (IEEE Computer Society Software and Systems Engineering Standards Committee and ISO/IEC JTC1/SC7) cooperated in the project.

The Guide is oriented toward a variety of audiences, all over the world. It aims to serve public and private organizations in need of a consistent view of software engineering for defining education and training requirements, classifying jobs, developing performance evaluation policies or specifying software development tasks. It also addresses practicing, or managing, software engineers and the officials responsible for making public policy regarding licensing and professional guidelines. In addition, professional societies and educators defining the certification rules, accreditation policies for university curricula, and guidelines for professional practice will benefit from the SWEBOK Guide, as well as the students learning the software engineering profession and educators and trainers engaged in defining curricula and course content.

Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK) — Version 3.0

1 Scope

This Technical report characterizes the boundaries of the software engineering discipline and provides topical access to the literature supporting that discipline.

2 Recommendations

As a Technical Report, this document does not make technical recommendations. The information provided is that contained in the following publication (reproduced on the following pages), which is adopted as a Technical Report:

Guide to the Software Engineering Body of Knowledge (SWEBOK) V3.0, IEEE Computer Society

3 Revision of the IEEE Computer Society Publication

It has been agreed with the IEEE Computer Society that ISO/IEC JTC 1/SC 7 will be consulted in the event of any revision or amendment of this IEEE Computer Society publication. Consultation will be accomplished by the existing Category A liaison relationship between SC 7 and IEEE Computer Society.

4 Publication

The report of the IEEE Computer Society appears on the following pages.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

SWEBOK® V3.0

Guide to the Software Engineering Body of Knowledge

Editors

Pierre Bourque
Richard E. (Dick) Fairley

IEEE

IEEE computer society

Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

Guide to the Software Engineering Body of Knowledge

Version 3.0

SWEBOK®

A Project of the IEEE Computer Society

Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

Guide to the Software Engineering Body of Knowledge

Version 3.0

SWEBOK®

Editors

Pierre Bourque, École de technologie supérieure (ÉTS)
Richard E. (Dick) Fairley, Software and Systems Engineering Associates (S2EA)

IEEE

IEEE computer society

Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

Copyright and Reprint Permissions. Educational or personal use of this material is permitted without fee provided such copies 1) are not made for profit or in lieu of purchasing copies for classes, and that this notice and a full citation to the original work appear on the first page of the copy and 2) do not imply IEEE endorsement of any third-party products or services. Permission to reprint/republish this material for commercial, advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org.

Reference to any specific commercial products, process, or service does not imply endorsement by IEEE. The views and opinions expressed in this work do not necessarily reflect those of IEEE.

IEEE makes this document available on an "as is" basis and makes no warranty, express or implied, as to the accuracy, capability, efficiency merchantability, or functioning of this document. In no event will IEEE be liable for any general, consequential, indirect, incidental, exemplary, or special damages, even if IEEE has been advised of the possibility of such damages.

Copyright © 2014 IEEE. All rights reserved.
Paperback ISBN-10: 0-7695-5166-1
Paperback ISBN-13: 978-0-7695-5166-1

Digital copies of *SWEBOK Guide* V3.0 may be downloaded free of charge for personal and academic use via www.swebok.org.

IEEE Computer Society Staff for This Publication
Angela Burgess, Executive Director
Anne Marie Kelly, Associate Executive Director, Director of Governance
Evan M. Butterfield, Director of Products and Services
John Keppler, Senior Manager, Professional Education
Kate Guillemette, Product Development Editor
Dorian McClenahan, Education Program Product Developer
Michelle Phon, Professional Education & Certification Program Coordinator
Jennie Zhu-Mai, Editorial Designer

IEEE Computer Society Products and Services. The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering journals, magazines, conference proceedings, and professional education products. Visit the Computer Society at www.computer.org for more information.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

TABLE OF CONTENTS

Foreword xvii
Foreword to the 2004 Edition xix
Editors xxi
Coeditors xxi
Contributing Editors xxi
Change Control Board xxi
Knowledge Area Editors xxiii
Knowledge Area Editors of Previous SWEBOK Versions xxv
Review Team xxvii
Acknowledgements xxix
Professional Activities Board, 2013 Membership xxix
Motions Regarding the Approval of SWEBOK Guide V3.0 xxx
Motions Regarding the Approval of SWEBOK Guide 2004 Version xxx
Introduction to the Guide xxxi

Chapter 1: Software Requirements 1-1
1. Software Requirements Fundamentals 1-1
1.1. Definition of a Software Requirement 1-1
1.2. Product and Process Requirements 1-2
1.3. Functional and Nonfunctional Requirements 1-3
1.4. Emergent Properties 1-3
1.5. Quantifiable Requirements 1-3
1.6. System Requirements and Software Requirements 1-3
2. Requirements Process 1-3
2.1. Process Models 1-4
2.2. Process Actors 1-4
2.3. Process Support and Management 1-4
2.4. Process Quality and Improvement 1-4
3. Requirements Elicitation 1-5
3.1. Requirements Sources 1-5
3.2. Elicitation Techniques 1-6
4. Requirements Analysis 1-7
4.1. Requirements Classification 1-7
4.2. Conceptual Modeling 1-8
4.3. Architectural Design and Requirements Allocation 1-9
4.4. Requirements Negotiation 1-9
4.5. Formal Analysis 1-10
5. Requirements Specification 1-10
5.1. System Definition Document 1-10
5.2. System Requirements Specification 1-10
5.3. Software Requirements Specification 1-11
6. Requirements Validation 1-11
6.1. Requirements Reviews 1-11
6.2. Prototyping 1-12

© ISO/IEC 2016 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

6.3. Model Validation *1-12*
6.4. Acceptance Tests *1-12*
7. Practical Considerations 1-12
7.1. Iterative Nature of the Requirements Process *1-13*
7.2. Change Management *1-13*
7.3. Requirements Attributes *1-13*
7.4. Requirements Tracing *1-14*
7.5. Measuring Requirements *1-14*
8. Software Requirements Tools 1-14
Matrix of Topics vs. Reference Material 1-15

Chapter 2: Software Design **2-1**
1. Software Design Fundamentals 2-2
1.1. General Design Concepts *2-2*
1.2. Context of Software Design *2-2*
1.3. Software Design Process *2-2*
1.4. Software Design Principles *2-3*
2. Key Issues in Software Design 2-3
2.1. Concurrency *2-4*
2.2. Control and Handling of Events *2-4*
2.3. Data Persistence *2-4*
2.4. Distribution of Components *2-4*
2.5. Error and Exception Handling and Fault Tolerance *2-4*
2.6. Interaction and Presentation *2-4*
2.7. Security *2-4*
3. Software Structure and Architecture 2-4
3.1. Architectural Structures and Viewpoints *2-5*
3.2. Architectural Styles *2-5*
3.3. Design Patterns *2-5*
3.4. Architecture Design Decisions *2-5*
3.5. Families of Programs and Frameworks *2-5*
4. User Interface Design 2-5
4.1. General User Interface Design Principles *2-6*
4.2. User Interface Design Issues *2-6*
4.3. The Design of User Interaction Modalities *2-6*
4.4. The Design of Information Presentation *2-6*
4.5. User Interface Design Process *2-7*
4.6. Localization and Internationalization *2-7*
4.7. Metaphors and Conceptual Models *2-7*
5. Software Design Quality Analysis and Evaluation 2-7
5.1. Quality Attributes *2-7*
5.2. Quality Analysis and Evaluation Techniques *2-8*
5.3. Measures *2-8*
6. Software Design Notations 2-8
6.1. Structural Descriptions (Static View) *2-8*
6.2. Behavioral Descriptions (Dynamic View) *2-9*
7. Software Design Strategies and Methods 2-10
7.1. General Strategies *2-10*
7.2. Function-Oriented (Structured) Design *2-10*
7.3. Object-Oriented Design *2-10*

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

7.4. Data Structure-Centered Design *2-10*
7.5. Component-Based Design (CBD) *2-10*
7.6. Other Methods *2-10*
8. Software Design Tools 2-11
Matrix of Topics vs. Reference Material 2-12

Chapter 3: Software Construction **3-1**
1. Software Construction Fundamentals 3-1
1.1. Minimizing Complexity *3-3*
1.2. Anticipating Change *3-3*
1.3. Constructing for Verification *3-3*
1.4. Reuse *3-3*
1.5. Standards in Construction *3-3*
2. Managing Construction 3-4
2.1. Construction in Life Cycle Models *3-4*
2.2. Construction Planning *3-4*
2.3. Construction Measurement *3-4*
3. Practical Considerations 3-5
3.1. Construction Design *3-5*
3.2. Construction Languages *3-5*
3.3. Coding *3-6*
3.4. Construction Testing *3-6*
3.5. Construction for Reuse *3-6*
3.6. Construction with Reuse *3-7*
3.7. Construction Quality *3-7*
3.8. Integration *3-7*
4. Construction Technologies 3-8
4.1. API Design and Use *3-8*
4.2. Object-Oriented Runtime Issues *3-8*
4.3. Parameterization and Generics *3-8*
4.4. Assertions, Design by Contract, and Defensive Programming *3-8*
4.5. Error Handling, Exception Handling, and Fault Tolerance *3-9*
4.6. Executable Models *3-9*
4.7. State-Based and Table-Driven Construction Techniques *3-9*
4.8. Runtime Configuration and Internationalization *3-10*
4.9. Grammar-Based Input Processing *3-10*
4.10. Concurrency Primitives *3-10*
4.11. Middleware *3-10*
4.12. Construction Methods for Distributed Software *3-11*
4.13. Constructing Heterogeneous Systems *3-11*
4.14. Performance Analysis and Tuning *3-11*
4.15. Platform Standards *3-11*
4.16. Test-First Programming *3-11*
5. Software Construction Tools 3-12
5.1. Development Environments *3-12*
5.2. GUI Builders *3-12*
5.3. Unit Testing Tools *3-12*
5.4. Profiling, Performance Analysis, and Slicing Tools *3-12*
Matrix of Topics vs. Reference Material 3-13

Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

Chapter 4: Software Testing **4-1**
1. Software Testing Fundamentals 4-3
1.1. Testing-Related Terminology 4-3
1.2. Key Issues 4-3
1.3. Relationship of Testing to Other Activities 4-4
2. Test Levels 4-5
2.1. The Target of the Test 4-5
2.2. Objectives of Testing 4-5
3. Test Techniques 4-7
3.1. Based on the Software Engineer's Intuition and Experience 4-8
3.2. Input Domain-Based Techniques 4-8
3.3. Code-Based Techniques 4-8
3.4. Fault-Based Techniques 4-9
3.5. Usage-Based Techniques 4-9
3.6. Model-Based Testing Techniques 4-10
3.7. Techniques Based on the Nature of the Application 4-10
3.8. Selecting and Combining Techniques 4-11
4. Test-Related Measures 4-11
4.1. Evaluation of the Program Under Test 4-11
4.2. Evaluation of the Tests Performed 4-12
5. Test Process 4-12
5.1. Practical Considerations 4-13
5.2. Test Activities 4-14
6. Software Testing Tools 4-15
6.1. Testing Tool Support 4-15
6.2. Categories of Tools 4-15
Matrix of Topics vs. Reference Material 4-17

Chapter 5: Software Maintenance **5-1**
1. Software Maintenance Fundamentals 5-1
1.1. Definitions and Terminology 5-1
1.2. Nature of Maintenance 5-2
1.3. Need for Maintenance 5-3
1.4. Majority of Maintenance Costs 5-3
1.5. Evolution of Software 5-3
1.6. Categories of Maintenance 5-3
2. Key Issues in Software Maintenance 5-4
2.1. Technical Issues 5-4
2.2. Management Issues 5-5
2.3. Maintenance Cost Estimation 5-6
2.4. Software Maintenance Measurement 5-7
3. Maintenance Process 5-7
3.1. Maintenance Processes 5-7
3.2. Maintenance Activities 5-8
4. Techniques for Maintenance 5-10
4.1. Program Comprehension 5-10
4.2. Reengineering 5-10
4.3. Reverse Engineering 5-10
4.4. Migration 5-10
4.5. Retirement 5-11

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

5. Software Maintenance Tools 5-11
Matrix of Topics vs. Reference Material 5-12

Chapter 6: Software Configuration Management 6-1
1. Management of the SCM Process 6-2
1.1. Organizational Context for SCM 6-2
1.2. Constraints and Guidance for the SCM Process 6-3
1.3. Planning for SCM 6-3
1.4. SCM Plan 6-5
1.5. Surveillance of Software Configuration Management 6-5
2. Software Configuration Identification 6-6
2.1. Identifying Items to Be Controlled 6-6
2.2. Software Library 6-8
3. Software Configuration Control 6-8
3.1. Requesting, Evaluating, and Approving Software Changes 6-8
3.2. Implementing Software Changes 6-9
3.3. Deviations and Waivers 6-10
4. Software Configuration Status Accounting 6-10
4.1. Software Configuration Status Information 6-10
4.2. Software Configuration Status Reporting 6-10
5. Software Configuration Auditing 6-10
5.1. Software Functional Configuration Audit 6-11
5.2. Software Physical Configuration Audit 6-11
5.3. In-Process Audits of a Software Baseline 6-11
6. Software Release Management and Delivery 6-11
6.1. Software Building 6-11
6.2. Software Release Management 6-12
7. Software Configuration Management Tools 6-12
Matrix of Topics vs. Reference Material 6-13

Chapter 7: Software Engineering Management 7-1
1. Initiation and Scope Definition 7-4
1.1. Determination and Negotiation of Requirements 7-4
1.2. Feasibility Analysis 7-4
1.3. Process for the Review and Revision of Requirements 7-5
2. Software Project Planning 7-5
2.1. Process Planning 7-5
2.2. Determine Deliverables 7-5
2.3. Effort, Schedule, and Cost Estimation 7-6
2.4. Resource Allocation 7-6
2.5. Risk Management 7-6
2.6. Quality Management 7-6
2.7. Plan Management 7-7
3. Software Project Enactment 7-7
3.1. Implementation of Plans 7-7
3.2. Software Acquisition and Supplier Contract Management 7-7
3.3. Implementation of Measurement Process 7-7
3.4. Monitor Process 7-7
3.5. Control Process 7-8
3.6. Reporting 7-8

AVA Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

4. Review and Evaluation 7-8
4.1. Determining Satisfaction of Requirements *7-8*
4.2. Reviewing and Evaluating Performance *7-9*
5. Closure 7-9
5.1. Determining Closure *7-9*
5.2. Closure Activities *7-9*
6. Software Engineering Measurement 7-9
6.1. Establish and Sustain Measurement Commitment *7-9*
6.2. Plan the Measurement Process *7-10*
6.3. Perform the Measurement Process *7-11*
6.4. Evaluate Measurement *7-11*
7. Software Engineering Management Tools 7-11
Matrix of Topics vs. Reference Material 7-13

Chapter 8: Software Engineering Process **8-1**
1. Software Process Definition 8-2
1.1. Software Process Management *8-3*
1.2. Software Process Infrastructure *8-4*
2. Software Life Cycles 8-4
2.1. Categories of Software Processes *8-5*
2.2. Software Life Cycle Models *8-5*
2.3. Software Process Adaptation *8-6*
2.4. Practical Considerations *8-6*
3. Software Process Assessment and Improvement 8-6
3.1. Software Process Assessment Models *8-7*
3.2. Software Process Assessment Methods *8-7*
3.3. Software Process Improvement Models *8-7*
3.4. Continuous and Staged Software Process Ratings *8-8*
4. Software Measurement 8-8
4.1. Software Process and Product Measurement *8-9*
4.2. Quality of Measurement Results *8-10*
4.3. Software Information Models *8-10*
4.4. Software Process Measurement Techniques *8-11*
5. Software Engineering Process Tools 8-12
Matrix of Topics vs. Reference Material 8-13

Chapter 9: Software Engineering Models and Methods **9-1**
1. Modeling 9-1
1.1. Modeling Principles *9-2*
1.2. Properties and Expression of Models *9-3*
1.3. Syntax, Semantics, and Pragmatics *9-3*
1.4. Preconditions, Postconditions, and Invariants *9-4*
2. Types of Models 9-4
2.1. Information Modeling *9-5*
2.2. Behavioral Modeling *9-5*
2.3. Structure Modeling *9-5*
3. Analysis of Models 9-5
3.1. Analyzing for Completeness *9-5*
3.2. Analyzing for Consistency *9-6*

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

3.3. Analyzing for Correctness *9-6*
3.4. Traceability *9-6*
3.5. Interaction Analysis *9-6*
4. Software Engineering Methods 9-7
4.1. Heuristic Methods *9-7*
4.2. Formal Methods *9-7*
4.3. Prototyping Methods *9-8*
4.4. Agile Methods *9-9*
Matrix of Topics vs. Reference Material 9-10

Chapter 10: Software Quality **10-1**
1. Software Quality Fundamentals 10-2
1.1. Software Engineering Culture and Ethics *10-2*
1.2. Value and Costs of Quality *10-3*
1.3. Models and Quality Characteristics *10-3*
1.4. Software Quality Improvement *10-4*
1.5. Software Safety *10-4*
2. Software Quality Management Processes 10-5
2.1. Software Quality Assurance *10-5*
2.2. Verification & Validation *10-6*
2.3. Reviews and Audits *10-6*
3. Practical Considerations 10-9
3.1. Software Quality Requirements *10-9*
3.2. Defect Characterization *10-10*
3.3. Software Quality Management Techniques *10-11*
3.4. Software Quality Measurement *10-12*
4. Software Quality Tools 10-12
Matrix of Topics vs. Reference Material 10-14

Chapter 11: Software Engineering Professional Practice **11-1**
1. Professionalism 11-2
1.1. Accreditation, Certification, and Licensing *11-3*
1.2. Codes of Ethics and Professional Conduct *11-4*
1.3. Nature and Role of Professional Societies *11-4*
1.4. Nature and Role of Software Engineering Standards *11-4*
1.5. Economic Impact of Software *11-5*
1.6. Employment Contracts *11-5*
1.7. Legal Issues *11-5*
1.8. Documentation *11-7*
1.9. Tradeoff Analysis *11-8*
2. Group Dynamics and Psychology 11-9
2.1. Dynamics of Working in Teams/Groups *11-9*
2.2. Individual Cognition *11-9*
2.3. Dealing with Problem Complexity *11-10*
2.4. Interacting with Stakeholders *11-10*
2.5. Dealing with Uncertainty and Ambiguity *11-10*
2.6. Dealing with Multicultural Environments *11-10*
3. Communication Skills 11-11
3.1. Reading, Understanding, and Summarizing *11-11*

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

3.2. Writing *11-11*
3.3. Team and Group Communication *11-11*
3.4. Presentation Skills *11-12*
Matrix of Topics vs. Reference Material 11-13

Chapter 12: Software Engineering Economics **12-1**
1. Software Engineering Economics Fundamentals 12-3
1.1. Finance *12-3*
1.2. Accounting *12-3*
1.3. Controlling *12-3*
1.4. Cash Flow *12-3*
1.5. Decision-Making Process *12-4*
1.6. Valuation *12-5*
1.7. Inflation *12-6*
1.8. Depreciation *12-6*
1.9. Taxation *12-6*
1.10. Time-Value of Money *12-6*
1.11. Efficiency *12-6*
1.12. Effectiveness *12-6*
1.13. Productivity *12-6*
2. Life Cycle Economics 12-7
2.1. Product *12-7*
2.2. Project *12-7*
2.3. Program *12-7*
2.4. Portfolio *12-7*
2.5. Product Life Cycle *12-7*
2.6. Project Life Cycle *12-7*
2.7. Proposals *12-8*
2.8. Investment Decisions *12-8*
2.9. Planning Horizon *12-8*
2.10. Price and Pricing *12-8*
2.11. Cost and Costing *12-9*
2.12. Performance Measurement *12-9*
2.13. Earned Value Management *12-9*
2.14. Termination Decisions *12-9*
2.15. Replacement and Retirement Decisions *12-10*
3. Risk and Uncertainty 12-10
3.1. Goals, Estimates, and Plans *12-10*
3.2. Estimation Techniques *12-11*
3.3. Addressing Uncertainty *12-11*
3.4. Prioritization *12-11*
3.5. Decisions under Risk *12-11*
3.6. Decisions under Uncertainty *12-12*
4. Economic Analysis Methods 12-12
4.1. For-Profit Decision Analysis *12-12*
4.2. Minimum Acceptable Rate of Return *12-13*
4.3. Return on Investment *12-13*
4.4. Return on Capital Employed *12-13*
4.5. Cost-Benefit Analysis *12-13*

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

4.6. Cost-Effectiveness Analysis *12-13*
4.7. Break-Even Analysis *12-13*
4.8. Business Case *12-13*
4.9. Multiple Attribute Evaluation *12-14*
4.10. Optimization Analysis *12-14*
5. Practical Considerations 12-14
5.1. The "Good Enough" Principle *12-14*
5.2. Friction-Free Economy *12-15*
5.3. Ecosystems *12-15*
5.4. Offshoring and Outsourcing *12-15*
Matrix of Topics vs. Reference Material 12-16

Chapter 13: Computing Foundations **13-1**
1. Problem Solving Techniques 13-3
1.1. Definition of Problem Solving *13-3*
1.2. Formulating the Real Problem *13-3*
1.3. Analyze the Problem *13-3*
1.4. Design a Solution Search Strategy *13-3*
1.5. Problem Solving Using Programs *13-3*
2. Abstraction 13-4
2.1. Levels of Abstraction *13-4*
2.2. Encapsulation *13-4*
2.3. Hierarchy *13-4*
2.4. Alternate Abstractions *13-5*
3. Programming Fundamentals 13-5
3.1. The Programming Process *13-5*
3.2. Programming Paradigms *13-5*
4. Programming Language Basics 13-6
4.1. Programming Language Overview *13-6*
4.2. Syntax and Semantics of Programming Languages *13-6*
4.3. Low-Level Programming Languages *13-7*
4.4. High-Level Programming Languages *13-7*
4.5. Declarative vs. Imperative Programming Languages *13-7*
5. Debugging Tools and Techniques 13-8
5.1. Types of Errors *13-8*
5.2. Debugging Techniques *13-8*
5.3. Debugging Tools *13-8*
6. Data Structure and Representation 13-9
6.1. Data Structure Overview *13-9*
6.2. Types of Data Structure *13-9*
6.3. Operations on Data Structures *13-9*
7. Algorithms and Complexity 13-10
7.1. Overview of Algorithms *13-10*
7.2. Attributes of Algorithms *13-10*
7.3. Algorithmic Analysis *13-10*
7.4. Algorithmic Design Strategies *13-11*
7.5. Algorithmic Analysis Strategies *13-11*
8. Basic Concept of a System 13-11
8.1. Emergent System Properties *13-11*

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

8.2. Systems Engineering *13-12*
8.3. Overview of a Computer System *13-12*
9. Computer Organization 13-13
9.1. Computer Organization Overview *13-13*
9.2. Digital Systems *13-13*
9.3. Digital Logic *13-13*
9.4. Computer Expression of Data *13-13*
9.5. The Central Processing Unit (CPU) *13-14*
9.6. Memory System Organization *13-14*
9.7. Input and Output (I/O) *13-14*
10. Compiler Basics 13-15
10.1. Compiler/Interpreter Overview *13-15*
10.2. Interpretation and Compilation *13-15*
10.3. The Compilation Process *13-15*
11. Operating Systems Basics 13-16
11.1. Operating Systems Overview *13-16*
11.2. Tasks of an Operating System *13-16*
11.3. Operating System Abstractions *13-17*
11.4. Operating Systems Classification *13-17*
12. Database Basics and Data Management 13-17
12.1. Entity and Schema *13-18*
12.2. Database Management Systems (DBMS) *13-18*
12.3. Database Query Language *13-18*
12.4. Tasks of DBMS Packages *13-18*
12.5. Data Management *13-19*
12.6. Data Mining *13-19*
13. Network Communication Basics 13-19
13.1. Types of Network *13-19*
13.2. Basic Network Components *13-19*
13.3. Networking Protocols and Standards *13-20*
13.4. The Internet *13-20*
13.5. Internet of Things *13-20*
13.6. Virtual Private Network (VPN) *13-21*
14. Parallel and Distributed Computing 13-21
14.1. Parallel and Distributed Computing Overview *13-21*
14.2. Difference between Parallel and Distributed Computing *13-21*
14.3. Parallel and Distributed Computing Models *13-21*
14.4. Main Issues in Distributed Computing *13-22*
15. Basic User Human Factors 13-22
15.1. Input and Output *13-22*
15.2. Error Messages *13-23*
15.3. Software Robustness *13-23*
16. Basic Developer Human Factors 13-23
16.1. Structure *13-24*
16.2. Comments *13-24*
17. Secure Software Development and Maintenance 13-24
17.1. Software Requirements Security *13-24*
17.2. Software Design Security *13-25*
17.3. Software Construction Security *13-25*
17.4. Software Testing Security *13-25*

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

17.5. Build Security into Software Engineering Process *13-25*
17.6. Software Security Guidelines *13-25*
Matrix of Topics vs. Reference Material 13-27

Chapter 14: Mathematical Foundations **14-1**
1. Set, Relations, Functions 14-1
1.1. Set Operations *14-2*
1.2. Properties of Set *14-3*
1.3. Relation and Function *14-4*
2. Basic Logic 14-5
2.1. Propositional Logic *14-5*
2.2. Predicate Logic *14-5*
3. Proof Techniques 14-6
3.1. Methods of Proving Theorems *14-6*
4. Basics of Counting 14-7
5. Graphs and Trees 14-8
5.1. Graphs *14-8*
5.2. Trees *14-10*
6. Discrete Probability 14-13
7. Finite State Machines 14-14
8. Grammars 14-15
8.1. Language Recognition *14-16*
9. Numerical Precision, Accuracy, and Errors 14-17
10. Number Theory 14-18
10.1. Divisibility *14-18*
10.2. Prime Number, GCD *14-19*
11. Algebraic Structures 14-19
11.1. Group *14-19*
11.2. Rings *14-20*
Matrix of Topics vs. Reference Material 14-21

Chapter 15: Engineering Foundations **15-1**
1. Empirical Methods and Experimental Techniques 15-1
1.1. Designed Experiment *15-1*
1.2. Observational Study *15-2*
1.3. Retrospective Study *15-2*
2. Statistical Analysis 15-2
2.1. Unit of Analysis (Sampling Units), Population, and Sample *15-2*
2.2. Concepts of Correlation and Regression *15-5*
3. Measurement 15-5
3.1. Levels (Scales) of Measurement *15-6*
3.2. Direct and Derived Measures *15-7*
3.3. Reliability and Validity *15-8*
3.4. Assessing Reliability *15-8*
4. Engineering Design 15-8
4.1. Engineering Design in Engineering Education *15-8*
4.2. Design as a Problem Solving Activity *15-9*
4.3. Steps Involved in Engineering Design *15-9*
5. Modeling, Simulation, and Prototyping 15-10
5.1. Modeling *15-10*

AVA Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

5.2. Simulation *15-11*
5.3. Prototyping *15-11*
6. Standards 15-12
7. Root Cause Analysis 15-12
7.1. Techniques for Conducting Root Cause Analysis *15-13*
Matrix of Topics vs. Reference Material 15-14

Appendix A: Knowledge Area Description Specifications **A-1**

Appendix B: IEEE and ISO/IEC Standards Supporting the Software Engineering Body of Knowledge (SWEBOK) **B-1**

Appendix C: Consolidated Reference List **C-1**

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

FOREWORD

Every profession is based on a body of knowledge, although that knowledge is not always defined in a concise manner. In cases where no formality exists, the body of knowledge is "generally recognized" by practitioners and may be codified in a variety of ways for a variety of different uses. But in many cases, a guide to a body of knowledge is formally documented, usually in a form that permits it to be used for such purposes as development and accreditation of academic and training programs, certification of specialists, or professional licensing. Generally, a professional society or similar body maintains stewardship of the formal definition of a body of knowledge.

During the past forty-five years, software engineering has evolved from a conference catchphrase into an engineering profession, characterized by 1) a professional society, 2) standards that specify generally accepted professional practices, 3) a code of ethics, 4) conference proceedings, 5) textbooks, 6) curriculum guidelines and curricula, 7) accreditation criteria and accredited degree programs, 8) certification and licensing, and 9) this Guide to the Body of Knowledge.

In this *Guide to the Software Engineering Body of Knowledge*, the IEEE Computer Society presents a revised and updated version of the body of knowledge formerly documented as SWEBOK 2004; this revised and updated version is denoted SWEBOK V3. This work is in partial fulfillment of the Society's responsibility to promote the advancement of both theory and practice for the profession of software engineering.

It should be noted that this *Guide* does not present the entire the body of knowledge for software engineering but rather serves as a guide to the body of knowledge that has been developed over more than four decades. The software engineering body of knowledge is constantly evolving. Nevertheless, this *Guide* constitutes a valuable characterization of the software engineering profession.

In 1958, John Tukey, the world-renowned statistician, coined the term *software*. The term software engineering was used in the title of a NATO conference held in Germany in 1968. The IEEE Computer Society first published its *Transactions on Software Engineering* in 1972, and a committee for developing software engineering standards was established within the IEEE Computer Society in 1976.

In 1990, planning was begun for an international standard to provide an overall view of software engineering. The standard was completed in 1995 with designation ISO/IEC 12207 and given the title of *Standard for Software Life Cycle Processes*. The IEEE version of 12207 was published in 1996 and provided a major foundation for the body of knowledge captured in SWEBOK 2004. The current version of 12207 is designated as ISO/IEC 12207:2008 and IEEE 12207-2008; it provides the basis for this SWEBOK V3.

This *Guide to the Software Engineering Body of Knowledge* is presented to you, the reader, as a mechanism for acquiring the knowledge you need in your lifelong career development as a software engineering professional.

Dick Fairley, Chair
Software and Systems Engineering Committee
IEEE Computer Society

Don Shafer, Vice President
Professional Activities Board
IEEE Computer Society

Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

FOREWORD TO THE 2004 EDITION

In this *Guide*, the IEEE Computer Society establishes for the first time a baseline for the body of knowledge for the field of software engineering, and the work partially fulfills the Society's responsibility to promote the advancement of both theory and practice in this field. In so doing, the Society has been guided by the experience of disciplines with longer histories but was not bound either by their problems or their solutions.

It should be noted that the *Guide* does not purport to define the body of knowledge but rather to serve as a compendium and guide to the body of knowledge that has been developing and evolving over the past four decades. Furthermore, this body of knowledge is not static. The *Guide* must, necessarily, develop and evolve as software engineering matures. It nevertheless constitutes a valuable element of the software engineering infrastructure.

In 1958, John Tukey, the world-renowned statistician, coined the term *software*. The term *software engineering* was used in the title of a NATO conference held in Germany in 1968. The IEEE Computer Society first published its *Transactions on Software Engineering* in 1972. The committee established within the IEEE Computer Society for developing software engineering standards was founded in 1976.

The first holistic view of software engineering to emerge from the IEEE Computer Society resulted from an effort led by Fletcher Buckley to develop IEEE standard 730 for software quality assurance, which was completed in 1979. The purpose of IEEE Std. 730 was to provide uniform, minimum acceptable requirements for preparation and content of software quality assurance plans. This standard was influential in completing the developing standards in the following topics: configuration management, software testing, software requirements, software design, and software verification and validation.

During the period 1981–1985, the IEEE Computer Society held a series of workshops concerning the application of software engineering standards. These workshops involved practitioners sharing their experiences with existing standards. The workshops also held sessions on planning for future standards, including one involving measures and metrics for software engineering products and processes. The planning also resulted in IEEE Std. 1002, *Taxonomy of Software Engineering Standards* (1986), which provided a new, holistic view of software engineering. The standard describes the form and content of a software engineering standards taxonomy. It explains the various types of software engineering standards, their functional and external relationships, and the role of various functions participating in the software life cycle.

In 1990, planning for an international standard with an overall view was begun. The planning focused on reconciling the software process views from IEEE Std. 1074 and the revised US DoD standard 2167A. The revision was eventually published as DoD Std. 498. The international standard was completed in 1995 with designation, ISO/IEC 12207, and given the title of *Standard for Software Life Cycle Processes*. Std. ISO/IEC 12207 provided a major point of departure for the body of knowledge captured in this book.

It was the IEEE Computer Society Board of Governors' approval of the motion put forward in May 1993 by Fletcher Buckley which resulted in the writing of this book. The Association for Computing Machinery (ACM) Council approved a related motion in August 1993. The two motions led to a joint committee under the leadership of Mario Barbacci and Stuart Zweben who served as cochairs. The mission statement of the joint committee was "To establish the appropriate sets(s) of criteria and norms for professional practice of software engineering upon which industrial decisions, professional certification, and educational curricula can be based." The steering committee organized task forces in the following areas:

1. Define Required Body of Knowledge and Recommended Practices.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

2. Define Ethics and Professional Standards.
3. Define Educational Curricula for undergraduate, graduate, and continuing education.

This book supplies the first component: required body of knowledge and recommend practices.

The code of ethics and professional practice for software engineering was completed in 1998 and approved by both the ACM Council and the IEEE Computer Society Board of Governors. It has been adopted by numerous corporations and other organizations and is included in several recent textbooks.

The educational curriculum for undergraduates is being completed by a joint effort of the IEEE Computer Society and the ACM and is expected to be completed in 2004.

Every profession is based on a body of knowledge and recommended practices, although they are not always defined in a precise manner. In many cases, these are formally documented, usually in a form that permits them to be used for such purposes as accreditation of academic programs, development of education and training programs, certification of specialists, or professional licensing. Generally, a professional society or related body maintains custody of such a formal definition. In cases where no such formality exists, the body of knowledge and recommended practices are "generally recognized" by practitioners and may be codified in a variety of ways for different uses.

It is hoped that readers will find this book useful in guiding them toward the knowledge and resources they need in their lifelong career development as software engineering professionals.

The book is dedicated to Fletcher Buckley in recognition of his commitment to promoting software engineering as a professional discipline and his excellence as a software engineering practitioner in radar applications.

Leonard L. Tripp, IEEE Fellow 2003
Chair, Professional Practices Committee, IEEE Computer Society (2001–2003)

Chair, Joint IEEE Computer Society and ACM Steering Committee for the Establishment of Software Engineering as a Profession (1998–1999)

Chair, Software Engineering Standards Committee, IEEE Computer Society (1992–1998)

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

EDITORS

Pierre Bourque, Department of Software and IT Engineering, École de technologie supérieure (ÉTS), Canada, pierre.bourque@etsmtl.ca
Richard E. (Dick) Fairley, Software and Systems Engineering Associates (S2EA), USA, dickfairley@gmail.com

COEDITORS

Alain Abran, Department of Software and IT Engineering, École de technologie supérieure (ÉTS), Canada, alain.abran@etsmtl.ca
Juan Garbajosa, Universidad Politecnica de Madrid (Technical University of Madrid, UPM), Spain, juan.garbajosa@upm.es
Gargi Keeni, Tata Consultancy Services, India, gargi@ieee.org
Beijun Shen, School of Software, Shanghai Jiao Tong University, China, bjshen@sjtu.edu.cn

CONTRIBUTING EDITORS

The following persons contributed to editing the *SWEBOK Guide* V3:
Don Shafer
Linda Shafer
Mary Jane Willshire
Kate Guillemette

CHANGE CONTROL BOARD

The following persons served on the *SWEBOK Guide* V3 Change Control Board:
Pierre Bourque
Richard E. (Dick) Fairley, Chair
Dennis Frailey
Michael Gayle
Thomas Hilburn
Paul Joannou
James W. Moore
Don Shafer
Steve Tockey

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

KNOWLEDGE AREA EDITORS

Software Requirements

Gerald Kotonya, School of Computing and Communications, Lancaster University, UK, gerald@comp.lancs.ac.uk
Peter Sawyer, School of Computing and Communications, Lancaster University, UK, sawyer@comp.lancs.ac.uk

Software Design

Yanchun Sun, School of Electronics Engineering and Computer Science, Peking University, China, sunyc@pku.edu.cn

Software Construction

Xin Peng, Software School, Fudan University, China, pengxin@fudan.edu.cn

Software Testing

Antonia Bertolino, ISTI-CNR, Italy, antonia.bertolino@isti.cnr.it
Eda Marchetti, ISTI-CNR, Italy, eda.marchetti@isti.cnr.it

Software Maintenance

Alain April, École de technologie supérieure (ÉTS), Canada, alain.april@etsmtl.ca
Mira Kajko-Mattsson, School of Information and Communication Technology, KTH Royal Institute of Technology, mekm2@kth.se

Software Configuration Management

Roger Champagne, École de technologie supérieure (ÉTS), Canada, roger.champagne@etsmtl.ca
Alain April, École de technologie supérieure (ÉTS), Canada, alain.april@etsmtl.ca

Software Engineering Management

James McDonald, Department of Computer Science and Software Engineering, Monmouth University, USA, jamesmc@monmouth.edu

Software Engineering Process

Annette Reilly, Lockheed Martin Information Systems & Global Solutions, USA, annette.reilly@computer.org
Richard E. Fairley, Software and Systems Engineering Associates (S2EA), USA, dickfairley@gmail.com

Software Engineering Models and Methods

Michael F. Siok, Lockheed Martin Aeronautics Company, USA, mike.f.siok@lmco.com

Software Quality

J. David Blaine, USA, jdavidblaine@gmail.com
Durba Biswas, Tata Consultancy Services, India, durba.biswas@tcs.com

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

Software Engineering Professional Practice
Aura Sheffield, USA, arsheff@acm.org
Hengming Zou, Shanghai Jiao Tong University, China, zou@sjtu.edu.cn

Software Engineering Economics
Christof Ebert, Vector Consulting Services, Germany, christof.ebert@vector.com

Computing Foundations
Hengming Zou, Shanghai Jiao Tong University, China, zou@sjtu.edu.cn

Mathematical Foundations
Nabendu Chaki, University of Calcutta, India, nabendu@ieee.org

Engineering Foundations
Amitava Bandyopadhayay, Indian Statistical Institute, India, bamitava@isical.ac.in
Mary Jane Willshire, Software and Systems Engineering Associates (S2EA), USA, mj.fairley@gmail.com

Appendix B: IEEE and ISO/IEC Standards Supporting SWEBOK
James W. Moore, USA, James.W.Moore@ieee.org

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

KNOWLEDGE AREA EDITORS OF PREVIOUS SWEBOK VERSIONS

The following persons served as Associate Editors for either the Trial version published in 2001 or for the 2004 version.

Software Requirements

Peter Sawyer, Computing Department, Lancaster University, UK
Gerald Kotonya, Computing Department, Lancaster University, UK

Software Design

Guy Tremblay, Département d'informatique, UQAM, Canada

Software Construction

Steve McConnell, Construx Software, USA
Terry Bollinger, the MITRE Corporation, USA
Philippe Gabrini, Département d'informatique, UQAM, Canada
Louis Martin, Département d'informatique, UQAM, Canada

Software Testing

Antonia Bertolino, ISTI-CNR, Italy
Eda Marchetti, ISTI-CNR, Italy

Software Maintenance

Thomas M. Pigoski, Techsoft Inc., USA
Alain April, École de technologie supérieure, Canada

Software Configuration Management

John A. Scott, Lawrence Livermore National Laboratory, USA
David Nisse, USA

Software Engineering Management

Dennis Frailey, Raytheon Company, USA
Stephen G. MacDonell, Auckland University of Technology, New Zealand
Andrew R. Gray, University of Otago, New Zealand

Software Engineering Process

Khaled El Emam, served while at the Canadian National Research Council, Canada

Software Engineering Tools and Methods

David Carrington, School of Information Technology and Electrical Engineering,
The University of Queensland, Australia

Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

Software Quality

Alain April, École de technologie supérieure, Canada
Dolores Wallace, retired from the National Institute of Standards and Technology, USA
Larry Reeker, NIST, USA

References Editor

Marc Bouisset, Département d'informatique, UQAM

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

REVIEW TEAM

The people listed below participated in the public review process of *SWEBOK Guide* V3. Membership of the IEEE Computer Society was not a requirement to participate in this review process, and membership information was not requested from reviewers. Over 1500 individual comments were collected and duly adjudicated.

Carlos C. Amaro, USA
Mark Ardis, USA
Mora-Soto Arturo, Spain
Ohad Barzilay, Israel
Gianni Basaglia, Italy
Denis J. Bergquist, USA
Alexander Bogush, UK
Christopher Bohn, USA
Steve Bollweg, USA
Reto Bonderer, Switzerland
Alexei Botchkarev, Canada
Pieter Botman, Canada
Robert Bragner, USA
Kevin Brune, USA
Ogihara Bryan, USA
Luigi Buglione, Italy
Rick Cagle, USA
Barbara Canody, USA
Rogerio A. Carvalho, Brazil
Daniel Cerys, USA
Philippe Cohard, France
Ricardo Colomo-Palacios, Spain
Mauricio Coria, Argentina
Marek Cruz, UK
Stephen Danckert, USA
Bipul K. Das, Canada
James D. Davidson, USA
Jon Dehn, USA
Lincoln P. Djang, USA
Andreas Doblander, Austria
Yi-Ben Doo, USA
Scott J. Dougherty, UK
Regina DuBord, USA
Fedor Dzerzhinskiy, Russia
Ann M. Eblen, Australia
David M. Endres, USA
Marilyn Escue, USA
Varuna Eswer, India
Istvan Fay, Hungary
Jose L. Fernandez-Sanchez, Spain
Dennis J. Frailey, USA
Tihana Galinac Grbac, Croatia
Colin Garlick, New Zealand
Garth J.G. Glynn, UK
Jill Gostin, USA
Christiane Gresse von Wangenheim, Brazil
Thomas Gust, USA
H.N. Mok, Singapore
Jon D. Hagar, USA
Anees Ahmed Haidary, India
Duncan Hall, New Zealand
James Hart, USA
Jens H.J. Heidrich, Germany
Rich Hilliard, USA
Bob Hillier, Canada
Norman M. Hines, USA
Dave Hirst, USA
Theresa L. Hunt, USA
Kenneth Ingham, USA
Masahiko Ishikawa, Japan
Michael A. Jablonski, USA
G. Jagadeesh, India
Sebastian Justicia, Spain
Umut Kahramankaptan, Belgium
Pankaj Kamthan, Canada
Perry Kapadia, USA
Tarig A. Khalid, Sudan
Michael K.A. Klaes, Germany
Maged Koshty, Egypt
Claude C. Laporte, Canada
Dong Li, China
Ben Linders, Netherlands
Claire Lohr, USA
Vladimir Mandic, Serbia
Matt Mansell, New Zealand
John Marien, USA

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

Stephen P. Masticola, USA
Nancy Mead, USA
Fuensanta Medina-Dominguez, Spain
Silvia Judith Meles, Argentina
Oscar A. Mondragon, Mexico
David W. Mutschler, USA
Maria Nelson, Brazil
John Noblin, USA
Bryan G. Ogihara, USA
Takehisa Okazaki, Japan
Hanna Oktaba, Mexico
Chin Hwee Ong, Hong Kong
Venkateswar Oruganti, India
Birgit Penzenstadler, Germany
Larry Peters, USA
S.K. Pillai, India
Vaclav Rajlich, USA
Kiron Rao, India
Luis Reyes, USA
Hassan Reza, USA
Steve Roach, USA
Teresa L. Roberts, USA
Dennis Robi, USA
Warren E. Robinson, USA
Jorge L. Rodriguez, USA
Alberto C. Sampaio, Portugal
Ed Samuels, USA
Maria-Isabel Sanchez-Segura, Spain
Vineet Sawant, USA
R. Schaaf, USA
James C. Schatzman, USA
Oscar A. Schivo, Argentina
Florian Schneider, Germany
Thom Schoeffling, USA
Reinhard Schrage, Germany
Neetu Sethia, India
Cindy C. Shelton, USA
Alan Shepherd, Germany
Katsutoshi Shintani, Japan
Erik Shreve, USA
Jaguaraci Silva, Brazil
M. Somasundaram, India
Peraphon Sophatsathit, Thailand
John Standen, UK
Joyce Statz, USA
Perdita P. Stevens, UK
David Struble, USA
Ohno Susumu, Japan
Urcun Tanik, USA
Talin Tasciyan, USA
J. Barrie Thompson, UK
Steve Tockey, USA
Miguel Eduardo Torres Moreno, Colombia
Dawid Trawczynski, USA
Adam Trendowicz, Germany
Norio Ueno, Japan
Cenk Uyan, Turkey
Chandra Sekar Veerappan, Singapore
Oruganti Venkateswar, India
Jochen Vogt, Germany
Hironori Washizaki, Japan
Ulf Westermann, Germany
Don Wilson, USA
Aharon Yadin, Israel
Hong Zhou, UK

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

ACKNOWLEDGEMENTS

Funding for the development of *SWEBOK Guide* V3 has been provided by the IEEE Computer Society. The editors and coeditors appreciate the important work performed by the KA editors and the contributing editors as well as by the the members of the Change Control Board. The editorial team must also acknowledge the indispensable contribution of reviewers.

The editorial team also wishes to thank the following people who contributed to the project in various ways: Pieter Botman, Evan Butterfield, Carine Chauny, Pierce Gibbs, Diane Girard, John Keppler, Dorian McClenahan, Kenza Meridji, Samuel Redwine, Annette Reilly, and Pam Thompson.

Finally, there are surely other people who have contributed to this *Guide*, either directly or indirectly, whose names we have inadvertently omitted. To those people, we offer our tacit appreciation and apologize for having omitted explicit recognition.

IEEE COMPUTER SOCIETY PRESIDENTS

Dejan Milojicic, 2014 President
David Alan Grier, 2013 President
Thomas Conte, 2015 President

PROFESSIONAL ACTIVITIES BOARD, 2013 MEMBERSHIP

Donald F. Shafer, Chair
Pieter Botman, CSDP
Pierre Bourque
Richard Fairley, CSDP
Dennis Frailey
S. Michael Gayle
Phillip Laplante, CSDP
Jim Moore, CSDP
Linda Shafer, CSDP
Steve Tockey, CSDP
Charlene “Chuck” Walrad

Documents
Document Sharing Hub
© ISO/IEC 2016 – All rights reserved
Full document details: ISO_IEC TR 19759_2015

MOTIONS REGARDING THE APPROVAL OF SWEBOK GUIDE V3.0

The *SWEBOK Guide* V3.0 was submitted to ballot by verified IEEE Computer Society members in November 2013 with the following question: "Do you approve this manuscript of the *SWEBOK Guide* V3.0 to move forward to formatting and publication?"

The results of this ballot were 259 Yes votes and 5 No votes.

The following motion was unanimously adopted by the Professional Activities Board of the IEEE Computer Society in December 2013:

> *The Professional Activities Board of the IEEE Computer Society finds that the* Guide to the Software Engineering Body of Knowledge *Version 3.0 has been successfully completed; and endorses the* Guide to the Software Engineering Body of Knowledge *Version 3.0 and commends it to the IEEE Computer Society Board of Governors for their approval.*

The following motion was adopted by the IEEE Computer Society Board of Governors in December 2013:

> *MOVED, that the Board of Governors of the IEEE Computer Society approves Version 3.0 of the* Guide to the Software Engineering Body of Knowledge *and authorizes the Chair of the Professional Activities Board to proceed with printing.*

MOTIONS REGARDING THE APPROVAL OF SWEBOK GUIDE 2004 VERSION

The following motion was unanimously adopted by the Industrial Advisory Board of the *SWEBOK Guide* project in February 2004:

> *The Industrial Advisory Board finds that the Software Engineering Body of Knowledge project initiated in 1998 has been successfully completed; and endorses the 2004 Version of the* Guide to the SWEBOK *and commends it to the IEEE Computer Society Board of Governors for their approval.*

The following motion was adopted by the IEEE Computer Society Board of Governors in February 2004:

> *MOVED, that the Board of Governors of the IEEE Computer Society approves the 2004 Edition of the* Guide to the Software Engineering Body of Knowledge *and authorizes the Chair of the Professional Practices Committee to proceed with printing.*

Please also note that the 2004 edition of the *Guide to the Software Engineering Body of Knowledge* was submitted by the IEEE Computer Society to ISO/IEC without any change and was recognized as Technical Report ISO/IEC TR 19759:2005.

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

INTRODUCTION TO THE GUIDE

KA	Knowledge Area
SWEBOK	Software Engineering Body of Knowledge

Publication of the 2004 version of this *Guide to the Software Engineering Body of Knowledge* (SWEBOK 2004) was a major milestone in establishing software engineering as a recognized engineering discipline. The goal in developing this update to SWEBOK is to improve the currency, readability, consistency, and usability of the *Guide*.

All knowledge areas (KAs) have been updated to reflect changes in software engineering since publication of SWEBOK 2004. Four new foundation KAs and a Software Engineering Professional Practices KA have been added. The Software Engineering Tools and Methods KA has been revised as Software Engineering Models and Methods. Software engineering tools is now a topic in each of the KAs. Three appendices provide the specifications for the KA description, an annotated set of relevant standards for each KA, and a listing of the references cited in the *Guide*.

This *Guide*, written under the auspices of the Professional Activities Board of the IEEE Computer Society, represents a next step in the evolution of the software engineering profession.

WHAT IS SOFTWARE ENGINEERING?

ISO/IEC/IEEE Systems and Software Engineering Vocabulary (SEVOCAB) defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software)."[1]

WHAT ARE THE OBJECTIVES OF THE SWEBOK GUIDE?

The *Guide* should not be confused with the Body of Knowledge itself, which exists in the published literature. The purpose of the *Guide* is to describe the portion of the Body of Knowledge that is generally accepted, to organize that portion, and to provide topical access to it.

The *Guide to the Software Engineering Body of Knowledge* (*SWEBOK Guide*) was established with the following five objectives:

1. To promote a consistent view of software engineering worldwide
2. To specify the scope of, and clarify the place of software engineering with respect to other disciplines such as computer science, project management, computer engineering, and mathematics
3. To characterize the contents of the software engineering discipline
4. To provide a topical access to the Software Engineering Body of Knowledge
5. To provide a foundation for curriculum development and for individual certification and licensing material

The first of these objectives, a consistent worldwide view of software engineering, was supported by a development process which engaged approximately 150 reviewers from 33 countries. More information regarding the development process can be found on the website (www.swebok.org). Professional and learned societies and public agencies involved in software engineering were contacted, made aware of this project to update SWEBOK, and invited to participate in the review process. KA editors were recruited from North America, the Pacific Rim, and Europe. Presentations on the project were made at various international venues.

The second of the objectives, the desire to specify the scope of software engineering, motivates the fundamental organization of the Guide. The material that is recognized as being within this discipline is organized into the fifteen KAs listed in Table I.1. Each of these KAs is treated in a chapter in this *Guide*.

1 See www.computer.org/sevocab.

© ISO/IEC 2016 – All rights reserved

Documents Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations

In specifying scope, it is also important to identify the disciplines that intersect with software engineering. To this end, SWEBOK V3 also recognizes seven related disciplines, listed in Table I.2. Software engineers should, of course, have knowledge of material from these disciplines (and the KA descriptions in this *Guide* may make reference to them). It is not, however, an objective of the *SWEBOK Guide* to characterize the knowledge of the related disciplines.

Table I.2. Related Disciplines
Computer Engineering
Computer Science
General Management
Mathematics
Project Management
Quality Management
Systems Engineering

The relevant elements of computer science and mathematics are presented in the Computing Foundations and Mathematical Foundations KAs of the *Guide* (Chapters 13 and 14).

HIERARCHICAL ORGANIZATION

The organization of the KA chapters supports the third of the project's objectives—a characterization of the contents of software engineering. The detailed specifications provided by the project's editorial team to the associate editors regarding the contents of the KA descriptions can be found in Appendix A.

The *Guide* uses a hierarchical organization to decompose each KA into a set of topics with recognizable labels. A two (sometime three) level breakdown provides a reasonable way to find topics of interest. The *Guide* treats the selected topics in a manner compatible with major schools of thought and with breakdowns generally found in industry and in software engineering literature and standards. The breakdowns of topics do not presume particular application domains, business uses, management philosophies, development methods, and so forth. The extent of each topic's description is only that needed to understand the generally accepted nature of the topics and for the reader to successfully find reference material; the Body of Knowledge is found in the reference materials themselves, not in the *Guide*.

REFERENCE MATERIAL AND MATRIX

To provide topical access to the knowledge—the fourth of the project's objectives—the *Guide* identifies authoritative reference material for each KA. Appendix C provides a Consolidated Reference List for the *Guide*. Each KA includes relevant references from the Consolidated Reference List and also includes a matrix relating the reference material to the included topics.

It should be noted that the *Guide* does not attempt to be comprehensive in its citations. Much material that is both suitable and excellent is not referenced. Material included in the Consolidated Reference List provides coverage of the topics described.

DEPTH OF TREATMENT

To achieve the SWEBOK fifth objective—providing a foundation for curriculum development,

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

certification, and licensing, the criterion of *generally accepted* knowledge has been applied, to be distinguished from advanced and research knowledge (on the grounds of maturity) and from specialized knowledge (on the grounds of generality of application).

The equivalent term *generally recognized* comes from the Project Management Institute: "Generally recognized means the knowledge and practices described are applicable to most projects most of the time, and there is consensus about their value and usefulness."[2]

However, the terms "generally accepted" or "generally recognized" do not imply that the designated knowledge should be uniformly applied to all software engineering endeavors—each project's needs determine that—but it does imply that competent, capable software engineers should be equipped with this knowledge for potential application. More precisely, generally accepted knowledge should be included in the study material for the software engineering licensing examination that graduates would take after gaining four years of work experience. Although this criterion is specific to the US style of education and does not necessarily apply to other countries, we deem it useful.

STRUCTURE OF THE KA DESCRIPTIONS

The KA descriptions are structured as follows.

In the introduction, a brief definition of the KA and an overview of its scope and of its relationship with other KAs are presented.

The breakdown of topics in each KA constitutes the core the KA description, describing the decomposition of the KA into subareas, topics, and sub-topics. For each topic or subtopic, a short description is given, along with one or more references.

The reference material was chosen because it is considered to constitute the best presentation of the knowledge relative to the topic. A matrix links the topics to the reference material.

The last part of each KA description is the list of recommended references and (optionally) further readings. Relevant standards for each KA are presented in Appendix B of the *Guide*.

APPENDIX A. KA DESCRIPTION SPECIFICATIONS

Appendix A describes the specifications provided by the editorial team to the associate editors for the content, recommended references, format, and style of the KA descriptions.

APPENDIX B. ALLOCATION OF STANDARDS TO KAS

Appendix B is an annotated list of the relevant standards, mostly from the IEEE and the ISO, for each of the KAs of the *SWEBOK Guide*.

APPENDIX C. CONSOLIDATED REFERENCE LIST

Appendix C contains the consolidated list of recommended references cited in the KAs (these references are marked with an asterisk (*) in the text).

2 *A Guide to the Project Management Body of Knowledge*, 5th ed., Project Management Institute, 2013; www.pmi.org.

© ISO/IEC 2016 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO_IEC TR 19759_2015

CHAPTER 1

SOFTWARE REQUIREMENTS

ACRONYMS

CIA	Confidentiality, Integrity, and Availability
DAG	Directed Acyclic Graph
FSM	Functional Size Measurement
INCOSE	International Council on Systems Engineering
UML	Unified Modeling Language
SysML	Systems Modeling Language

INTRODUCTION

The Software Requirements knowledge area (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements as well as the management of requirements during the whole life cycle of the software product. It is widely acknowledged amongst researchers and industry practitioners that software projects are critically vulnerable when the requirements-related activities are poorly performed.

Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem.

The term "requirements engineering" is widely used in the field to denote the systematic handling of requirements. For reasons of consistency, the term "engineering" will not be used in this KA other than for software engineering per se.

For the same reason, "requirements engineer," a term which appears in some of the literature, will not be used either. Instead, the term "software engineer" or, in some specific cases, "requirements specialist" will be used, the latter where the role in question is usually performed by an individual other than a software engineer. This does not imply, however, that a software engineer could not perform the function.

A risk inherent in the proposed breakdown is that a waterfall-like process may be inferred. To guard against this, topic 2, Requirements Process, is designed to provide a high-level overview of the requirements process by setting out the resources and constraints under which the process operates and which act to configure it.

An alternate decomposition could use a product-based structure (system requirements, software requirements, prototypes, use cases, and so on). The process-based breakdown reflects the fact that the requirements process, if it is to be successful, must be considered as a process involving complex, tightly coupled activities (both sequential and concurrent), rather than as a discrete, one-off activity performed at the outset of a software development project.

The Software Requirements KA is related closely to the Software Design, Software Testing, Software Maintenance, Software Configuration Management, Software Engineering Management, Software Engineering Process, Software Engineering Models and Methods, and Software Quality KAs.

BREAKDOWN OF TOPICS FOR SOFTWARE REQUIREMENTS

The breakdown of topics for the Software Requirements KA is shown in Figure 1.1.

1. Software Requirements Fundamentals

[1*, c4, c4s1, c10s1, c10s4] [2*, c1, c6, c12]

1.1. Definition of a Software Requirement

At its most basic, a software requirement is a property that must be exhibited by something in

© ISO/IEC 2016 – All rights reserved

AVA Documents Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

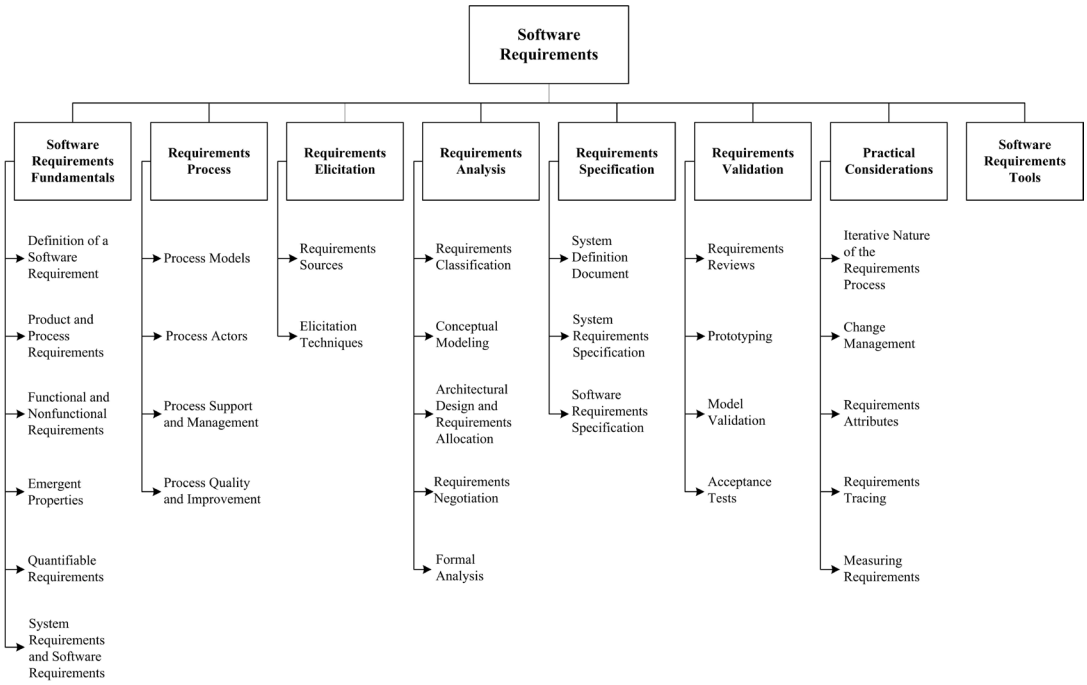

Figure 1.1. Breakdown of Topics for the Software Requirements KA

order to solve some problem in the real world. It may aim to automate part of a task for someone to support the business processes of an organization, to correct shortcomings of existing software, or to control a device—to name just a few of the many problems for which software solutions are possible. The ways in which users, business processes, and devices function are typically complex. By extension, therefore, the requirements on particular software are typically a complex combination from various people at different levels of an organization, and who are in one way or another involved or connected with this feature from the environment in which the software will operate.

An essential property of all software requirements is that they be verifiable as an individual feature as a functional requirement or at the system level as a nonfunctional requirement. It may be difficult or costly to verify certain software requirements. For example, verification of the throughput requirement on a call center may necessitate the development of simulation software. Software requirements, software testing, and quality personnel must ensure that the requirements can be verified within available resource constraints.

Requirements have other attributes in addition to behavioral properties. Common examples include a priority rating to enable tradeoffs in the face of finite resources and a status value to enable project progress to be monitored. Typically, software requirements are uniquely identified so that they can be subjected to software configuration management over the entire life cycle of the feature and of the software.

1.2. Product and Process Requirements

A product requirement is a need or constraint on the software to be developed (for example, "The software shall verify that a student meets all prerequisites before he or she registers for a course").

A process requirement is essentially a constraint on the development of the software (for example, "The software shall be developed using a RUP process").

Some software requirements generate implicit process requirements. The choice of verification

© ISO/IEC 2016 – All rights reserved

Documents Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

technique is one example. Another might be the use of particularly rigorous analysis techniques (such as formal specification methods) to reduce faults that can lead to inadequate reliability. Process requirements may also be imposed directly by the development organization, their customer, or a third party such as a safety regulator.

1.3. Functional and Nonfunctional Requirements

Functional requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal. They are sometimes known as capabilities or features. A functional requirement can also be described as one for which a finite set of test steps can be written to validate its behavior.

Nonfunctional requirements are the ones that act to constrain the solution. Nonfunctional requirements are sometimes known as constraints or quality requirements. They can be further classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, interoperability requirements or one of many other types of software requirements (see Models and Quality Characteristics in the Software Quality KA).

1.4. Emergent Properties

Some requirements represent emergent properties of software—that is, requirements that cannot be addressed by a single component but that depend on how all the software components interoperate. The throughput requirement for a call center would, for example, depend on how the telephone system, information system, and the operators all interacted under actual operating conditions. Emergent properties are crucially dependent on the system architecture.

1.5. Quantifiable Requirements

Software requirements should be stated as clearly and as unambiguously as possible, and, where appropriate, quantitatively. It is important to avoid vague and unverifiable requirements that depend for their interpretation on subjective judgment ("the software shall be reliable"; "the software shall be user-friendly"). This is particularly important for nonfunctional requirements. Two examples of quantified requirements are the following: a call center's software must increase the center's throughput by 20%; and a system shall have a probability of generating a fatal error during any hour of operation of less than $1 * 10^{-8}$. The throughput requirement is at a very high level and will need to be used to derive a number of detailed requirements. The reliability requirement will tightly constrain the system architecture.

1.6. System Requirements and Software Requirements

In this topic, "system" means

> an interacting combination of elements to accomplish a defined objective. These include hardware, software, firmware, people, information, techniques, facilities, services, and other support elements,

as defined by the International Council on Software and Systems Engineering (INCOSE) [3].

System requirements are the requirements for the system as a whole. In a system containing software components, *software* requirements are derived from system requirements.

This KA defines "user requirements" in a restricted way, as the requirements of the system's customers or end users. System requirements, by contrast, encompass user requirements, requirements of other stakeholders (such as regulatory authorities), and requirements without an identifiable human source.

2. Requirements Process

[1*, c4s4] [2*, c1–4, c6, c22, c23]

This section introduces the software requirements process, orienting the remaining five topics and showing how the requirements process dovetails with the overall software engineering process.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

2.1. Process Models

The objective of this topic is to provide an understanding that the requirements process

- is not a discrete front-end activity of the software life cycle, but rather a process initiated at the beginning of a project that continues to be refined throughout the life cycle;
- identifies software requirements as configuration items and manages them using the same software configuration management practices as other products of the software life cycle processes;
- needs to be adapted to the organization and project context.

In particular, the topic is concerned with how the activities of elicitation, analysis, specification, and validation are configured for different types of projects and constraints. The topic also includes activities that provide input into the requirements process, such as marketing and feasibility studies.

2.2. Process Actors

This topic introduces the roles of the people who participate in the requirements process. This process is fundamentally interdisciplinary, and the requirements specialist needs to mediate between the domain of the stakeholder and that of software engineering. There are often many people involved besides the requirements specialist, each of whom has a stake in the software. The stakeholders will vary across projects, but will always include users/operators and customers (who need not be the same).

Typical examples of software stakeholders include (but are not restricted to) the following:

- Users: This group comprises those who will operate the software. It is often a heterogeneous group involving people with different roles and requirements.
- Customers: This group comprises those who have commissioned the software or who represent the software's target market.
- Market analysts: A mass-market product will not have a commissioning customer, so marketing people are often needed to establish what the market needs and to act as proxy customers.
- Regulators: Many application domains, such as banking and public transport, are regulated. Software in these domains must comply with the requirements of the regulatory authorities.
- Software engineers: These individuals have a legitimate interest in profiting from developing the software by, for example, reusing components in or from other products. If, in this scenario, a customer of a particular product has specific requirements that compromise the potential for component reuse, the software engineers must carefully weigh their own stake against those of the customer. Specific requirements, particularly constraints, may have major impact on project cost or delivery because they either fit well or poorly with the skill set of the engineers. Important tradeoffs among such requirements should be identified.

It will not be possible to perfectly satisfy the requirements of every stakeholder, and it is the software engineer's job to negotiate tradeoffs that are both acceptable to the principal stakeholders and within budgetary, technical, regulatory, and other constraints. A prerequisite for this is that all the stakeholders be identified, the nature of their "stake" analyzed, and their requirements elicited.

2.3. Process Support and Management

This section introduces the project management resources required and consumed by the requirements process. It establishes the context for the first topic (Initiation and Scope Definition) of the Software Engineering Management KA. Its principal purpose is to make the link between the process activities identified in 2.1 and the issues of cost, human resources, training, and tools.

2.4. Process Quality and Improvement

This topic is concerned with the assessment of the quality and improvement of the requirements process. Its purpose is to emphasize the key role the requirements process plays in terms of the

AVA Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

cost and timeliness of a software product and of the customer's satisfaction with it. It will help to orient the requirements process with quality standards and process improvement models for software and systems. Process quality and improvement is closely related to both the Software Quality KA and Software Engineering Process KA, comprising

- requirements process coverage by process improvement standards and models;
- requirements process measures and benchmarking;
- improvement planning and implementation;
- security/CIA improvement/planning and implementation.

3. Requirements Elicitation

[1*, c4s5] [2*, c5, c6, c9]

Requirements elicitation is concerned with the origins of software requirements and how the software engineer can collect them. It is the first stage in building an understanding of the problem the software is required to solve. It is fundamentally a human activity and is where the stakeholders are identified and relationships established between the development team and the customer. It is variously termed "requirements capture," "requirements discovery," and "requirements acquisition."

One of the fundamental principles of a good requirements elicitation process is that of effective communication between the various stakeholders. This communication continues through the entire Software Development Life Cycle (SDLC) process with different stakeholders at different points in time. Before development begins, requirements specialists may form the conduit for this communication. They must mediate between the domain of the software users (and other stakeholders) and the technical world of the software engineer. A set of internally consistent models at different levels of abstraction facilitate communications between software users/stakeholders and software engineers.

A critical element of requirements elicitation is informing the project scope. This involves providing a description of the software being specified and its purpose and prioritizing the deliverables to ensure the customer's most important business needs are satisfied first. This minimizes the risk of requirements specialists spending time eliciting requirements that are of low importance, or those that turn out to be no longer relevant when the software is delivered. On the other hand, the description must be scalable and extensible to accept further requirements not expressed in the first formal lists and compatible with the previous ones as contemplated in recursive methods.

3.1. Requirements Sources

Requirements have many sources in typical software, and it is essential that all potential sources be identified and evaluated. This topic is designed to promote awareness of the various sources of software requirements and of the frameworks for managing them. The main points covered are as follows:

- Goals. The term "goal" (sometimes called "business concern" or "critical success factor") refers to the overall, high-level objectives of the software. Goals provide the motivation for the software but are often vaguely formulated. Software engineers need to pay particular attention to assessing the value (relative to priority) and cost of goals. A feasibility study is a relatively low-cost way of doing this.
- Domain knowledge. The software engineer needs to acquire or have available knowledge about the application domain. Domain knowledge provides the background against which all elicited requirements knowledge must be set in order to understand it. It's a good practice to emulate an ontological approach in the knowledge domain. Relations between relevant concepts within the application domain should be identified.
- Stakeholders (see section 2.2, Process Actors). Much software has proved unsatisfactory because it has stressed the requirements of one group of stakeholders at the expense of others. Hence, the delivered software is difficult to use, or subverts the cultural or political structures of the customer organization. The software engineer needs to identify, represent, and manage

© ISO/IEC 2016 – All rights reserved

AVA Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

the "viewpoints" of many different types of stakeholders.

- Business rules. These are statements that define or constrain some aspect of the structure or the behavior of the business itself. "A student cannot register in next semester's courses if there remain some unpaid tuition fees" would be an example of a business rule that would be a requirement source for a university's course-registration software.
- The operational environment. Requirements will be derived from the environment in which the software will be executed. These may be, for example, timing constraints in real-time software or performance constraints in a business environment. These must be sought out actively because they can greatly affect software feasibility and cost as well as restrict design choices.
- The organizational environment. Software is often required to support a business process, the selection of which may be conditioned by the structure, culture, and internal politics of the organization. The software engineer needs to be sensitive to these since, in general, new software should not force unplanned change on the business process.

3.2. Elicitation Techniques

Once the requirements sources have been identified, the software engineer can start eliciting requirements information from them. Note that requirements are seldom elicited ready-made. Rather, the software engineer elicits information from which he or she formulates requirements. This topic concentrates on techniques for getting human stakeholders to articulate requirements-relevant information. It is a very difficult task and the software engineer needs to be sensitized to the fact that (for example) users may have difficulty describing their tasks, may leave important information unstated, or may be unwilling or unable to cooperate. It is particularly important to understand that elicitation is not a passive activity and that, even if cooperative and articulate stakeholders are available, the software engineer has to work hard to elicit the right information. Many business or technical requirements are tacit or in feedback that has yet to be obtained from end users. The importance of planning, verification, and validation in requirements elicitation cannot be overstated. A number of techniques exist for requirements elicitation; the principal ones are these:

- Interviews. Interviewing stakeholders is a "traditional" means of eliciting requirements. It is important to understand the advantages and limitations of interviews and how they should be conducted.
- Scenarios. Scenarios provide a valuable means for providing context to the elicitation of user requirements. They allow the software engineer to provide a framework for questions about user tasks by permitting "what if" and "how is this done" questions to be asked. The most common type of scenario is the use case description. There is a link here to topic 4.2 (Conceptual Modeling) because scenario notations such as use case diagrams are common in modeling software.
- Prototypes. This technique is a valuable tool for clarifying ambiguous requirements. They can act in a similar way to scenarios by providing users with a context within which they can better understand what information they need to provide. There is a wide range of prototyping techniques—from paper mock-ups of screen designs to beta-test versions of software products—and a strong overlap of their separate uses for requirements elicitation and for requirements validation (see section 6.2, Prototyping). Low fidelity prototypes are often preferred to avoid stakeholder "anchoring" on minor, incidental characteristics of a higher quality prototype that can limit design flexibility in unintended ways.
- Facilitated meetings. The purpose of these meetings is to try to achieve a summative effect, whereby a group of people can bring more insight into their software requirements than by working individually. They can brainstorm and refine ideas that may be difficult to bring to the surface using interviews. Another advantage is that conflicting requirements surface early on in a way that lets the stakeholders recognize where these occur. When it works well, this technique

AVA Documents
Document Sharing Hub

© ISO/IEC 2016 – All rights reserved

Full document details: ISO_IEC TR 19759_2015

may result in a richer and more consistent set of requirements than might otherwise be achievable. However, meetings need to be handled carefully (hence the need for a facilitator) to prevent a situation in which the critical abilities of the team are eroded by group loyalty, or in which requirements reflecting the concerns of a few outspoken (and perhaps senior) people that are favored to the detriment of others.

- Observation. The importance of software context within the organizational environment has led to the adaptation of observational techniques such as ethnography for requirements elicitation. Software engineers learn about user tasks by immersing themselves in the environment and observing how users perform their tasks by interacting with each other and with software tools and other resources. These techniques are relatively expensive but also instructive because they illustrate that many user tasks and business processes are too subtle and complex for their actors to describe easily.
- User stories. This technique is commonly used in adaptive methods (see Agile Methods in the Software Engineering Models and Methods KA) and refers to short, high-level descriptions of required functionality expressed in customer terms. A typical user story has the form: *"As a <role>, I want <goal/desire> so that <benefit>."* A user story is intended to contain just enough information so that the developers can produce a reasonable estimate of the effort to implement it. The aim is to avoid some of the waste that often happens in projects where detailed requirements are gathered early but become invalid before the work begins. Before a user story is implemented, an appropriate acceptance procedure must be written by the customer to determine whether the goals of the user story have been fulfilled.
- Other techniques. A range of other techniques for supporting the elicitation of requirements information exist and range from analyzing competitors' products to applying data mining techniques to using sources of domain knowledge or customer request databases.

4. Requirements Analysis

[1*, c4s1, c4s5, c10s4, c12s5]
[2*, c7, c11, c12, c17]

This topic is concerned with the process of analyzing requirements to

- detect and resolve conflicts between requirements;
- discover the bounds of the software and how it must interact with its organizational and operational environment;
- elaborate system requirements to derive software requirements.

The traditional view of requirements analysis has been that it be reduced to conceptual modeling using one of a number of analysis methods, such as the structured analysis method. While conceptual modeling is important, we include the classification of requirements to help inform tradeoffs between requirements (requirements classification) and the process of establishing these tradeoffs (requirements negotiation).

Care must be taken to describe requirements precisely enough to enable the requirements to be validated, their implementation to be verified, and their costs to be estimated.

4.1. Requirements Classification

Requirements can be classified on a number of dimensions. Examples include the following:

- Whether the requirement is functional or nonfunctional (see section 1.3, Functional and Nonfunctional Requirements).
- Whether the requirement is derived from one or more high-level requirements or an emergent property (see section 1.4, Emergent Properties), or is being imposed directly on the software by a stakeholder or some other source.
- Whether the requirement is on the product or the process (see section 1.2, Product and Process Requirements). Requirements on the process can constrain the choice of contractor, the software engineering process to be adopted, or the standards to be adhered to.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

- The requirement priority. The higher the priority, the more essential the requirement is for meeting the overall goals of the software. Often classified on a fixed-point scale such as mandatory, highly desirable, desirable, or optional, the priority often has to be balanced against the cost of development and implementation.
- The scope of the requirement. Scope refers to the extent to which a requirement affects the software and software components. Some requirements, particularly certain nonfunctional ones, have a global scope in that their satisfaction cannot be allocated to a discrete component. Hence, a requirement with global scope may strongly affect the software architecture and the design of many components, whereas one with a narrow scope may offer a number of design choices and have little impact on the satisfaction of other requirements.
- Volatility/stability. Some requirements will change during the life cycle of the software—and even during the development process itself. It is useful if some estimate of the likelihood that a requirement will change can be made. For example, in a banking application, requirements for functions to calculate and credit interest to customers' accounts are likely to be more stable than a requirement to support a particular kind of tax-free account. The former reflects a fundamental feature of the banking domain (that accounts can earn interest), while the latter may be rendered obsolete by a change to government legislation. Flagging potentially volatile requirements can help the software engineer establish a design that is more tolerant of change.

Other classifications may be appropriate, depending upon the organization's normal practice and the application itself.

There is a strong overlap between requirements classification and requirements attributes (see section 7.3, Requirements Attributes).

4.2. Conceptual Modeling

The development of models of a real-world problem is key to software requirements analysis. Their purpose is to aid in understanding the situation in which the problem occurs, as well as depicting a solution. Hence, conceptual models comprise models of entities from the problem domain, configured to reflect their real-world relationships and dependencies. This topic is closely related to the Software Engineering Models and Methods KA.

Several kinds of models can be developed. These include use case diagrams, data flow models, state models, goal-based models, user interactions, object models, data models, and many others. Many of these modeling notations are part of *the Unified Modeling Language (UML)*. Use case diagrams, for example, are routinely used to depict scenarios where the boundary separates the actors (users or systems in the external environment) from the internal behavior where each use case depicts a functionality of the system.

The factors that influence the choice of modeling notation include these:

- The nature of the problem. Some types of software demand that certain aspects be analyzed particularly rigorously. For example, state and parametric models, which are part of SysML [4], are likely to be more important for real-time software than for information systems, while it would usually be the opposite for object and activity models.
- The expertise of the software engineer. It is often more productive to adopt a modeling notation or method with which the software engineer has experience.
- The process requirements of the customer (see section 1.2, Product and Process Requirements). Customers may impose their favored notation or method or prohibit any with which they are unfamiliar. This factor can conflict with the previous factor.

Note that, in almost all cases, it is useful to start by building a model of the software context. The software context provides a connection between the intended software and its external environment.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

This is crucial to understanding the software's context in its operational environment and to identifying its interfaces with the environment.

This subtopic does not seek to "teach" a particular modeling style or notation but rather provides guidance on the purpose and intent of modeling.

4.3. Architectural Design and Requirements Allocation

At some point, the solution architecture must be derived. Architectural design is the point at which the requirements process overlaps with software or systems design and illustrates how impossible it is to cleanly decouple the two tasks. This topic is closely related to Software Structure and Architecture in the Software Design KA. In many cases, the software engineer acts as software architect because the process of analyzing and elaborating the requirements demands that the architecture/design components that will be responsible for satisfying the requirements be identified. This is requirements allocation–the assignment to architecture components responsible for satisfying the requirements.

Allocation is important to permit detailed analysis of requirements. Hence, for example, once a set of requirements has been allocated to a component, the individual requirements can be further analyzed to discover further requirements on how the component needs to interact with other components in order to satisfy the allocated requirements. In large projects, allocation stimulates a new round of analysis for each subsystem. As an example, requirements for a particular braking performance for a car (braking distance, safety in poor driving conditions, smoothness of application, pedal pressure required, and so on) may be allocated to the braking hardware (mechanical and hydraulic assemblies) and an antilock braking system (ABS). Only when a requirement for an antilock braking system has been identified, and the requirements allocated to it, can the capabilities of the ABS, the braking hardware, and emergent properties (such as car weight) be used to identify the detailed ABS software requirements.

Architectural design is closely identified with conceptual modeling (see section 4.2, Conceptual Modeling).

4.4. Requirements Negotiation

Another term commonly used for this subtopic is "conflict resolution." This concerns resolving problems with requirements where conflicts occur between two stakeholders requiring mutually incompatible features, between requirements and resources, or between functional and nonfunctional requirements, for example. In most cases, it is unwise for the software engineer to make a unilateral decision, so it becomes necessary to consult with the stakeholder(s) to reach a consensus on an appropriate tradeoff. It is often important, for contractual reasons, that such decisions be traceable back to the customer. We have classified this as a software requirements analysis topic because problems emerge as the result of analysis. However, a strong case can also be made for considering it a requirements validation topic (see topic 6, Requirements Validation).

Requirements prioritization is necessary, not only as a means to filter important requirements, but also in order to resolve conflicts and plan for staged deliveries, which means making complex decisions that require detailed domain knowledge and good estimation skills. However, it is often difficult to get real information that can act as a basis for such decisions. In addition, requirements often depend on each other, and priorities are relative. In practice, software engineers perform requirements prioritization frequently without knowing about all the requirements. Requirements prioritization may follow a cost-value approach that involves an analysis from the stakeholders defining in a scale the benefits or the aggregated value that the implementation of the requirement brings them, versus the penalties of not having implemented a particular requirement. It also involves an analysis from the software engineers estimating in a scale the cost of implementing each requirement, relative to other requirements. Another requirements prioritization approach called the analytic hierarchy process involves comparing all unique pairs of requirements to determine which of the two is of higher priority, and to what extent.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

4.5. Formal Analysis

Formal analysis concerns not only topic 4, but also sections 5.3 and 6.3. This topic is also related to Formal Methods in the Software Engineering Models and Methods Knowledge Area.

Formal analysis has made an impact on some application domains, particularly those of high-integrity systems. The formal expression of requirements requires a language with formally defined semantics. The use of a formal analysis for requirements expression has two benefits. First, it enables requirements expressed in the language to be specified precisely and unambiguously, thus (in principle) avoiding the potential for misinterpretation. Secondly, requirements can be reasoned over, permitting desired properties of the specified software to be proven. Formal reasoning requires tool support to be practicable for anything other than trivial systems, and tools generally fall into two types: theorem provers or model checkers. In neither case can proof be fully automated, and the level of competence in formal reasoning needed in order to use the tools restricts the wider application of formal analysis.

Most formal analysis is focused on relatively late stages of requirements analysis. It is generally counterproductive to apply formalization until the business goals and user requirements have come into sharp focus through means such as those described elsewhere in section 4. However, once the requirements have stabilized and have been elaborated to specify concrete properties of the software, it may be beneficial to formalize at least the critical requirements. This permits static validation that the software specified by the requirements does indeed have the properties (for example, absence of deadlock) that the customer, users, and software engineer expect it to have.

5. Requirements Specification

[1*, c4s2, c4s3, c12s2–5] [2*, c10]

For most engineering professions, the term "specification" refers to the assignment of numerical values or limits to a product's design goals. In software engineering, "software requirements specification" typically refers to the production of a document that can be systematically reviewed, evaluated, and approved. For complex systems, particularly those involving substantial nonsoftware components, as many as three different types of documents are produced: system definition, system requirements, and software requirements. For simple software products, only the third of these is required. All three documents are described here, with the understanding that they may be combined as appropriate. A description of systems engineering can be found in the Related Disciplines of Software Engineering chapter of this *Guide*.

5.1. System Definition Document

This document (sometimes known as the user requirements document or concept of operations document) records the system requirements. It defines the high-level system requirements from the domain perspective. Its readership includes representatives of the system users/customers (marketing may play these roles for market-driven software), so its content must be couched in terms of the domain. The document lists the system requirements along with background information about the overall objectives for the system, its target environment, and a statement of the constraints, assumptions, and nonfunctional requirements. It may include conceptual models designed to illustrate the system context, usage scenarios, and the principal domain entities, as well as workflows.

5.2. System Requirements Specification

Developers of systems with substantial software and nonsoftware components—a modern airliner, for example—often separate the description of system requirements from the description of software requirements. In this view, system requirements are specified, the software requirements are derived from the system requirements, and then the requirements for the software components are specified. Strictly speaking, system requirements specification is a systems engineering activity and falls outside the scope of this *Guide*.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015

5.3. Software Requirements Specification

Software requirements specification establishes the basis for agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do.

Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.

Organizations can also use a software requirements specification document as the basis for developing effective verification and validation plans.

Software requirements specification provides an informed basis for transferring a software product to new users or software platforms. Finally, it can provide a basis for software enhancement.

Software requirements are often written in natural language, but, in software requirements specification, this may be supplemented by formal or semiformal descriptions. Selection of appropriate notations permits particular requirements and aspects of the software architecture to be described more precisely and concisely than natural language. The general rule is that notations should be used that allow the requirements to be described as precisely as possible. This is particularly crucial for safety-critical, regulatory, and certain other types of dependable software. However, the choice of notation is often constrained by the training, skills, and preferences of the document's authors and readers.

A number of quality indicators have been developed that can be used to relate the quality of software requirements specification to other project variables such as cost, acceptance, performance, schedule, and reproducibility. Quality indicators for individual software requirements specification statements include imperatives, directives, weak phrases, options, and continuances. Indicators for the entire software requirements specification document include size, readability, specification, depth, and text structure.

6. Requirements Validation

[1*, c4s6] [2*, c13, c15]

The requirements documents may be subject to validation and verification procedures. The requirements may be validated to ensure that the software engineer has understood the requirements; it is also important to verify that a requirements document conforms to company standards and that it is understandable, consistent, and complete. In cases where documented company standards or terminology are inconsistent with widely accepted standards, a mapping between the two should be agreed on and appended to the document.

Formal notations offer the important advantage of permitting the last two properties to be proven (in a restricted sense, at least). Different stakeholders, including representatives of the customer and developer, should review the document(s). Requirements documents are subject to the same configuration management practices as the other deliverables of the software life cycle processes. When practical, the individual requirements are also subject to configuration management, generally using a requirements management tool (see topic 8, Software Requirements Tools).

It is normal to explicitly schedule one or more points in the requirements process where the requirements are validated. The aim is to pick up any problems before resources are committed to addressing the requirements. Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right software (that is, the software that the users expect).

6.1. Requirements Reviews

Perhaps the most common means of validation is by inspection or reviews of the requirements document(s). A group of reviewers is assigned a brief to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice. The composition of the group that conducts the review is important (at least one representative of the customer should be included for a customer-driven project, for example), and it may help to provide guidance on what to look for in the form of checklists.

© ISO/IEC 2016 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO_IEC TR 19759_2015