SEBoK GUIDE TO THE SYSTEMS ENGINEERING BODY OF KNOWLEDGE

Guide to the Systems Engineering Body of Knowledge (SEBoK), version 2.2

Part 6

Please note that this is a PDF extraction of the content from www.sebokwiki.org

Released 15 May 2020

Guide to the Systems Engineering Body of Knowledge, Part 6

version 2.2

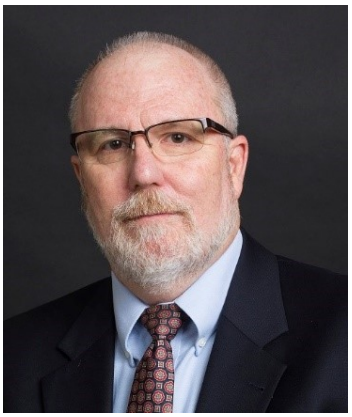
Contents

Articles

References

Front Matter

Letter from the Editor



Hi there. Welcome to the May 2020 instantiation of the SEBoK. We are now at version 2.2. If you remember, we celebrated our 7th anniversary last update. Well, this update we are celebrating too. In the month of April 2020, we had our 2 millionth visit since we started. And, we have over 4 million page views since we first rolled out the SEBoK! Month over month usage of the SEBoK continues to grow. That could mean that the editorial staff and authors continue to add value to you our stakeholders and customers or it could mean that Systems Engineering is growing around the world, and we are the "go to" location for that information. I choose to believe it is a bit of both. Thank you for continuing to visit the SEBoK, contribute to its content, and to tell others about this resource.

In case you are wondering, here are the top 10 pages visited in April 2020, in order:

1. Stakeholder Needs and Requirements
2. System Requirements
3. Reliability, Availability, Maintainability
4. Types of Systems
5. Types of Models
6. System Life Cycle Process Models: Vee
7. Systems Architecture
8. Systems Engineering Overview
9. Life Cycle Models
10. Logical Architecture Model Development

So, what is new for Version 2.2?

First update, and this is big - notice the IEEE logo on the top of the page has changed from the IEEE Computer Society to the **IEEE Systems Council**! We are excited to have them onboard and are already coordinating new contributions and participation of IEEE members. Welcome! I'd also like to thank the IEEE Computer Society for all of their guidance and support of the SEBoK since 2013.

Second update – notice that we have updated the organization of Part 7: Implementation Examples. Examples are now aligned with engineering domains. We hope this makes it easier for you to find relevant examples of Systems Engineering in the real world.

Third update – in addition to reorganizing Part 7, we have added an entirely new Part to the SEBoK: Part 8, Emerging Knowledge. Systems Engineering is evolving faster and faster as the world is changing. In Part 8, the SEBoK will endeavor to inform you of trends that are taking root in some of our systems engineering communities. We moved the SE Transformation items from Part 1 to this new part. Additionally, we have added a section for Emerging Research. This is a place to provide pointers to doctoral level systems engineering that has been defended in the recent past.

New articles to check out:

- Systems Engineering Principles
- Apollo 1 Disaster

I would like to point out some changes in the **editorial organization** of the SEBoK. Tom McDermott has agreed to be the Lead Editor for Part 4: Applications of Systems Engineering. Nicole Hutchison, our Managing Editor, will become the Lead Editor for Part 5: Enabling Systems Engineering. Art Pyster is now the Lead Editor for Part 6: Related Disciplines. And finally, Dan DeLaurentis will become the Lead Editor for the new Part 8: Emerging Knowledge. Thank you all for your ongoing commitment to the SEBoK.

OPPORTUNITY: Finally, we continue to look for ways to add some multimedia to the SEBoK. In this update, we have identified some links to relevant YouTube talks that we believe might be of value to you. However, most of that material was intended for something else. I am looking for one or more amateur videographers and hobbyists to produce a number of 3-5 minute videos on systems engineering specifically for the SEBoK. NO AGENDAS. NO PROMOTIONS. NO ADVERTISEMENTS. Just straight talk on a specific topic of systems engineering. Ideally, these will have good quality, good volume, and great content. I am hoping they do not look like they were shot at a conference or in a classroom. If you are up to this challenge, please contact me at: rob@calimar.com [1]. I look forward to your ideas.

THANK YOU for reading this rather lengthy missive. If you would like to contribute an article to the SEBoK, or have an idea for one, please reach out to me – we always need new articles, video, etc. And, I am still in search of a Lead Editor for Part 3: Systems Engineering and Management. Thanks to all for your ongoing support and readership.

References

[1] mailto:rob@calimar.com

BKCASE Governance and Editorial Board

BKCASE Governing Board

The three SEBoK steward organizations – the International Council on Systems Engineering (INCOSE), the Institute of Electrical and Electronics Engineers Systems Council (IEEE-SYSC), and the Systems Engineering Research Center (SERC) provide the funding and resources needed to sustain and evolve the SEBoK and make it available as a free and open resource to all. The stewards appoint the BKCASE Governing Board to be their primary agents to oversee and guide the SEBoK and its companion BKCASE product, GRCSE.

The BKCASE Governing Board includes:

- **The International Council on Systems Engineering (INCOSE)**
  - Art Pyster (Governing Board Chair), Paul Frenz
- **Systems Engineering Research Center (SERC)**
  - Jon Wade, Cihan Dagli
- **IEEE Systems Council (IEEE-SYSC)**
  - Stephanie White, Bob Rassa

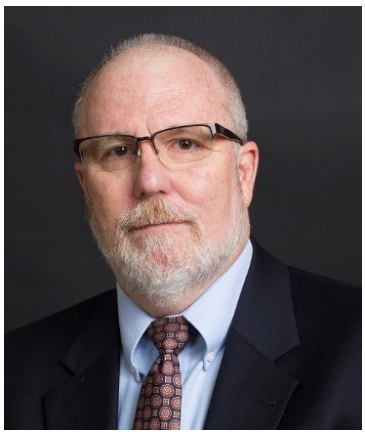
Past INCOSE governors Bill Miller, Kevin Forsberg, David Newbern, David Walden, Courtney Wright, Dave Olwell, Ken Nidiffer, Richard Fairley, Massood Towhidnejad, John Keppler. The governors would also like to acknowledge Andy Chen and Rich Hilliard, IEEE Computer Society, who were instrumental in helping the Governors to work within the IEEE CS structure and who supported the SEBoK transition to the IEEE Systems Council.

The stewards appoint the SEBoK Editor in Chief to manage the SEBoK and oversee the Editorial Board.

SEBoK Editorial Board

The SEBoK Editorial Board is chaired by the Editor in Chief, who provide the strategic vision for the SEBoK. The EIC is supported by a group of Editors, each of whom are responsible for a specific aspect of the SEBoK. The Editorial Board is supported by the Managing Editor, who handles all day-to-day operations. The EIC, Managing Editor, and Editorial Board are supported by a student, Madeline Haas, whose hard work and dedication are greatly appreciated.

SEBoK Editor in Chief



Robert J. Cloutier

University of South Alabama

rcloutier@southalabama.edu [1]

Responsible for the appointment of SEBoK Editors and for the strategic direction and overall quality and coherence of the SEBoK.

SEBoK Managing Editor

Nicole Hutchison

Systems Engineering Research Center

nicole.hutchison@stevens.edu [2] or emtnicole@gmail.com [3]

Responsible for the the day-to-day operations of the SEBoK and supports the Editor in Chief.

Each Editor has his/her area(s) of responsibility, or shared responsibility, highlighted in the table below.

SEBoK Part 1: SEBoK Introduction

Lead Editor: Robert J. Cloutier

University of South Alabama

rcloutier@southalabama.edu [1]

SEBoK Part 2: Foundations of Systems Engineering

Lead Editor: Gary Smith

Airbus

gary.r.smith@airbus.com [4]

Assistant Editor: Dov Dori

Massachusetts Institute of Technology (USA) and Technion Israel Institute of Technology (Israel)

dori@mit.edu [5]

Responsible for the Representing Systems with Models knowledge area

Assistant Editor: Duane Hybertson

MITRE (USA)

dhyberts@mitre.org [6]

Jointly responsible for the Systems Fundamentals, Systems Science and Systems Thinking knowledge areas.

Assistant Editor: Peter Tuddenham

College of Exploration (USA)

Peter@coexploration.net [7]

Assistant Editor: Cihan Dagli

Missouri University of Science & Technology (USA)

dagli@mst.edu [8]

Responsible for the Systems Approach Applied to Engineered Systems knowledge areas.

SEBoK Part 3: Systems Engineering and Management

Assistant Editor: Barry Boehm

University of Southern California (USA)

boehm@usc.edu [9]

Jointly responsible for the Systems Engineering Management and Life Cycle Models knowledge areas

Assistant Editor: Kevin Forsberg

OGR Systems

kforsberg@ogrsystems.com [10]

Jointly responsible for the Systems Engineering Management and Life Cycle Models knowledge areas

Assistant Editor: Gregory Parnell

University of Arkansas (USA)

gparnell@uark.edu [11]

Responsible for Systems Engineering Management knowledge area.

Assistant Editor: Garry Roedler

Lockheed Martin (USA)

garry.j.roedler@lmco.com [12]

Responsible for the Concept Definition and System Definition knowledge areas.

Assistant Editor: Phyllis Marbach

Incose LA (USA)

prmarbach@gmail.com [13]

Assistant Editor: Ken Zemrowski

ENGILITY

kenneth.zemrowski@incose.org [14]

Responsible for the Systems Engineering Standards knowledge area.

SEBoK Part 4: Applications of Systems Engineering

Lead Editor: Tom McDermott

Systems Engineering Research Center (SERC)

tmcdermo@stevens.edu [15]

Assistant Editor: Judith Dahmann

MITRE Corporation (USA)

jdahmann@mitre.org [16]

Jointly responsible for Product Systems Engineering and Systems of Systems (SoS) knowledge areas.

Assistant Editor: Michael Henshaw

Loughborough University (UK)

M.J.d.Henshaw@lboro.ac.uk [17]

Jointly responsible for Product Systems Engineering and Systems of Systems (SoS) knowledge areas

Assistant Editor: James Martin

The Aerospace Corporation

james.martin@incose.org [18]

Responsible for the Enterprise Systems Engineering knowledge area.

SEBoK Part 5: Enabling Systems Engineering

Lead Editor: Nicole Hutchison

Systems Engineering Research Center

[Mailto:nicole.hutchison@stevens.edu nicole.hutchison@stevens.edu]

Assistant Editor: Emma Sparks

Cranfield University

Jointly responsible for the Enabling Individuals and Enabling Teams knowledge areas.

Assistant Editor: Rick Hefner

California Institute of Technology

Rick.Hefner@ngc.com [19]

Assistant Editor: Bernardo Delicado

MBDA / INCOSE

bernardo.delicado@mbda-systems.com [20]

SEBoK Part 6: Related Disciplines

Lead Editor: Art Pyster

George Mason University (USA)

apyster@gmu.edu [21]

SEBoK Part 7: Systems Engineering Implementation Examples

Lead Editor: Clif Baldwin

FAA Technical Center

cliftonbaldwin@gmail.com [22]

SEBoK Part 8: Emerging Knowledge

Lead Editor: Daniel DeLaurentis

Purdue University

ddelaure@purdue.edu [23]

Student Support

Madeline Haas, a student at George Mason University, is currently supporting the SEBoK and we gratefully acknowledge her exemplary efforts. Ms. Haas has also taken responsibility for managing the Emerging Research knowledge area of the SEBoK. The EIC and Managing Editor are very proud of the work Madeline has done and look forward to continuing to work with her.

Interested in Editing?

The Editor in Chief is looking for additional editors to support the evolution of the SEBoK. Editors are responsible for maintaining and updating one to two knowledge areas, including recruiting and working with authors, ensuring the incorporation of community feedback, and maintaining the quality of SEBoK content. We are specifically interested in support for the following knowledge areas:

- System Deployment and Use
- Product and Service Life Management
- Enabling Businesses and Enterprises
- Systems Engineering and Software Engineering
- Procurement and Acquisition
- Systems Engineering and Specialty Engineering

In addition, the Editor in Chief is looking for a new Lead Editor for Part 3: Systems Engineering and Management.

If you are interested in being considered for participation on the Editorial Board, please contact the SEBoK Staff directly at sebok@incose.org [24].

SEBoK v. 2.2, released 15 May 2020

References

[1] mailto:rcloutier@southalabama.edu
[2] mailto:nicole.hutchison@stevens.edu
[3] mailto:emtnicole@gmail.com
[4] mailto:gary.r.smith@airbus.com
[5] mailto:dori@mit.edu
[6] mailto:dhyberts@mitre.org
[7] mailto:Peter@coexploration.net
[8] mailto:dagli@mst.edu
[9] mailto:boehm@usc.edu
[10] mailto:kforsberg@ogrsystems.com
[11] mailto:gparnell@uark.edu
[12] mailto:garry.j.roedler@lmco.com
[13] mailto:prmarbach@gmail.com
[14] mailto:kenneth.zemrowski@incose.org
[15] mailto:tmcdermo@stevens.edu
[16] mailto:jdahmann@mitre.org
[17] mailto:M.J.d.Henshaw@lboro.ac.uk
[18] mailto:james.martin@incose.org
[19] mailto:Rick.Hefner@ngc.com
[20] mailto:bernardo.delicado@mbda-systems.com
[21] mailto:apyster@gmu.edu
[22] mailto:cliftonbaldwin@gmail.com
[23] mailto:ddelaure@purdue.edu
[24] mailto:sebok@incose.org

Acknowledgements and Release History

This article describes the contributors to the current version of the SEBoK. For information on contributors to past versions of the SEBoK, please follow the links under "SEBoK Release History" below. To learn more about the updates to the SEBoK for v. 2.2, please see the Letter from the Editor.

The BKCASE Project began in the fall of 2009. Its aim was to add to the professional practice of systems engineering by creating two closely related products:

- *Guide to the Systems Engineering Body of Knowledge (SEBoK)*
- *Graduate Reference Curriculum for Systems Engineering (GRCSE)*

BKCASE History, Motivation, and Value

The **Guide to the Systems Engineering Body of Knowledge (SEBoK)** is a living authoritative guide that discusses knowledge relevant to Systems Engineering. It defines how that knowledge should be structured to facilitate understanding, and what reference sources are the most important to the discipline. The curriculum guidance in the **Graduate Reference Curriculum for Systems Engineering (GRCSE)** (Pyster and Olwell et al. 2015) makes reference to sections of the SEBoK to define its core knowledge; it also suggests broader program outcomes and objectives which reflect aspects of the professional practice of systems engineering as discussed across the SEBoK.

Between 2009 and 2012, BKCASE was led by Stevens Institute of Technology and the Naval Postgraduate School in coordination with several professional societies and sponsored by the U.S. Department of Defense (DoD), which provided generous funding. More than 75 authors and many other reviewers and supporters from dozens of companies, universities, and professional societies across 10 countries contributed many thousands of hours writing the SEBoK articles; their organizations provided significant other contributions in-kind.

The SEBoK came into being through recognition that the systems engineering discipline could benefit greatly by having a living authoritative guide closely related to those groups developing guidance on advancing the practice, education, research, work force development, professional certification, standards, etc.

At the beginning of 2013, BKCASE transitioned to a new governance model with shared stewardship between the Systems Engineering Research Center (SERC) [1], the International Council on Systems Engineering (INCOSE) [2], and the Institute of Electrical and Electronics Engineers Computer Society (IEEE-CS) [3]. This governance structure was formalized in a memorandum of understanding between the three stewards that was finalized in spring of 2013 and subsequently updated. In January 2020, the IEEE Systems Council [4] replaced the IEEE-CS in representing IEEE as a steward. The stewards have reconfirmed their commitment to making the SEBoK available at no cost to all users, a key principle of BKCASE.

As of April 2020, SEBoK articles have had over 4.2M pageviews from 1.7M unique visitors. We hope the SEBoK will regularly be used by thousands of systems engineers and others around the world as they undertake technical activities such as eliciting requirements, creating systems architectures, or analyzing system test results; and professional development activities such as developing career paths for systems engineers, deciding new curricula for systems engineering university programs, etc.

Governance

The SEBoK is shaped by the SEBoK Editorial Board and is overseen by the BKCASE Governing Board. A complete list of members for each of these bodies can be found on the BKCASE Governance and Editorial Board page.

Content and Feature Updates for 2.2

This version of the SEBoK was released 15 May 2020. This is a significant release of the SEBoK which includes new articles, new functionality and minor updates throughout. The SEBoK PDF was also updated (see Download SEBoK PDF).

For more information about this release please refer to Development of SEBoK v. 2.2.

SEBoK Release History

There have been 22 releases of the SEBoK to date, collected into 14 main releases.

Main Releases

- Version 2.2 - Current version. This is a significant release, including the first new Part to be added since v. 1.0 - Emerging Knowledge - which is a place to highlight new topics in systems engineering that are important but may not yet have a large body of literature. Recent dissertations around emerging topics are also included. A new case study on Apollo 1 was added to Part 7, which has also been reorganized around topics. Additional minor updates have occurred throughout.
- Version 2.1 - This was a significant release with new articles, new functionality, and minor updates throughout.
- Version 2.0 - This was a major release of the SEBoK which included incorporation of multi-media and a number of changes to the functions of the SEBoK.
- Version 1.9.1 - This was a micro release of the SEBoK which included updates to the editorial board, and a number of updates to the wiki software.
- Version 1.9 - A minor update which included updates to the System Resilience article in Part 6: Related Disciplines, as well as a major restructuring of Part 7: Systems Engineering Implementation Examples. A new example has been added around the use of model based systems engineering for the thirty-meter telescope.
- Version 1.8 - A minor update, including an update of the Systems of Systems (SoS) knowledge area in Part 4: Applications of Systems Engineering where a number of articles were updated on the basis of developments in

the area as well as on comments from the SoS and SE community. Part 6: Related Disciplines included updates to the Manufacturability and Producibility and Reliability, Availability, and Maintainability articles.

- Version 1.7 - A minor update, including a new Healthcare SE Knowledge Area (KA), expansion of the MBSE area with two new articles, Technical Leadership and Reliability, Availability, and Maintainability and a new case study on the Northwest Hydro System.
- Version 1.6 - A minor update, including a reorganization of Part 1 SEBoK Introduction, a new article on the Transition towards Model Based Systems Engineering and a new article giving an overview of Healthcare Systems Engineering, a restructure of the Systems Engineering and Specialty Engineering KA.
- Version 1.5 - A minor update, including a restructure and extension of the Software Engineering Knowledge Area, two new case studies, and a number of corrections of typographical errors and updates of outdated references throughout the SEBoK.
- Version 1.4 - A minor update, including changes related to ISO/IEC/IEEE 15288:2015 standard, three new case studies and updates to a number of articles.
- Version 1.3 - A minor update, including three new case studies, a new use case, updates to several existing articles, and updates to references.
- Version 1.2 - A minor update, including two new articles and revision of several existing articles.
- Version 1.1 - A minor update that made modest content improvements.
- Version 1.0 - The first version intended for broad use.

Click on the links above to read more information about each release.

Wiki Team

In January 2011, the authors agreed to move from a document-based SEBoK to a wiki-based SEBoK, and beginning with v. 0.5, the SEBoK has been available at www.sebokwiki.org [5] Making the transition to a wiki provided three benefits:

1. easy worldwide access to the SEBoK;
2. more methods for search and navigation; and
3. a forum for community feedback alongside content that remains stable between versions.

The Managing Editor is responsible for maintenance of the wiki infrastructure as well as technical review of all materials prior to publication. Contact the managing editor at emtnicole@gmail.com [3]

The wiki is currently supported by Ike Hecht from WikiWorks.

SEBoK v. 2.2, released 15 May 2020

References

[1] http://www.sercuarc.org
[2] http://www.incose.org
[3] http://www.computer.org
[4] https://ieeesystemscouncil.org/
[5] http://www.sebokwiki.org

Cite the SEBoK

When **citing the SEBoK in general**, users must cite in the following manner:

> SEBoK Editorial Board. 2020. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 2.2, R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed [DATE]. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society.

To **cite a specific article** within the SEBoK, please use:

> Author name(s). "Article Title." in SEBoK Editorial Board. 2020. *The Guide to the Systems Engineering Body of Knowledge (SEBoK)*, v. 2.2 R.J. Cloutier (Editor in Chief). Hoboken, NJ: The Trustees of the Stevens Institute of Technology. Accessed [DATE]. www.sebokwiki.org. BKCASE is managed and maintained by the Stevens Institute of Technology Systems Engineering Research Center, the International Council on Systems Engineering, and the Institute of Electrical and Electronics Engineers Computer Society.

Note that each many pages include the by line (author names) for the article. If no byline is listed, please use "SEBoK Authors".

When **using material** from the SEBoK, attribute the work as follows:

> This material is used under a Creative Commons Attribution-NonCommercial ShareAlike 3.0 Unported License from The Trustees of the Stevens Institute of Technology. See Stevens Terms for Publication located in Copyright Information.

Cite this Page

This feature is located under "Tools" on the left menu. It provides full information to cite the specific article that you are currently viewing; this information is provided in various common citation styles including APA, MLA, and Chicago.

Bkcase Wiki:Copyright

Please read this page which contains information about how and on what terms you may use, copy, share, quote or cite the Systems Engineering Body of Knowledge (SEBoK):

Copyright and Licensing

Attribution

When **using text material from the SEBoK**, users who have accepted one of the Creative Commons Licenses described above terms noted below must attribute the work as follows:

This material is used under a Creative Commons Attribution-NonCommercial ShareAlike 3.0 Unported License from The Trustees of the Stevens Institute of Technology.

When **citing the SEBoK in general**, please refer to the format described on the Cite the SEBoK page.

When **using images, figures, or tables from the SEBoK**, please note the following intellectual property (IP) classifications:

- Materials listed as "SEBoK Original" may be used in accordance with the Creative Commons attribution (above).
- Materials listed as "Public Domain" may be used in accordance with information in the public domain.
- Materials listed as "Used with Permission" are copyrighted and *permission must be sought from the copyright owner* to reuse them.

Part 6: Related Disciplines

Related Disciplines

Lead Author: *Art Pyster*, ***Contributing Authors:*** *Dick Fairley, Tom Hilburn, Alice Squires*

Systems engineers routinely work within broad multidisciplinary teams (Pyster, et al, 2018). Part 6 of the Guide to the SE Body of Knowledge (SEBoK) presents knowledge that should be useful to systems engineers as they interact with these other fields and professionals in those fields.

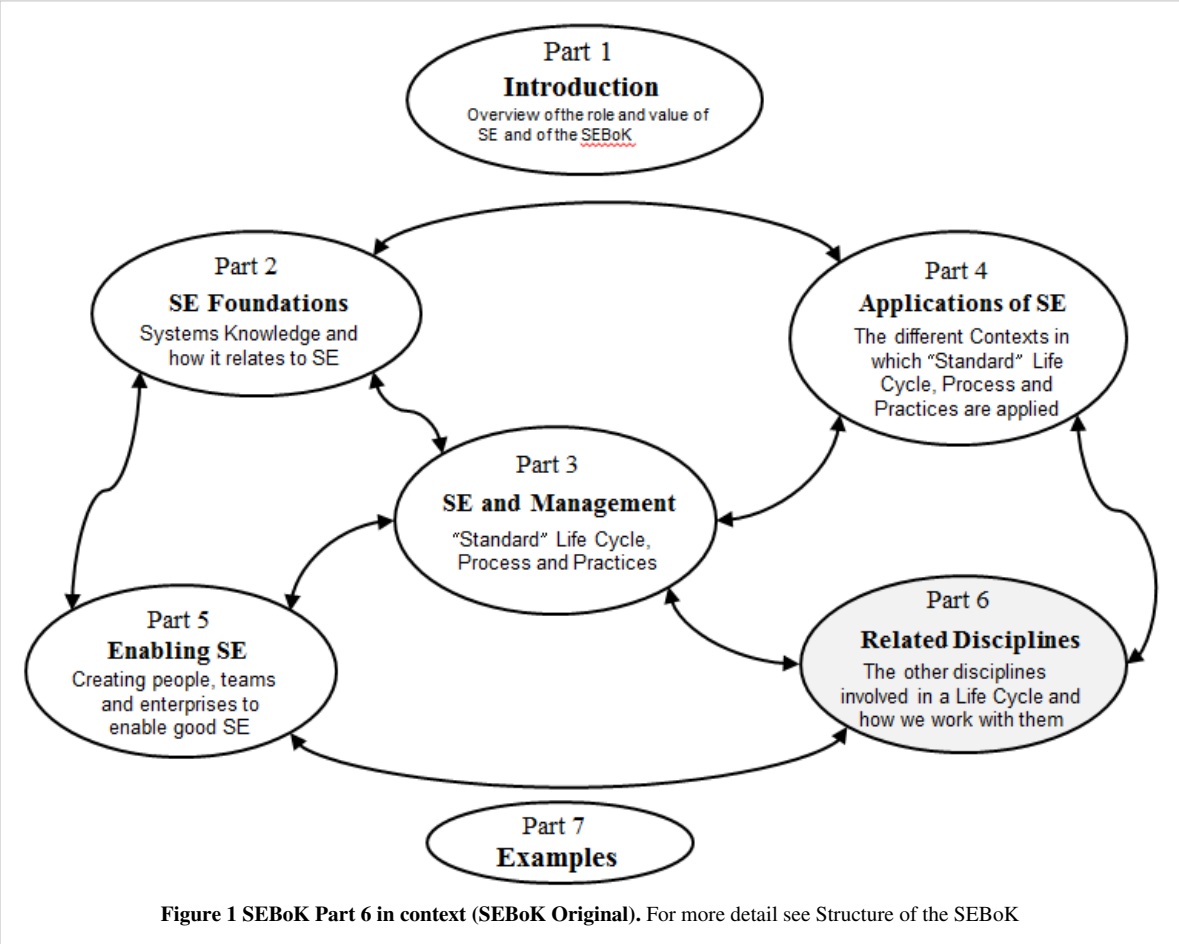


Figure 1 SEBoK Part 6 in context (SEBoK Original). For more detail see Structure of the SEBoK

SE intersects with virtually every other recognized discipline. Besides the other engineering disciplines such as electrical and mechanical engineering, SE intersects with the physical sciences, social sciences, project management, philosophy, etc. For example, a systems engineer leading the design of an autonomous car would work with electrical engineers, software engineers, project managers, mechanical engineers, computer scientists, radio engineers, data analysts, human factors specialists, cybersecurity engineers, economists, and professionals from many other disciplines. The knowledge areas (KAs) contained in Part 6 and the topics under them provide an overview of some of these disciplines with emphasis on what a systems engineer needs to know to be effective, accompanied by pointers to that knowledge. The KAs covered in Part 6 could run into the dozens, but only a handful are addressed in this version of the SEBoK. Subsequent SEBoK releases will expand the number of related disciplines and offer deeper insight into their relationship with SE.

Knowledge Areas in Part 6

Each part of the SEBoK is divided into knowledge areas (KAs), which are groupings of information with a related theme. Part 6 contains the following KAs:

- Systems Engineering and Software Engineering
- Systems Engineering and Project Management
- Systems Engineering and Industrial Engineering
- Systems Engineering and Specialty Engineering

Each KA above except the last is a major well-recognized stand-alone discipline. Each is widely taught in universities around the world, has professional societies devoted to it, standards that assist its practitioners, publications that describe its knowledge and practices, and a vibrant community of practitioners and researchers who often have one or more university degrees in the discipline. The last KA is different. It describes the disciplines associated with engineering system properties; e.g., security is a system property. Security engineering is the discipline through which system security is realized in a system. The security of a modern car is widely understood to be a function of many factors such as the strength of its physical exterior, its alarm system which may have extensive sensors and software, and its communications system which can wirelessly alert the owner or police if someone attempts to break into it. Similarly, the reliability of a car is a function of such factors as the reliability of its individual subsystems and components (mechanical, electronic, software, etc.) and how the car has been designed to compensate for a failed subsystem or component (e.g. if the electronic door lock fails, can the driver use a physical key to lock and unlock the car?). The topics included in this KA are among the most important ones a systems engineer would typically consider.

Collectively, these disciplines, which address the engineering of system properties, are often referred to as specialty engineering.

References

Works Cited

Pyster, A., N. Hutchison, D. Henry. 2018. The Paradoxical Mindset of Systems Engineers. Hoboken, NJ, USA: Wiley.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *SWEBOK: Guide to the Software Engineering Body of Knowledge*, version 3.0. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

DAU. 2010. *Defense Acquisition Guidebook (DAG)*. Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/US Department of Defense.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

None.

Knowledge Area: Systems Engineering and Software Engineering

Systems Engineering and Software Engineering

Lead Authors: *Dick Fairley, Tom Hilburn*, ***Contributing Authors:*** *Ray Madachy, Alice Squires*

Software is prominent in most modern systems architectures and is often the primary means for integrating complex system components. Software engineering and systems engineering are not merely related disciplines; they are intimately intertwined. (See Systems Engineering and Other Disciplines.) Good systems engineering is a key factor in enabling good software engineering.

The SEBoK explicitly recognizes and embraces the intertwining between systems engineering and software engineering, as well as defining the relationship between the SEBoK and the Guide to the Software Engineering Body of Knowledge (SWEBOK) (Bourque, and Fairley, 2014).

This knowledge area describes the nature of software, provides an overview of the SWEBOK, describes the concepts that are shared by systems engineers and software engineers, and indicates the similarities and difference in how software engineers and systems engineers apply these concepts and use common terminology. It also describes the nature of the relationships between software engineering and systems engineering and describes some of the methods, models and tools used by software engineers.

Topics

Each part of the SEBoK is divided into knowledge areas (KAs), which are groupings of information with a related theme. The Kas, in turn, are divided into topics. This KA contains the following topics:

- Software Engineering in the Systems Engineering Life Cycle
- The Nature of Software
- An Overview of the SWEBOK Guide
- Key Points a Systems Engineer Needs to Know about Software Engineering
- Software Engineering Features - Models, Methods, Tools, Standards, and Metrics

Discussion

Software engineers, like systems engineers,

- engage in analysis and design, allocation of requirements, oversight of component development, component integration, verification and validation, life cycle sustainment, and system retirement.
- work with or as a component specialist (for example, user interface, database, computation, and communication specialists) who construct or otherwise obtain the needed software components.
- adapt existing components and incorporate components supplied by customers and affiliated organizations.

These commonalities would make it appear that software engineering is merely an application of systems engineering, but this is only a superficial appearance. The differences between the two disciplines arise from two fundamental issues:

1. Differences in educational backgrounds (traditional engineering disciplines for SE and the computing disciplines for SWE) and work experiences that result in different approaches to problem solving, and

2. Different ways of applying shared concepts based on the contrasting natures of the software medium and the physical media of traditional engineering.

Table 1 itemizes some of the shared concepts that are applied in different ways by systems engineers and software engineers. Each discipline has made contributions to the other. Table 1 indicates the methods and techniques developed by systems engineers adapted for use by software engineers and, conversely, those that have been adapted for use by systems engineers.

Table 1. Adaptation of Methods Across SE and SWE (Fairley and Willshire 2011) Reprinted with permission of Dick Fairley and Mary Jane Willshire. All other rights are reserved by the copyright owner. *

Systems Engineering Methods Adapted to Software Engineering	Software Engineering Methods Adapted to Systems Engineering
• Stakeholder Analysis	• Model-Driven Development
• Requirements Engineering	• UML-SysML
• Functional Decomposition	• Use Cases
• Design Constraints	• Object-Oriented Design
• Architectural Design	• Iterative Development
• Design Criteria	• Agile Methods
• Design Tradeoffs	• Continuous Integration
• Interface Specification	• Process Modeling
• Traceability	• Process Improvement
• Configuration Management	• Incremental Verification and Validation
• Systematic Verification and Validation	

The articles in this knowledge area give an overview of software and software engineering aimed at systems engineers. It also provides more details on the relationship between systems and software life cycles and some of the detailed tools used by software engineers. As systems become more dependent on software as a primary means of delivering stakeholder value, the historical distinction between software and systems engineering may need to be challenged. This is a current area of joint discussion between the two communities which will affect the future knowledge in both SEBoK and SWEBoK.

References

Works Cited

Bourque, P. and Fairley, R.E. (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*). Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Fairley, R.E. and Willshire M.J., 2011. *Teaching systems engineering to software engineering students*, *CSEET* 2011, Software Engineering Education and Training, p: 219-226, ISBN: 978-1-4577-0349-2.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Brooks, F. 1995. *The Mythical Man-Month,* Anniversary Edition. Boston, MA, USA: Addison Wesley Longman Inc.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, NJ, USA: John Wiley and Sons.

Additional References

Pressman, R. 2009. *Software Engineering: A Practitioner's Approach.* 7th Ed. New York, NY, USA: McGraw Hill.

Schneidewind, N. 2009. *Systems and Software Engineering with Applications*. New York, NY, USA: Institute of Electrical and Electronics Engineers.

Sommerville, I. 2010. *Software Engineering.* 9th Ed. Boston, MA, USA: Addison Wesley.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Software Engineering in the Systems Engineering Life Cycle

Lead Authors: *Tom Hilburn, Dick Fairley*, ***Contributing Author:*** *Alice Squires*

This article describes how software engineering (SwE) life cycle processes integrate with the SE life cycle. A joint workshop organized by INCOSE, the Systems Engineering Research Center and the IEEE Computer Society was held to consider this relationship (Pyster et al, 2015). This workshop concluded that:

> *Software is fundamental to the performance, features, and value of most modern engineering systems. It is not merely part of the system, but often shapes the system architecture; drives much of its complexity and emergent behavior; strains its verification; and drives much of the cost and schedule of its development. Given how significant an impact software has on system development and given how complex modern systems are, one would expect the relationship between the disciplines of systems engineering (SE) and software engineering (SWE) to be well defined. However, the relationship is, in fact, not well understood or articulated.*

In this article we give some of the basic relationships between SwE and SE and discuss how these can be related to some of the SEBoK knowledge areas.

Systems Engineering and Software Engineering Life Cycles

The Guide to the Software Engineering Body of Knowledge (SWEBoK) (Bourque and Fairley 2014) describes the life cycle of a software product as:

- analysis and design,
- construction,
- testing,
- operation,
- maintenance, and eventually
- retirement or replacement.

This life cycle is common to most other mature engineering disciplines.

In Part 3 of the SEBoK, SE and Management, there is a discussion of SE life cycle models and life cycle processes. A Generic Life Cycle Model is described and reproduced in Fig. 1 below. This is used to describe necessary stages in the life cycle of a typical engineered system.

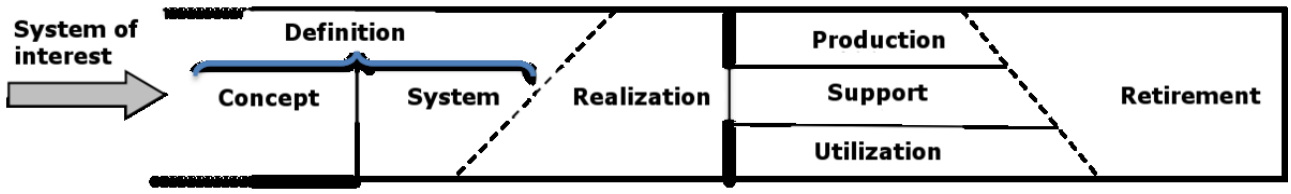


Figure 1. A Generic Life Cycle Model. (SEBoK Original)

Part 3 defines a collection of generic SE life cycle processes which define the activities and information needed across the SE life cycle. These processes include activities which contribute across the whole life cycle, with peaks of focused activity in certain stages (see Applying Life Cycle Processes for details).

The following sections provide a brief discussion of how SwE life cycle processes fit into SE life cycle process models. In practice, the details of this relationship are a key part of how a system life cycle is planned and delivered. The relationship will be shaped by the operating domain practice and solution type. Some examples of this are provided in the Implementation Examples.

Systems Engineering and Software Engineering Standards

The Systems Engineering life cycle processes described in Part 3, SE and Management, are largely based on those defined in the ISO/IEC/IEEE SE Life Cycle Processes 15288 Standard (2015).

The SWEBoK references the equivalent ISO/IECIEEE Software Engineering Life Cycle Processes 12207 Standard (2008), which defines a very similar set of processes for software systems. Figure 2 shows the relationship between the Enabling, Acquisition, Project and Technical Systems and Software processes in both 15288 and 12207 and the software specific processes of 12207. This alignment is from the last updates of both 12207 and 15288 in 2008. The SE processes have been further updated in 15288:2015, see Systems Engineering and Management for details. This change has not yet been applied to 12207. An update of 12207 is planned for 2016, in which the alignment to 15288 will be reviewed. See Alignment and Comparison of the Standards for more discussion of the relationships between the standards.

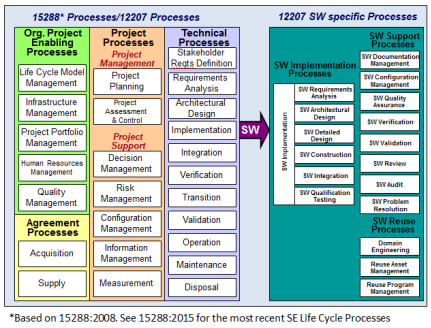


Figure 2. Aligned Process Models for ISO/IEC/IEEE 15288 & 12207: 2008 (Adapted from Roedler 2011). Reprinted with permission of Garry Roedler. All other rights are reserved by the copyright owner.

Systems Engineering and Software Engineering Life Cycle Relationships

Pyster et al (2015) define two technical dimensions of engineered systems and of the engineering disciplines associated with them. The vertical dimensions of a system are those that modularize around technically focused engineering concerns involving specific elements of the system; the horizontal dimensions of a system involve cross-cutting concerns at the systems level. Examples of vertical concerns include quality attributes and performance effectiveness; and cost, schedule and risk of physical, organizational or human system elements associated with a particular technology domain. Examples of horizontal concerns include addressing evolving customer preferences that drive systems-level quality attributes, trade-off and optimization; resolving system architecture, decomposition and integration issues; implementing system development processes; and balancing system economics, cost, risk and schedule.

In complex systems projects, SE has a horizontal role while traditional engineering disciplines such as electrical, mechanical, and chemical engineering have vertical roles. To the extent that it is responsible for all aspects of the successful delivery of software related elements, SwE can be considered as one of the vertical disciplines. All of these traditional vertical disciplines will have some input to the horizontal dimension. However, the nature of software and its role in many complex systems makes SwE a critical discipline for many horizontal concerns. This is discussed further below.

The ISO/IEC/IEEE 12207 software engineering standard (2008) considers two situations:

- The life cycle of software products, containing minimal physical hardware, should use software specific processes and a simple life cycle
- The life cycle of systems with a significant software content (sometimes called software intensive systems) should integrate the software processes into the SE life cycle

The second of these situations is the one relevant to the practice of SE and requires a significant horizontal contribution from SwE.

The relationship central to this is the way **SwE Implementation Processes** (see Fig 2) are used in the SE life cycle to support the implementation of software intensive system elements. This simple relationship must be seen in the context of the concurrency, iteration and recursion relationship between SE life cycle processes described in Applying Life Cycle Processes. This means that, in general, software requirements and architecture processes will be applied alongside system requirements and architecture processes; while software integration and test processes are applied alongside system integration, verification and validation processes. These interrelationships help with vertical software concerns, ensuring detailed software design and construction issues are considered at the system level. They also help with horizontal concerns, ensuring whole system issues are considered and are influenced by an understanding of software. See the Nature of Software for more details.

The ways these related processes work together will depend on the systems approach to solution synthesis used and how this influences the life cycle. If a top down approach is used, problem needs and system architecture will drive software implementation and realization. If a bottom up approach is used, the architecture of existing software will strongly influence both the system solution and the problem which can be considered. In Applying Life Cycle Processes, a "middle-out" approach is described which combines these two ideas and is the most common way to develop systems. This approach needs a two-way relationship between SE and SwE technical processes.

The **SW Support Processes** may also play these vertical and horizontal roles. Part 3 contains knowledge areas on both System Deployment and Use which includes operation, maintenance and logistics; and Systems Engineering Management which covers the project processes shown in Figure 2. SwE support processes focus on the successful vertical deployment and use of software system elements and the management needed to achieve this. They also support their equivalent horizontal SE processes in contributing to the success of the whole system life cycle. The **Software Reuse Processes** have a particularly important role to play in deployment and use and Product and Service Life Management processes. The latter considers Service Life Extension; Capability Updates, Upgrades, and

Modernization; and system Disposal and Retirement. All of these horizontal software engineering activities rely on the associated SE activities having a sufficient understanding of the strengths and limitations of software and SwE (see Key Points a Systems Engineer Needs to Know about Software Engineering).

The Life Cycle Models knowledge area also defines how Vee and Iterative life cycle models provide a framework to tailor the generic life cycle and process definitions to different types of system development. Both models, with some modification, apply equally to the development of products and services containing software. Thus, the simple relationships between SE and SwE processes will form the basis for tailoring to suit project needs within a selected life cycle model.

Software and Systems Challenges

Pyster et al. (2015) define three classes of software intensive systems distinguished by the primary sources of novelty, functionality, complexity and risk in their conception, development, operation and evolution. These are briefly described below:

- **Physical Systems** operate on and generate matter or energy. While they often utilize computation and software technologies as components, those components are not dominant in the horizontal dimension of engineering. Rather, in such systems, they are defined as discrete system elements and viewed and handled as vertical concerns.
- **Computational Systems** include those in which computational behavior and, ipso facto, software are dominant at the systems level. The primary purpose of these systems is to operate on and produce data and information. While these systems always include physical and human elements, these are not the predominant challenges in system development, operation and evolution.
- **Cyber-Physical Systems** are a complex combination of computational and physical dimensions. Such systems are innovative, functionally complex and risky in both their cyber and physical dimensions. They pose major horizontal engineering challenges across the board. In cyber-physical systems, cyber and physical elements collaborate in complex ways to deliver expected system behavior.

Some of the challenges of physical and computational systems are well known and can be seen in many SE and SwE case studies. For example, physical system life cycles often make key decisions about the system architecture or hardware implementation which limit the subsequent development of software architecture and designs. This can lead to software which is inefficient and difficult or expensive to change. Problems which arise later in the life of such systems may be dealt with by changing software or human elements. This is sometimes done in a way which does not fully consider SwE design and testing practices. Similarly, computational systems may be dominated by the software architecture, without sufficient care taken to consider the best solutions for enabling hardware or people. In particular, operator interfaces, training and support may not be considered leading to the need for expensive organizational fixes once they are in use. Many computational systems in the past have been developed without a clear view of the user need they contribute to, or the other systems they must work with to do so. These and other related issues point to a need for system and software engineers with a better understanding of each other's disciplines. Pyster et al. consider how SE and SwE education might be better integrated to help achieve this aim.

Examples of cyber-physical systems increasingly abound – smart automobiles, power grids, robotic manufacturing systems, defense and international security systems, supply-chain systems, the so-called internet of things, etc. In these systems there is no clear distinction between software elements and the whole system solution. The use of software in these systems is central to the physical outcome and software is often the integrating element which brings physical elements and people together. These ideas are closely aligned with the Service System Engineering approach described in Part 4.

SEBoK Part 3 includes a Business and Mission Analysis process which is based on the equivalent process in the updated ISO/IEC/IEEE 15288 (2015). This process enables SE to be involved in the selection and bounding of the problem situation which forms the starting point for an engineered system life cycle. For cyber physical systems, an

understanding of the nature of software is needed in the formulation of the problem, since this is often fundamentally driven by the use of software to create complex adaptive solution concepts. This close coupling of software, physical and human system elements across the system of interest continues throughout the system life cycle making it necessary to consider all three in most horizontal system level decisions.

The life cycle of cyber physical systems cannot be easily partitioned into SE and SwE achieving their own outcomes but working together on horizontal system issues. It will require a much more closely integrated approach, requiring systems and software engineers with a complementary set of competencies, and changes how the two disciplines are seen in both team and organizational structures. See Enabling Systems Engineering.

References

Works Cited

Pyster, A., Adcock, R., Ardis, M., Cloutier, R., Henry, D., Laird, L., Lawson, H. 'Bud'., Pennotti, M., Sullivan, K., Wade J. 2015. "Exploring the relationship between systems engineering and software engineering." 13th Conference on Systems Engineering Research (CSER). In Procedia Computer Science, Volume 44, 2015, pp. 708-717.

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.swebok.org.

ISO/IEC/IEEE. 2015. *Systems and Software Engineering -- System Life Cycle Processes*. Geneva, Switzerland: International Organisation for Standardisation / International Electrotechnical Commissions / Institute of Electrical and Electronics Engineers. ISO/IEC/IEEE 15288:2015.

ISO/IECIEEE. 2008. Systems and Software Engineering — Software Life Cycle Processes. Geneva, Switzerland: International Organization for Standards (ISO)/Institute of Electrical & Electronics Engineers (IEEE) Computer Society, ISO//IECIEEE 12207:2008(E).

Roedler, G. 2011. "Towards Integrated Systems and Software Engineering Standards." National Defense Industrial Association (NDIA) Conference, San Diego, CA, USA.

Primary References

Pyster, A., Adcock, R., Ardis, M., Cloutier, R., Henry, D., Laird, L., Lawson, H. 'Bud'., Pennotti, M., Sullivan, K., Wade J. 2015. Exploring the relationship between systems engineering and software engineering. 13th Conference on Systems Engineering Research (CSER). In Procedia Computer Science, Volume 44, 2015, pp. 708-717.

Additional References

Roedler, G. 2010. *An overview of ISO/IEC/IEE 15288, system life cycle processes.* Asian Pacific Council on Systems Engineering (APCOSE) Conference.

The Nature of Software

Lead Author: *Alice Squires*

The nature of the software medium has many consequences for systems engineering (SE) of software-intensive systems. Fred Brooks has famously observed that four properties of software, taken together, differentiate it from other kinds of engineering artifacts (Brooks 1995). These four properties are:

1. complexity,
2. conformity,
3. changeability,
4. invisibility.

Brooks states:

> *Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level). If they are, we make the two similar parts into a subroutine — open or closed. In this respect, software systems differ profoundly from computers, buildings, or automobiles, where repeated elements abound.* (Brooks 1995, p 82)

Complexity

The complexity of software arises from the large number of unique interacting parts in a software system. The parts are unique because they are encapsulated as functions, subroutines, or objects, and invoked as needed rather than being replicated. Software parts have several different kinds of interactions, including serial and concurrent invocations, state transitions, data couplings, and interfaces to databases and external systems.

Depiction of a software entity often requires several different design representations to portray the numerous static structures, dynamic couplings, and modes of interaction that exist in computer software. Complexity within the parts and in the connections among parts requires that changes undergo substantial design rigor and regression testing. Software provides functionality for components that are embedded, distributed and data centric. Software can implement simple control loops as well as complex algorithms and heuristics.

Complexity can hide defects that may not be discovered easily, thus requiring significant additional and unplanned rework.

Conformity

Software, unlike a physical product, has no underlying natural principles which it must conform to, such as Newton's laws of motion. However, software must conform to exacting specifications in the representation of each of its parts, in the interfaces to other internal parts, and in the connections to the environment in which it operates. A missing semicolon or other syntactic error can be detected by a compiler, but a defect in the program logic or a timing error may be difficult to detect until encountered during operation.

Unlike software, tolerance among the interfaces of physical entities is the foundation of manufacturing and assembly. No two physical parts that are joined together have, or are required to have, exact matches. There are no corresponding tolerances in the interfaces among software entities or between software entities and their environments. There are no interface specifications for software stating that a parameter can be *an integer plus or minus 2%.* Interfaces among software parts must agree exactly in numbers, types of parameters and kinds of couplings.

Lack of conformity can cause problems when an existing software component cannot be reused as planned because it does not conform to the needs of the product under development. Lack of conformity might not be discovered until

late in a project, thus necessitating the development and integration of an acceptable component to replace the one that cannot be reused. This requires an unplanned allocation of resources (usually) and can delay project completion.

Changeability

Software coordinates the operation of physical components and provides most of the functionality in software-intensive systems. Because software is the most malleable (easily changed) element in a software-intensive system, it is the most frequently changed element. This is particularly true during the late stages of a development project and during system sustainment. However, this does not mean that software is easy to change. Complexity and the need for conformity can make changing software an extremely difficult task. Changing one part of a software system often results in undesired side effects in other parts of the system, requiring more changes before the software can operate at maximum efficiency.

Invisibility

Software is said to be invisible because it has no physical properties. While the effects of executing software on a digital computer are observable, software itself cannot be seen, tasted, smelled, touched, or heard. Software is an intangible entity because our five human senses are incapable of directly sensing it.

Work products such as requirements specifications, design documents, source code and object code are representations of software, but they are not the software. At the most elemental level, software resides in the magnetization and current flow in an enormous number of electronic elements within a digital device. Because software has no physical presence, software engineers must use different representations at different levels of abstraction in an attempt to visualize the inherently invisible entity.

Uniqueness

One other point about the nature of software that Brooks alludes to but does not explicitly call out is the uniqueness of software. Software and software projects are unique for the following reasons:

- Software has no physical properties;
- Software is the product of intellect-intensive teamwork;
- Productivity of software developers varies more widely than the productivity of other engineering disciplines;
- Estimation and planning for software projects is characterized by a high degree of uncertainty, which can be at best partially mitigated by best practices;
- Risk management for software projects is predominantly process-oriented;
- Software alone is useless, as it is always a part of a larger system; and
- Software is the most frequently changed element of software intensive systems.

References

Works Cited

Brooks, F. 1995. *The Mythical Man-Month,* Anniversary Edition. Boston, MA, USA: Addison Wesley Longman Inc.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, New Jersey: John Wiley and Sons.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK).* Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Brooks, F. 1995. *The Mythical Man-Month,* Anniversary Edition. Boston, MA, USA: Addison Wesley Longman Inc.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, New Jersey: John Wiley and Sons.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

An Overview of the SWEBOK Guide

Lead Authors: *Heidi Davidz, Alice Squires*

Systems engineers are fortunate that the software community has developed its own body of knowledge. The introduction to Version 3 of the *Guide to the Software Engineering Body of Knowledge* states:

> *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.* (Bourque and Fairley 2014)

SWEBOK Guide Version 3

The purposes of SWEBOK V3 are as follows:

- to characterize the contents of the software engineering discipline;
- to promote a consistent view of software engineering worldwide;
- to clarify the place of, and set the boundary of, software engineering with respect to other disciplines;
- to provide a foundation for training materials and curriculum development; and
- to provide a basis for certification and licensing of software engineers.

SWEBOK V3 contains 15 knowledge areas (KAs). Each KA includes an introduction, a descriptive breakdown of topics and sub-topics, recommended references, references for further reading, and a matrix matching reference material with each topic. An appendix provides a list of standards most relevant to each KA. An overview of the individual KAs presented in the guide is provided in the next two sections.

Knowledge Areas Characterizing the Practice of Software Engineering

Software Requirements

The Software Requirements KA is concerned with the elicitation, negotiation, analysis, specification, and validation of software requirements. It is widely acknowledged within the software industry that software engineering projects are critically vulnerable when these activities are performed poorly. Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems.

Software Design

Design is defined as both *the process of defining the architecture, components, interfaces, and other characteristics of a system or component* and *the result of [that] process* (IEEE 1991). The Software Design KA covers the design process and the resulting product. The software design process is the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the software's internal structure and its behavior that will serve as the basis for its construction. A software design (the result) must describe the software architecture – that is, how software is decomposed and organized into components and the interfaces between those components. It must also describe the components at a level of detail that enables their construction.

Software Construction

Software construction refers to the detailed creation of working software through a combination of detailed design, coding, unit testing, integration testing, debugging, and verification. The Software Construction KA includes topics related to the development of software programs that will satisfy their requirements and design constraints. This KA covers software construction fundamentals; managing software construction; construction technologies; practical considerations; and software construction tools.

Software Testing

Testing is an activity performed to evaluate product quality and to improve it by identifying defects. Software testing involves dynamic verification of the behavior of a program against expected behavior on a finite set of test cases. These test cases are selected from the (usually very large) execution domain. The Software Testing KA includes the fundamentals of software testing; testing techniques; human-computer user interface testing and evaluation; test-related measures; and practical considerations.

Software Maintenance

Software maintenance involves enhancing existing capabilities, adapting software to operate in new and modified operating environments, and correcting defects. These categories are referred to as perfective, adaptive, and corrective software maintenance. The Software Maintenance KA includes fundamentals of software maintenance (nature of and need for maintenance, categories of maintenance, maintenance costs); key issues in software maintenance (technical issues, management issues, maintenance cost estimation, measurement of software maintenance); the maintenance process; software maintenance techniques (program comprehension, re-engineering, reverse engineering, refactoring, software retirement); disaster recovery techniques, and software maintenance tools.

Software Configuration Management

The configuration of a system is the functional and/or physical characteristics of hardware, firmware, software, or a combination of these. It can also be considered as a collection of specific versions of hardware, firmware, or software items combined according to specific build procedures to serve a particular purpose. Software configuration management (SCM) is thus the discipline of identifying the configuration of a system at distinct points in time for the purposes of systematically controlling changes to the configuration, as well as maintaining the integrity and traceability of the configuration throughout the software life cycle. The Software Configuration Management KA covers management of the SCM process; software configuration identification, control, status accounting, and auditing; software release management and delivery; and software configuration management tools.

Software Engineering Management

Software engineering management involves planning, coordinating, measuring, reporting, and controlling a project or program to ensure that development and maintenance of the software is systematic, disciplined, and quantified. The Software Engineering Management KA covers initiation and scope definition (determining and negotiating requirements, feasibility analysis, and review and revision of requirements); software project planning (process planning, estimation of effort, cost, and schedule, resource allocation, risk analysis, planning for quality); software project enactment (measuring, reporting, and controlling; acquisition and supplier contract management); product acceptance; review and analysis of project performance; project closure; and software management tools.

Software Engineering Process

The Software Engineering KA is concerned with definition, implementation, assessment, measurement, management, and improvement of software life cycle processes. Topics covered include process implementation and change (process infrastructure, models for process implementation and change, and software process management); process definition (software life cycle models and processes, notations for process definition, process adaptation, and process automation); process assessment models and methods; measurement (process measurement, products measurement, measurement techniques, and quality of measurement results); and software process tools.

Software Engineering Models and Methods

The Software Engineering Models and Methods KA addresses methods that encompass multiple life cycle stages; methods specific to particular life cycle stages are covered by other KAs. Topics covered include modeling (principles and properties of software engineering models; syntax vs. semantics vs. invariants; preconditions, post-conditions, and invariants); types of models (information, structural, and behavioral models); analysis (analyzing for correctness, completeness, consistency, quality and interactions; traceability; and tradeoff analysis); and software development methods (heuristic methods, formal methods, prototyping methods, and agile methods).

Software Quality

Software quality is a pervasive software life cycle concern that is addressed in many of the SWEBOK V3 KAs. In addition, the Software Quality KA includes fundamentals of software quality (software engineering cultures, software quality characteristics, the value and cost of software quality, and software quality improvement); software quality management processes (software quality assurance, verification and validation, reviews and audits); and practical considerations (defect characterization, software quality measurement, and software quality tools).

Software Engineering Professional Practice

Software engineering professional practice is concerned with the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The Software Engineering Professional Practice KA covers professionalism (professional conduct, professional societies, software engineering standards, employment contracts, and legal issues); codes of ethics; group dynamics (working in teams, cognitive problem complexity, interacting with stakeholders, dealing with uncertainty and ambiguity, dealing with multicultural environments); and communication skills.

Knowledge Areas Characterizing the Educational Requirements of Software Engineering

Software Engineering Economics

The Software Engineering Economics KA is concerned with making decisions within the business context to align technical decisions with the business goals of an organization. Topics covered include fundamentals of software engineering economics (proposals, cash flow, the time-value of money, planning horizons, inflation, depreciation, replacement and retirement decisions); not for-profit decision-making (cost-benefit analysis, optimization analysis); estimation; economic risk and uncertainty (estimation techniques, decisions under risk and uncertainty); and multiple attribute decision making (value and measurement scales, compensatory and non-compensatory techniques).

Computing Foundations

The Computing Foundations KA covers fundamental topics that provide the computing background necessary for the practice of software engineering. Topics covered include problem solving techniques, abstraction, algorithms and complexity, programming fundamentals, the basics of parallel and distributed computing, computer organization, operating systems, and network communication.

Mathematical Foundations

The Mathematical Foundations KA covers fundamental topics that provide the mathematical background necessary for the practice of software engineering. Topics covered include sets, relations, and functions; basic propositional and predicate logic; proof techniques; graphs and trees; discrete probability; grammars and finite state machines; and number theory.

Engineering Foundations

The Engineering Foundations KA covers fundamental topics that provide the engineering background necessary for the practice of software engineering. Topics covered include empirical methods and experimental techniques; statistical analysis; measurements and metrics; engineering design; simulation and modeling; and root cause analysis.

Related Disciplines

SWEBOK V3 also discusses related disciplines. The related disciplines are those that share a boundary, and often a common intersection, with software engineering. SWEBOK V3 does not characterize the knowledge of the related disciplines but, rather, indicates how those disciplines interact with the software engineering discipline. The related disciplines include:

- Computer Engineering
- Computer Science
- General Management

- Mathematics
- Project Management
- Quality Management
- Systems Engineering

References

Works Cited

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.swebok.org.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.swebok.org.

Additional References

None.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Key Points a Systems Engineer Needs to Know about Software Engineering

Lead Author: *Dick Fairley*, ***Contributing Author:*** *Alice Squires*

The field of software engineering is extensive and specialized. Its importance to modern systems makes it necessary for systems engineers to be knowledgeable about software engineering and its relationship to systems engineering.

Key Concepts a Systems Engineer Needs to Know about Software Engineering

The following items are significant aspects that systems engineers need to know about software and software engineering. Most are documented in (Fairley and Willshire 2011):

1. **For the time, effort, and expense devoted to developing it, software is more complex than most other system components** - Software complexity arises because few elements in a software program (even down to the statement level) are identical, as well as because of the large number of possible decision paths found even in small programs, with the number of decision paths through a large program often being astronomical. There are several detailed references on software complexity. The SWEBOK (Bourque and Fairley 2014) discusses minimizing complexity as part of software construction fundamentals. Zuse (1991) has a highly cited article on software complexity measures and methods. Chapters 2 and 3 of the SWEBOK also have further references.
2. **Software testing and reviews are sampling processes** - In all but the simplest cases, exhaustive testing of software is impossible because of the large number of decision paths through most programs. Also, the combined values of the input variables selected from a wide combinatorial range may reveal defects that other combinations of the variables would not detect. Software test cases and test scenarios are chosen in an attempt to gain confidence that the testing samples are representative of the ways the software will be used in practice. Structured

reviews of software are an effective mechanism for finding defects, but the significant effort required limits exhaustive reviewing. Criteria must be established to determine which components (or sub-components) should be reviewed. Although there are similar concerns about exhaustive testing and reviewing of physical products, the complexity of software makes software testing, reviews, and the resulting assurance provided more challenging. Other points include:

1. All software testing approaches and techniques are heuristic. Hence, there is no universal "best" approach, practice, or technique for testing, since these must be selected based on the software context.
2. Exhaustive testing is not possible.
3. Errors in software tend to cluster within the software structures; therefore, any one specific approach or a random approach to testing is not advised.
4. Pesticide paradox exists. As a result, running the same test over and over on the same software-system provides no new information.
5. Testing can reveal the presence of defects but cannot guarantee that there will be no errors, except under the specific conditions of a given test.
6. Testing, including verification and validation (V&V), must be performed early and continually throughout the lifecycle (end to end).
7. Even after extensive testing and V&V, errors are likely to remain after long term use of the software.
8. Chapter 4 of the SWEBOK discusses software testing and provides a bibliography.

3. **Software often provides the interfaces that interconnect other system components** - Software is often referred to as the *glue* that holds a system together because the interfaces among components, as well as the interfaces to the environment and other systems, are often provided by digital sensors and controllers that operate via software. Because software interfaces are behavioral rather than physical, the interactions that occur among software components often exhibit emergent behaviors that cannot always be predicted in advance. In addition to component interfaces, software usually provides the computational and decision algorithms needed to generate command and control signals. The SWEBOK has multiple discussions of interfaces: Chapter 2 on Software Design is a good starting point and includes a bibliography.
4. **Every software product is unique** - The goal of manufacturing physical products is to produce replicated copies that are as nearly identical as much as possible, given the constraints of material sciences and manufacturing tools and techniques. Because replication of existing software is a trivial process (as compared to manufacturing of physical products), the goal of software development is to produce one perfect copy (or as nearly perfect as can be achieved given the constraints on schedule, budget, resources, and technology). Much of software development involves altering existing software. The resulting product, whether new or modified, is uniquely different from all other software products known to the software developers. Chapter 3 of the SWEBOK provides discussion of software reuse and several references.
5. **In many cases, requirements allocated to software must be renegotiated and reprioritized** - Software engineers often see more efficient and effective ways to restate and prioritize requirements allocated to software. Sometimes, the renegotiated requirements have system-wide impacts that must be taken into account. One or more senior software engineers should be, and often are, involved in analysis of system-level requirements. This topic is addressed in the SWEBOK in Chapter 1, with topics on the iterative nature of software and change management.
6. **Software requirements are prone to frequent change** - Software is the most frequently changed component in complex systems, especially late in the development process and during system sustainment. This is because software is perceived to be the most easily changed component of a complex system. This is not to imply that changes to software requirements and the resulting changes to the impacted software can be easily done without undesired side effects. Careful software configuration management is necessary, as discussed in Chapter 6 of the SWEBOK, which includes extensive references.

7. **Small changes to software can have large negative effects** (A corollary to frequently changing software requirements: *There are no small software changes*) - In several well-known cases, modifying a few lines of code in very large systems that incorporated software negatively impacted the safety, security, and/or reliability of those systems. Applying techniques such as traceability, impact analysis, object-oriented software development, and regression testing reduces undesired side effects of changes to software code. These approaches limit but do not eliminate this problem.
8. **Some quality attributes for software are subjectively evaluated** - Software typically provides the interfaces to systems that have human users and operators. The intended users and operators of these systems often subjectively evaluate quality attributes, such as ease of use, adaptability, robustness, and integrity. These quality attributes determine the acceptance of a system by its intended users and operators. In some cases, systems have been rejected because they were not judged to be suitable for use by the intended users in the intended environment, even though those systems satisfied their technical requirements. Chapter 10 of the SWEBOK provides an overview of software quality, with references.
9. **The term *prototyping* has different connotations for systems engineers and software engineers** - For a systems engineer, a prototype is typically the first functioning version of a hardware. For software engineers, software prototyping is primarily used for two purposes: (1) as a mechanism to elicit user requirements by iteratively evolving mock-ups of user interfaces, and (2) as an experimental implementation of some limited element of a proposed system to explore and evaluate alternative algorithms. Chapter 1 of the SWEBOK discusses this and provides excellent references.
10. **Cyber security is a present and growing concern for systems that incorporate software** - In addition to the traditional specialty disciplines of safety, reliability, and maintainability, systems engineering teams increasingly include security specialists at both the software level and the systems level in an attempt to cope with the cyber-attacks that may be encountered by systems that incorporate software. Additional information about security engineering can be found in the Systems Engineering and Specialty Engineering KA.
11. **Software growth requires spare capacity** - Moore's Law no longer fully comes to the rescue (Moore, 1965). As systems adapt to changing circumstances, the modifications can most easily be performed and upgraded in the software, requiring additional computer execution cycles and memory capacity (Belady and Lehman 1979). For several decades, this growth was accommodated by Moore's Law, but recent limits that have occurred as a result of heat dissipation have influenced manufacturers to promote potential computing power growth by slowing down the processors and putting more of them on a chip. This requires software developers to revise their programs to perform more in parallel, which is often an extremely difficult problem (Patterson 2010). This problem is exacerbated by the growth in mobile computing and limited battery power.
12. **Several Pareto 80-20 distributions apply to software** - These refers to the 80% of the avoidable rework that comes from 20% of the defects, that 80% of the defects come from 20% of the modules, and 90% of the downtime comes from at most 10% of the defects (Boehm and Basili 2001). These, along with recent data indicating that 80% of the testing business value comes from 20% of the test cases (Bullock 2000), indicate that much more cost-effective software development and testing can come from determining which 20% need the most attention.
13. **Software estimates are often inaccurate** - There are several reasons software estimates are frequently inaccurate. Some of these reasons are the same as the reasons systems engineering estimates are often inaccurate: unrealistic assumptions, vague and changing requirements, and failure to update estimates as conditions change. In addition, software estimates are often inaccurate because productivity and quality are highly variable among seemingly similar software engineers. Knowing the performance characteristics of the individuals who will be involved in a software project can greatly increase the accuracy of a software estimate. Another factor is the cohesion of the software development team. Working with a team that has worked together before and knowing their collective performance characteristics can also increase the accuracy of a software estimate. Conversely, preparing an estimate for unknown teams and their members can result in a very low degree of accuracy. Chapter

7 of the SWEBOK briefly discusses this further. Kitchenam (1997) discusses the organizational context of uncertainty in estimates. Lederer and Prasad (1995) also identify organizational and management issues that increase uncertainty; additionally, a recent dissertation from Sweden by Magazinus (2012) shows that the issues persist.

14. **Most software projects are conducted iteratively** - "Iterative development" has a different connotation for systems engineers and software engineers. A fundamental aspect of iterative software development is that each iteration of a software development cycle adds features and capabilities to produce a next working version of partially completed software. In addition, each iteration cycle for software development may occur on a daily or weekly basis, while (depending on the scale and complexity of the system) the nature of physical system components typically involves iterative cycles of longer durations. Classic articles on this include (Royce 1970) and (Boehm 1988), among others. Larman and Basili (2003) provide a history of iterative development, and the SWEBOK discusses this in life cycle processes in Chapter 8.
15. **Teamwork within software projects is closely coordinated** - The nature of software and its development requires close coordination of work activities that are predominately intellectual in nature. Certainly, other engineers engage in intellectual problem solving, but the collective and ongoing daily problem solving required of a software team requires a level of communication and coordination among software developers that is of a different, more elevated type. Highsmith (2000) gives a good overview.
16. **Agile development processes are increasingly used to develop software** - Agile development of software is a widely used and growing approach to developing software. Agile teams are typically small and closely coordinated, for the reasons cited above. Multiple agile teams may be used on large software projects, although this is highly risky without an integrating architecture (Elssamadisy and Schalliol 2002). Agile development proceeds iteratively in cycles that produce incremental versions of software, with cycle durations that vary from one day to one month, although shorter durations are more common. Among the many factors that distinguish agile development is the tendency to evolve the detailed requirements iteratively. Most agile approaches do not produce an explicit design document. Martin (2003) gives a highly cited overview.
17. **Verification and validation (V&V) of software should preferably proceed incrementally and iteratively** - Iterative development of working product increments allows incremental verification, which ensures that the partial software product satisfies the technical requirements for that incremental version; additionally, it allows for the incremental validation (ISO/IEC/IEEE 24765) of the partial product to make certain that it satisfies its intended use, by its intended users, in its intended environment. Incremental verification and validation of working software allows early detection and correction of encountered problems. Waiting to perform integration, verification, and validation of complex systems until later life cycle stages, when these activities are on the critical path to product release, can result in increased cost and schedule impacts. Typically, schedules have minimal slack time during later stages in projects. However, with iterative V&V, software configuration management processes and associated traceability aspects may become complex and require special care to avoid further problems. Chapter 4 of the SWEBOK discusses software testing, and provides numerous references, including standards. Much has been written on the subject; a representative article is (Wallace and Fujii 1989).
18. **Performance trade-offs are different for software than systems** - Systems engineers use “performance” to denote the entire operational envelope of a system; whereas software engineers use “performance” to mean response time and the throughput of software. Consequentially, systems engineers have a larger design space in which to conduct trade studies. In software, performance is typically enhanced by reducing other attributes, such as security or ease of modification. Conversely, enhancing attributes such as security and ease of modification typically impacts performance of software (response time and throughput) in a negative manner.
19. **Risk management for software projects differs in kind from risk management for projects that develop physical artifacts** - Risk management for development of hardware components is often concerned with issues such as supply chain management, material science, and manufacturability. Software and hardware share some similar risk factors: uncertainty in requirements, schedule constraints, infrastructure support, and resource

availability. In addition, risk management in software engineering often focuses on issues that result from communication problems and coordination difficulties within software development teams, across software development teams, and between software developers and other project members (e.g., hardware developers, technical writers, and those who perform independent verification and validation). See (Boehm 1991) for a foundational article on the matter.

20. **Software metrics include product measures and process measures** - The metrics used to measure and report progress of software projects include product measures and process (ISO/IEC/IEEE 24765) measures. Product measures include the amount of software developed (progress), defects discovered (quality), avoidable rework (defect correction), and budgeted resources used (technical budget, memory and execution cycles consumed, etc.). Process measures include the amount of effort expended (because of the people-intensive nature of software development), productivity (software produced per unit of effort expended), production rate (software produced per unit time), milestones achieved and missed (schedule progress), and budgeted resources used (financial budget). Software metrics are often measured on each (or, periodically, some) of the iterations of a development project that produces a next working version of the software. Chapter 8 and Chapter 7 of the SWEBOK address this.
21. **Progress on software projects is sometimes inadequately tracked** - In some cases, progress on software projects is not adequately tracked because relevant metrics are not collected and analyzed. A fundamental problem is that accurate tracking of a software project depends on knowing how much software has been developed that is suitable for delivery into the larger system or into a user environment. Evidence of progress in the form of working software is one of the primary advantages of the iterative development of working software increments.

References

Works Cited

Belady, L. and M. Lehman. 1979. "Characteristics of large systems." In P. Wegner (ed.), *Research Directions in Software Technology*. Cambridge, MA, USA: MIT Press.

Boehm, B. 1991. "Software risk management: Principles and practices." *IEEE Software*. 8(1):32-41.

Boehm, B. and V. Basili. 2001. "Software defect reduction Top 10 List." *Computer*. 34(1):135-137.

Brooks, F. 1995. *The Mythical Man-Month, Anniversary Edition*. Boston, MA, USA: Addison Wesley Longman Inc.

Bullock, J. 2000. "Calculating the value of testing." *Software Testing and Quality Engineering*, May-June, 56-62.

DeMarco, T. and T. Lister. 1987. *Peopleware: Predictive Projects and Teams*. New York, NY, USA: Dorset House.

Elssamadisy, A. and G. Schalliol. 2002. "Recognizing and responding to 'bad smells' in extreme programming." *Proceedings, ICSE 2002*, ACM-IEEE, 617-622.

Fairley, R.E. and M.J. Willshire. 2011. "Teaching software engineering to undergraduate systems engineering students." *Proceedings of the 2011 American Society for Engineering Education (ASEE) Annual Conference and Exposition*. 26-29 June 2011. Vancouver, BC, Canada.

Kitchenham, B. 1997. "Estimates, uncertainty, and risk." *IEEE Software*. 14(3): 69-74.

Larman, C. and V.R. Basili. 2003. "Iterative and incremental developments: A brief history." *Computer*. 36(6): 47-56.

Lederer, A.L. and J. Prasad. 1995. "Causes of inaccurate software development cost estimates." *Journal of Systems and Software*. 31(2):125-134.

Magazinius, A. 2012. *Exploring Software Cost Estimation Inaccuracy*. Doctoral Dissertation. Chalmers University of Technology. Goteborg, SE.

Martin, R.C. 2002. *Agile Software Development: Principles, Patterns and Practices*. Upper Saddle River, NJ, USA: Prentice Hall.

Moore, G.E. 1965. "Cramming more components onto integrated circuits," *Electronics Magazine*, April 19, 4.

Patterson, D. 2010. "The trouble with multicore." *IEEE Spectrum*, July, 28-32, 52-53.

Royce, W.W. 1970. "Managing the development of large software systems." *Proceedings of IEEE WESCON*. August 1970.

Wallace, D.R. and R.U. Fujii. 1989. "Software verification and validation: An overview." *IEEE Software*. 6(3):10-17.

Zuse, Horst. 1991. *Software complexity: Measures and methods*. Hawthorne, NJ, USA: Walter de Gruyter and Co.

Primary References

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Brooks, Fred 1995. The Mythical Man-Month, Anniversary Edition. Reading, Massachusetts: Addison Wesley.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, NJ, USA: John Wiley & Sons.

PMI. 2013A. *A Guide to the Project Management Body of Knowledge|(PMBOK® Guide)*. 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

PMI 2013B. *Software Extension to the PMBOK® Guide*, Fifth Edition, Newtown Square, PA, USA: Project Management Institute (PMI) and Los Alamitos, CA, USA: IEEE Computer Society.

Pyster, A., M. Ardis, D. Frailey, D. Olwell, A. Squires. 2010. "Global workforce development projects in software engineering." *Crosstalk - The Journal of Defense Software Engineering*, Nov/Dec, 36-41. Available at: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA535633 Accessed 02 Dec 2015.

< Previous Article | Parent Article | Next Article >

Software Engineering Features - Models, Methods, Tools, Standards, and Metrics

Lead Author: *Tom Hilburn*

In recent decades, software has become ubiquitous. Almost all modern engineered systems include significant software subsystems; this includes systems in the transportation, finance, education, healthcare, legal, military, and business sectors. Along with the increase in software utility, capability, cost, and size there has been a corresponding growth in methods, models, tools, metrics and standards, which support software engineering.

Chapter 10 of the SWEBOK discusses modeling principles and types, and the methods and tools that are used to develop, analyze, implement, and verify the models. The other SWEBOK chapters on the software development phases (e.g., Software Design) discuss methods and tools specific to the phase. Table 1 identifies software engineering features for different life-cycle phases. The table is not meant to be complete; it simply provides examples. In Part 2 of the SEBoK there is a discussion of models and the following is one of the definitions offered: “an abstraction of a system, aimed at understanding, communicating, explaining, or designing aspects of interest of that system” (Dori 2002).

For the purposes of Table 1 the definition of a model is extended to some aspect of the software system or its development. As an example, “Project Plan” is listed as a model in the Software Management area. The idea is that the Project Plan provides a model of how the project is going to be carried out: the project team organization, the process to be used, the work to be done, the project schedule, and the resources needed.

Table 1: SWE Features (SEBoK Original)

Life-Cycle Activity	Models	Methods & Tools	Standards
Software Management	• Life-Cycle Process Model • Work Breakdown Structure • Constructive Cost Model (COCOMO) • Project Plan • Configuration Management (CM) Plan • Risk Management Plan	• Effort, Schedule and Cost Estimation • Risk Analysis • Data Collection • Project Tracking • CM Management • Iterative/Incremental Development • Agile Development	• [IEEE 828] • [IEEE 1058] • [IEEE 1540] • [IEEE 12207]
Software Requirements	• Functional Model • User Class Model • Data Flow Diagram • Object Model • Formal Model • User Stories	• Requirements Elicitation • Prototyping • Structural Analysis • Data-Oriented Analysis • Object-Oriented Analysis • Object Modeling Language (OML) • Formal Methods • Requirements Specification • Requirements Inspection	• [IEEE 830] • [IEEE 1012] • [IEEE 12207]
Software Design	• Architectural Model • Structure Diagram • Object Diagram • Class Specification • Data Model	• Structured Design • Object-Oriented Design • OML • Modular Design • Integrated Development Environment (IDE) • Database Management System (DBMS) • Design Review • Refinement	• [IEEE 1012] • [IEEE 1016] • [IEEE 12207] • [IEEE 42010]

Software Construction	• Detail Design Document • Pseudocode • Flow Chart • Program Code • Unit Test Plan • Integration Test Plan	• Detailed Design • Functional Programming • Object-Oriented Programming • IDE • DBMS • Black Box/White Box Testing • Basic Path Testing • Unit Testing • Code Review • Proof of Correctness • Software Reuse • Integration • Integration Testing	• [IEEE 1008] • [IEEE 1012] • [IEEE 1016] • [IEEE 12207]
Software Testing	• System Test Plan • Reliability Model • Software Maintenance Process	• Usability Testing • System Testing • Acceptance Testing • Regression Testing • Reliability Testing • Non-Functional Software Testing	• [IEEE 829] • [IEEE 1012] • [IEEE 12207]
Software Maintenance	• Software Maintenance Process	• Automated Testing Tools • Maintenance Change • Impact Analysis • Inventory Analysis • Restructuring • Reverse Engineering • Re-engineering	• [IEEE 1219] • [IEEE 12207] • [IEEE 14764]

Software Metric

A software metric is a quantitative measure of the degree a software system, component, or process possesses a given attribute. Because of the abstract nature of software and special problems with software schedule, cost, and quality, data collection and the derived metrics are an essential part of software engineering. This is evidenced by the repeated reference to measurement and metrics in the SWEBOK. Table 2 describes software metrics that are collected and used in different areas of software development. As in Table 1 the list is not meant to be complete, but to illustrate the type and range of measures used in practice.

Table 2: Software Metrics * (SEBoK Original)

Category	Metrics
Management Metrics	• Size: Lines of Code (LOC*), Thousand Lines of Code (KLOC) • Size: Function points, Feature Points • Individual Effort: Hours • Task Completion Time: Hours, Days, Weeks • Project Effort: Person-Hours • Project Duration: Months • Schedule: Earned Value • Risk Projection: Risk Description, Risk Likelihood, Risk Impact
Software Quality Metrics	• Defect Density - Defects/KLOC (e.g., for system test) • Defect Removal Rate – Defects Removed/Hour (for review and test) • Test Coverage • Failure Rate

Software Requirements Metrics	• Change requests (received, open, and closed) • Change request frequency • Effort required to implement a requirement change • Status of requirements traceability • User stories in the backlog
Software Design Metrics	• Cyclomatic Complexity • Weighted Methods per Class • Cohesion - Lack of Cohesion of Methods • Coupling - Coupling Between Object Classes • Inheritance - Depth of Inheritance Tree, Number of Children
Software Maintenance and Operation	• Mean Time Between Changes (MTBC) • Mean Time to Change (MTTC) • System Reliability • System Availability • Total Hours of Downtime

**Note: Even though the LOC metric is widely used, using it comes with some problems and concerns: different languages, styles, and standards can lead to different LOC counts for the same functionality; there are a variety of ways to define and count LOC– source LOC, logical LOC, with or without comment lines, etc.; and automatic code generation has reduced the effort required to produce LOC.*

References

Works Cited

Bourque, P. and R.E. Fairley (eds.). 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. Los Alamitos, CA, USA: IEEE Computer Society. Available at: http://www.Swebok.org.

Dori, D. 2003. "Conceptual modeling and system architecting." *Communications of the ACM*, 46(10), pp. 62-65.

[IEEE 828] IEEE Computer Society, IEEE Standard for *Computer Configuration Management in Systems and Software Engineering*, IEEE Std 828- 2012, 20012.

[IEEE 829] IEEE Computer Society, IEEE Standard for *Software and System Test Documentation*, IEEE Std 829-2008, 2008.

[IEEE 830] IEEE Computer Society, IEEE Standard for *Recommended Practice for Software Requirements Specifications*, IEEE Std 830-1998, 1998.

[IEEE 1008] IEEE Computer Society, IEEE Standard for *Software Unit Testing,* IEEE Std 1008-1987, 1987.

[IEEE 1012] IEEE Computer Society, IEEE Standard for *System and Software Verification and Validation*, IEEE Std 1012-2002, 2012.

[IEEE 1016] IEEE Computer Society, IEEE Standard for *Recommended Practice for Software Design Descriptions*, IEEE Std 1016-2002, 2002.

[IEEE 1058] IEEE Computer Society, IEEE Standard for *Software Project Plans*, IEEE Std 1058-1998, 1998.

[IEEE 1219] IEEE Computer Society, IEEE Standard for *Software Maintenance*, IEEE Std 1219-1998, 1998.

[IEEE 1540] IEEE Computer Society, IEEE Standard for *Risk Management*, IEEE Std 1540-2001, 2001.

[IEEE 12207] IEEE Computer Society, IEEE Standard for *Systems and Software Engineering —Software Life Cycle Processes*, IEEE Std 12207-2008, 2008.

[IEEE 14764] IEEE Computer Society, IEEE Standard for *Software Engineering - Software Life Cycle Processes - Maintenance*. IEEE Std 14764-2006, 2006.

[IEEE 42010] IEEE Computer Society, IEEE Standard for *Systems and Software Engineering — Architecture Description*, IEEE Std 42010-2011, 2011.

Additional References

Chidamber, S.R., C.F. Kemerer. 1994. "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*. Vol. 20, No. 6. June 1994.

Kan, Stephen H. 2003. *Metrics and Models in Software Quality Engineering*, 2nd edition. Reading, Massachusetts, USA: Addison-Wesley.

Li, M. and Smidts, C. 2003. "A ranking of software engineering measures based on expert opinion." *IEEE Transactions on Software Engineering*. September 2003.

McConnell, Steve. 2009. *Code Complete*, 2nd Ed. Microsoft Press.

Moore, James. 1997. *Software Engineering Standards: A User's Road Map*. Hoboken, NJ, USA: Wiley-IEEE Computer Society Press.

Sommerville, I. 2010. *Software Engineering*. 9th Ed. Boston, MA, USA: Addison Wesley.

< Previous Article | Parent Article | Next Article >

Knowledge Area: Systems Engineering and Project Management

Systems Engineering and Project Management

Lead Author: *Dick Fairley*, ***Contributing Authors:*** *Richard Turner, Alice Squires*

The goal of project management is to plan and coordinate the work activities needed to deliver a satisfactory product, service, or enterprise endeavor within the constraints of schedule, budget, resources, infrastructure, and available staffing and technology. The purpose of this knowledge area (KA) is to acquaint systems engineers with the elements of project management and to explain the relationships between systems engineering (SE) and project management (PM).

Topics

Each part of the SEBoK is divided into knowledge areas (KAs), which are groupings of information with a related theme. The KAs, in turn, are divided into topics. This KA contains the following topics:

- The Nature of Project Management
- An Overview of the PMBOK® Guide
- Relationships between Systems Engineering and Project Management
- The Influence of Project Structure and Governance on Systems Engineering and Project Management Relationships
- Procurement and Acquisition

References

Works Cited

None.

Primary References

Fairley, R.E. 2009. *Managing and Leading Software Projects*. Hoboken, NJ, USA: John Wiley & Sons.

Forsberg, K., H. Mooz, and H. Cotterman. 2005. *Visualizing Project Management*, 3rd ed. New York, NY, USA: John Wiley & Sons.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

None.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

The Nature of Project Management

Lead Author: *Heidi Davidz*, ***Contributing Authors:*** *Richard Turner, Alice Squires*

While *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)* provides an overview of project management for those seeking PMI certification, Fairley (2009) and Forsberg (2005) suggest another way to characterize the important aspects of project management:

- Planning and Estimating
- Measuring and Controlling
- Leading and Directing
- Managing Risk

Introduction

Project managers and systems engineers are both concerned with management issues such as planning, measuring and controlling, leading, directing, and managing risk. In the case of project managers, the project attributes to be managed include project plans; estimates; schedule; budget; project structure; staffing; resources; infrastructure; and risk factors. Product attributes managed by systems engineers include items such as requirements allocation and flow-down; system architecture; structure of and interactions among technical teams; specialty engineering; integration; verification; and validation.

The exact allocation of the SE and PM duties depend on many factors, such as customer and stakeholder interactions, organizational structure of the parent organization, and relationships with affiliate contractors and subcontractors. (See the article on The Influence of Project Structure and Governance on Systems Engineering and Project Management Relationships in this KA.)

Planning and Estimating

Planning

Planning a project involves providing answers to the who, what, where, when, and why of every project:

- **Who:** Addresses staffing issues (competencies, numbers of staff, communication and coordination)
- **What:** Addresses the scope of activities
- **Where:** Addresses issues of locale (local, geographically distributed)
- **When:** Addresses scheduling issues
- **Why:** Addresses rationale for conducting a project

Guidance for developing project plans can be found in INCOSE (2012), NASA (2007), and ISO/IEC/IEEE Standard 16326:2009. It is often observed that communication and coordination among stakeholders during project planning are equally as important as (and sometimes more important than) the documented plan that is produced.

In defense work, event-driven integrated master plans and time-driven integrated master schedules are planning products. Chapter 11 of the Defense Acquisition Guidebook provides details (DAU 2010).

Estimating

Estimation is an important element of planning. An estimate is a projection from past to future, adjusted to account for differences between past and future. Estimation techniques include analogy, rule of thumb, expert judgment, and use of parametric models such as the PRICE model for hardware, COCOMO for software projects and COSYSMO for systems projects (Stewart 1990; Boehm et al. 2000; Valerdi 2008).

Entities estimated include (but are not limited to) schedule, cost, performance, and risk.

Systems engineering contributes to project estimation efforts by ensuring that:

- the overall system life cycle is understood;
- dependencies on other systems and organizations are identified;
- the logical dependencies during development are identified; and
- resources and key skills are identified and planned.

Additionally, high-level system architecture and risk assessment provide the basis for both the work breakdown structure and the organizational breakdown structure.

Measuring and Controlling

Measuring and controlling are the key elements of executing a project. Measurement includes collecting measures for work products and work processes. For example, determining the level of coverage of requirements in a design specification can be assessed through review, analysis, prototyping, and traceability. Effort and schedule expended on the work processes can be measured and compared to estimates; earned value tracking can be used for this purpose. Controlling is concerned with analyzing measurement data and implementing corrective actions when actual status does not align with planned status.

Systems engineers may be responsible for managing all technical aspects of project execution, or they may serve as staff support for the project manager or project management office. Organizational relationships between systems engineers and project managers are presented in Team Capability. Other organizational considerations for the relationships between systems engineering and project management are covered in the Enabling Systems Engineering knowledge area.

Additional information on measurement and control of technical factors can be found in the Measurement and Assessment and Control articles in Part 3: Systems Engineering and Management.

Leading and Directing

Leading and directing requires communication and coordination among all project stakeholders, both internal and external. Systems engineers may be responsible for managing all technical aspects of project execution, or they may serve as staff support for the project manager or project management office. Organizational relationships between systems engineers and project managers are presented in the article Team Capability in Part 5. Other organizational considerations for the relationships between systems engineering and project management are discussed in Part 5: Enabling Systems Engineering.

Managing Risk

Risk management is concerned with identifying and mitigating potential problems before they become real problems. Systems engineering projects are, by nature, high-risk endeavors because of the many unknowns and uncertainties that are inherent in projects. Because new risk factors typically emerge during a project, ongoing continuous risk management is an important activity for both systems engineers and project managers.

Potential and actual problems may exist within every aspect of a project. Systems engineers are typically concerned with technical risk and project managers with programmatic risk. Sometimes, technical risk factors are identified and confronted by systems engineers and programmatic risk factors are identified and confronted by project managers without adequate communication between them. In these cases, appropriate tradeoffs among requirements, schedule, budget, infrastructure, and technology may not be made, which creates additional risk for the successful outcome of a project.

In the last ten years, there has been an increasing interest in opportunity management as the converse of risk management. Hillson (2003), Olsson (2007), and Chapman and Ward (2003) provide highly cited introductions.

Additional information on risk management for systems engineering projects can be found in the Risk Management article in Part 3: Systems Engineering and Management.

References

Works Cited

Boehm, B., C. Abts, A.W. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. 2000. *Software Cost Estimation with COCOMO II.* Upper Saddle River, NJ, USA: Prentice Hall.

Chapman, C., and S. Ward. 2003. *Project Risk Management: Processes, Techniques and Insights.* Chichester, West Sussex, England, UK: John Wiley & Sons.

DAU. 2010. *Defense Acquisition Guidebook (DAG)*. Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/U.S. Department of Defense (DoD). February 19, 2010.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, NJ, USA: John Wiley & Sons.

Forsberg, K., H. Mooz, and H. Cotterman. 2005. *Visualizing Project Management.* Hoboken, NJ, USA: John Wiley & Sons.

Hillson, D. 2003. *Effective Opportunity Management for Projects: Exploiting Positive Risk.* Boca Raton, FL, USA: CRC Press.

INCOSE. 2012. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities,* version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2.

ISO/IEC/IEEE. 2009. ISO/IEC/IEEE 16326:2009(E). *Systems and Software Engineering - Life Cycle Processes - Project Management*. Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC)/Institute of Electrical and Electronics Engineers (IEEE).

NASA. 2007. *Systems Engineering Handbook*. Washington, D.C., USA: National Aeronautics and Space Administration.

Olsson, Rolf. 2007. "In search of opportunity management: Is the risk management process enough?" *International Journal of Project Management*, 25 (8), 745–752, 2011.

Stewart, Rodney. 1990. *Cost Estimating.* New York, NY, USA: Wiley.

Valerdi, R. *The Constructive Systems Engineering Cost Model (COSYSMO): Quantifying the Costs of Systems Engineering Effort.* Saarbrucken, Germany: VDM Verlag.

Primary References

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, NJ, USA: John Wiley & Sons.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

Blanchard, B. 2008. *System Engineering Management.* Hoboken, NJ, USA: John Wiley & Sons.

Kerzner, Harold. 2003. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 8th ed. Hoboken, NJ, USA: John Wiley & Sons.

Martin, J. 1997. *Systems Engineering Guidebook: A Process for Developing Systems and Products.* London, UK: Taylor and Francis Group CRC-Press, LLC.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

An Overview of the PMBOK® Guide

Lead Author: *Richard Turner*, ***Contributing Author:*** *Alice Squires*

The *Guide to the Project Management Book of Knowledge (PMBOK® Guide)* is published and maintained by the Project Management Institute (PMI). It is acknowledged as the authoritative documentation of good practices in project management. It is also the basis for certification exams to qualify Project Management Professionals (PMPs). Many organizations require PMP certification as a basic qualification for the role of project manager.

Overview

According to Section 1.3 of the *PMBOK® Guide*, project management is *accomplished through the appropriate application and integration of the 47 logically grouped project management processes, which are categorized into five Process Groups* (PMI 2013). The five Process Groups are:

1. Initiating Process Group
2. Planning Process Group
3. Executing Process Group
4. Monitoring and Controlling Process Group
5. Closing Process Group

Each of the 47 processes is specified by Inputs, Tools & Techniques, and Outputs. Data flow diagrams are used in the PMBOK to illustrate the relationships between each process and the other processes in which each process interacts. The processes are also grouped into ten Knowledge Areas. These Knowledge Areas are:

1. Project Integration Management
2. Project Scope Management
3. Project Time Management
4. Project Cost Management
5. Project Quality Management
6. Project Human Resources Management
7. Project Communications Management
8. Project Risk Management

9. Project Procurement Management
10. Project Stakeholder Management

The five process groups are discussed in more detail in the following section.

Initiating Process Group

Activities performed in the **Initiating** process group include: obtaining authorization to start a project; defining the high-level scope of the project; developing and obtaining approval for the project charter; performing key stakeholder analysis; and identifying and documenting high-level risks, assumptions, and constraints. The **Initiating** process group contains two processes: develop the project charter and identify stakeholders.

Planning Process Group

The **Planning** process group consists of 24 processes, including: assessing detailed project requirements, constraints, and assumptions with stakeholders; developing the project management plan; creating the work breakdown structure; developing a project schedule; determining a project budget; and planning for quality management, human resource management, communication management, change and risk management, procurement management, and stakeholder management. The integrated project management plan is presented to key stakeholders.

Executing Process Group

The **Executing** process group includes eight processes that involve performing the work necessary to achieve the stated objectives of the project. Activities include: obtaining and managing project resources; executing the tasks defined in the project plan; implementing approved changes according to the change management plan; performing quality assurance; acquiring, developing, and managing the project team; managing communications; conducting procurements; and managing stakeholder engagement.

Monitoring and Controlling Process Group

The **Monitoring and Controlling** process group is comprised of 11 processes that include: validate and control scope; control schedule; control cost; control quality; control communications; control risks; control procurements; and control stakeholder engagement. Activities include: measuring project performance and using appropriate tools and techniques; managing changes to the project scope, schedule, and costs; ensuring that project deliverables conform to quality standards; updating the risk register and risk response plan; assessing corrective actions on the issues register; and communicating project status to stakeholders.

Closing Process Group

The **Closing** process group involves two processes: closing project or phase and closing procurements. Closing the project or phase involves finalizing all project activities, archiving documents, obtaining acceptance for deliverables, and communicating project closure. Other activities include: transferring ownership of deliverables; obtaining financial, legal, and administrative closure; distributing the final project report; collating lessons learned; archiving project documents and materials; and measuring customer satisfaction.

The scope of project management, as specified in the PMBOK Guide, encompasses the total set of management concerns that contribute to successful project outcomes.

References

Works Cited

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Primary References

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

Blanchard, B. 2008. *System Engineering Management*. Hoboken, NJ, USA: John Wiley & Sons.

Fairley, R.E. 2009. *Managing and Leading Software Projects*. Hoboken, NJ, USA: John Wiley & Sons.

Martin, J. 1997. *Systems Engineering Guidebook: A Process for Developing Systems and Products*. London, UK: Taylor and Francis Group CRC-Press, LLC.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Relationships between Systems Engineering and Project Management

Lead Author: *Richard Turner*, ***Contributing Author:*** *Alice Squires*

This topic discusses the relationship between systems engineering (SE) and project management (PM). As with software engineering, there is a great deal of overlap. Depending on the environment and organization, the two disciplines can be disjoint, partially intersecting, or one can be seen as a subset of the other. While there is no standard relationship, the project manager and the systems engineer encompass the technical and managerial leadership of a project between them, which requires the enterprise of each project manager and system engineer to work out the particular details for their own context.

Overlap

There is a great deal of significant overlap between the scope of systems engineering, as described here (in the SEBoK), CMMI (2011), and other resources and the scope of project management, as described in the *PMBOK® Guide* (PMI 2013), CMMI (2011), and other resources as illustrated in Figure 1.

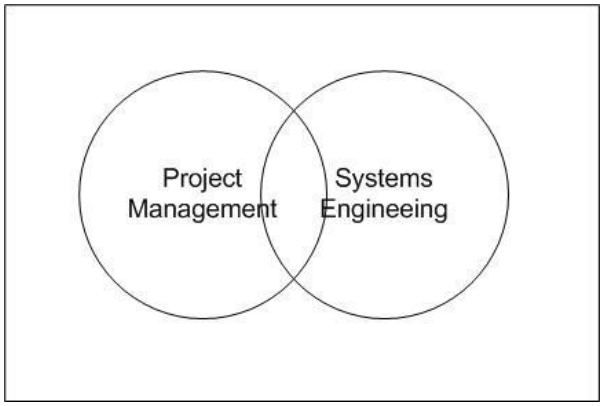


Figure 1. Overlap of PM and SE. (SEBoK Original)

These sources describe the importance of understanding the scope of the work at hand, how to plan for critical activities, how to manage efforts while reducing risk, and how to successfully deliver value to a customer. The systems engineer working on a project will plan, monitor, confront risk, and deliver the technical aspects of the project, while the project manager is concerned with the same kinds of activities for the overall project. Because of these shared concerns, at times there may be confusion and tension between the roles of the project manager and the systems engineer on a given project. As shown in Figure 2, on some projects there is no overlap in responsibility. On other projects, there may be shared responsibilities for planning and managing activities. In some cases, particularly for smaller projects, the project manager may also be the lead technical member of the team performing both roles of project manager and systems engineer.

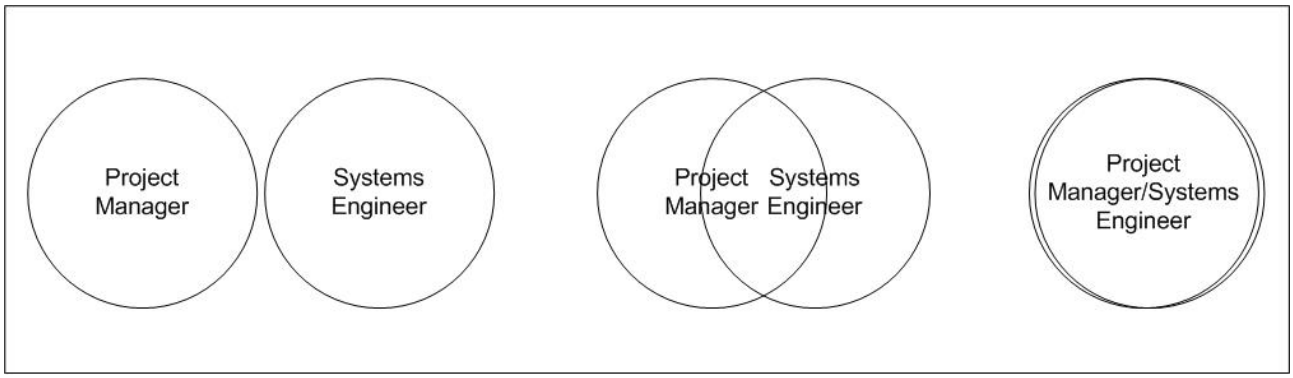


Figure 2. Overlap of Project Roles. (SEBoK Original)

Defining Roles and Responsibilities

Regardless of how the roles are divided up on a given project, the best way to reduce confusion is to explicitly describe the roles and responsibilities of the project manager and the systems engineer, as well as other key team members. The Project Management Plan (PMP) and the Systems Engineering Management Plan (SEMP) are key documents used to define the processes and methodologies the project will employ to build and deliver a product or service.

The PMP is the master planning document for the project. It describes all activities, including technical activities, to be integrated and controlled during the life of the program. The SEMP is the master planning document for the systems engineering technical elements. It defines SE processes and methodologies used on the project and the relationship of SE activities to other project activities. The SEMP must be consistent with and evolve in concert with the PMP. In addition, some customers have technical management plans and expectations that the project's SEMP integrate with customer plans and activities. In the U.S. Department of Defense, most government project teams have a systems engineering plan (SEP) with an expectation that the contractor's SEMP will integrate and remain consistent with customer technical activities. In cases where the project is developing a component of a larger system, the component project's SEMP will need to integrate with the overall project's SEMP.

Given the importance of planning and managing the technical aspects of the project, an effective systems engineer will need to have a strong foundation in management skills and prior experience, as well as possess strong technical depth. From developing and defending basis of estimates, planning and monitoring technical activities, identifying and mitigating technical risk, and identifying and including relevant stakeholders during the life of the project, the systems engineer becomes a key member of the project's management and leadership team. Additional information on Systems Engineering Management and Stakeholder Needs and Requirements can be found in Part 3: Systems Engineering and Management.

Practical Considerations

Effective communication between the project manager and the system engineer is essential for mission accomplishment. This communication needs to be established early and occur frequently.

Resource reallocation, schedule changes, product/system changes and impacts, risk changes: all these and more need to be quickly and clearly discussed between the PM and SE.

References

Works Cited

CMMI. 2011. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Old Tappan, NJ, USA: Pearson Education.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Primary References

Chrissis, M.B, M. Konrad, S. Shrum. 2011. *CMMI for Development: Guidelines for Process Integration and Product Improvement*, 3rd ed. Boston, MA, USA: Addison-Wesley Professional.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Additional References

NASA. 2007. *Systems Engineering Handbook*, Revision 1. Washington, DC, USA: National Aeronautics and Space Administration (NASA). NASA/SP-2007-6105.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

The Influence of Project Structure and Governance on Systems Engineering and Project Management Relationships

Lead Author: *Alice Squires*

This article reviews various project structures that impact or provide governance to the project and that require key involvement from the program manager and the systems engineer. These structures include: the structure of the organization itself (functional, project, matrix, and specialized teams, such as Integrated Product Teams (IPTs), Change Control Boards (CCBs), and Engineering Review Boards (ERBs). This article also addresses the influence of schedule-driven versus requirements-driven projects on these structures.

Relationships between Systems Engineering and Project Management are covered in a related article.

An Overview of Project Structures

Project management and systems engineering governance are dependent on the organization's structure. For some projects, systems engineering is subordinated to project management and in other cases, project management provides support to systems engineering. These alternatives are illustrated in Figures 1 and 2 of the Organizing the Team section in Team Capability.

A project exists within the structural model of an organization. Projects are one-time, transient events that are initiated to accomplish a specific purpose and are terminated when the project objectives are achieved. Sometimes, on small projects, the same person accomplishes the work activities of both project management and systems engineering. Because the natures of the work activities are significantly different, it is sometimes more effective to have two persons performing project management and systems engineering, each on a part-time basis. On larger projects there are typically too many tasks to be accomplished for one person to accomplish all of the necessary work. Very large projects may have project management and systems engineering offices with a designated project manager and a designated lead systems engineer.

Projects are typically organized in one of three ways: (1) by functional structure, (2) by project structure, and (3) by a matrix structure (see Systems Engineering Organizational Strategy for a fourth structure and related discussion). In a function-structured organization, workers are grouped by the functions they perform. The systems engineering functions can be: (1) distributed among some of the functional organizations, (2) centralized within one organization

or (3) a hybrid, with some of the functions being distributed to the projects, some centralized and some distributed to functional organization. The following figure provides an organizational structure continuum and illustrates levels of governance among the functional organizations and the project.

- In a functional-structured organization, the project manager is a coordinator and typically has only limited control over the systems engineering functions. In this type of organization, the functional manager typically controls the project budget and has authority over the project resources. However, the organization may or may not have a functional unit for systems engineering. In the case where there is a functional unit for systems engineering, systems engineers are assigned across existing projects. Trades can be made among their projects to move the priority of a specific systems engineering project ahead of other projects; thus reducing the nominal schedule for that selected project. However, in the case where there is not a functional unit for systems engineering, the project manager may have to find alternate sources of staffing for systems engineering – for example, hiring systems engineering talent or consultants, promoting or expanding the responsibilities of a current team member, etc.
- In a project-structured organization, the project manager has full authority and responsibility for managing the budget and resources to meet the schedule requirements. The systems engineer is subject to the direction of the project manager. The project manager may work with human resources or a personnel manager or may go outside the organization to staff the project.
- Matrix-structured organization can have the advantages of both the functional and project structures. For a schedule driven project, function specialists are assigned to projects as needed to work for the project manager to apply their expertise on the project. Once they are no longer needed, they are returned to their functional groups (e.g. home office). In a weak matrix, the functional managers have authority to assign workers to projects and project managers must accept the workers assigned to them. In a strong matrix, the project manager controls the project budget and can reject workers from functional groups and hire outside workers if functional groups do not have sufficient available and trained workers.

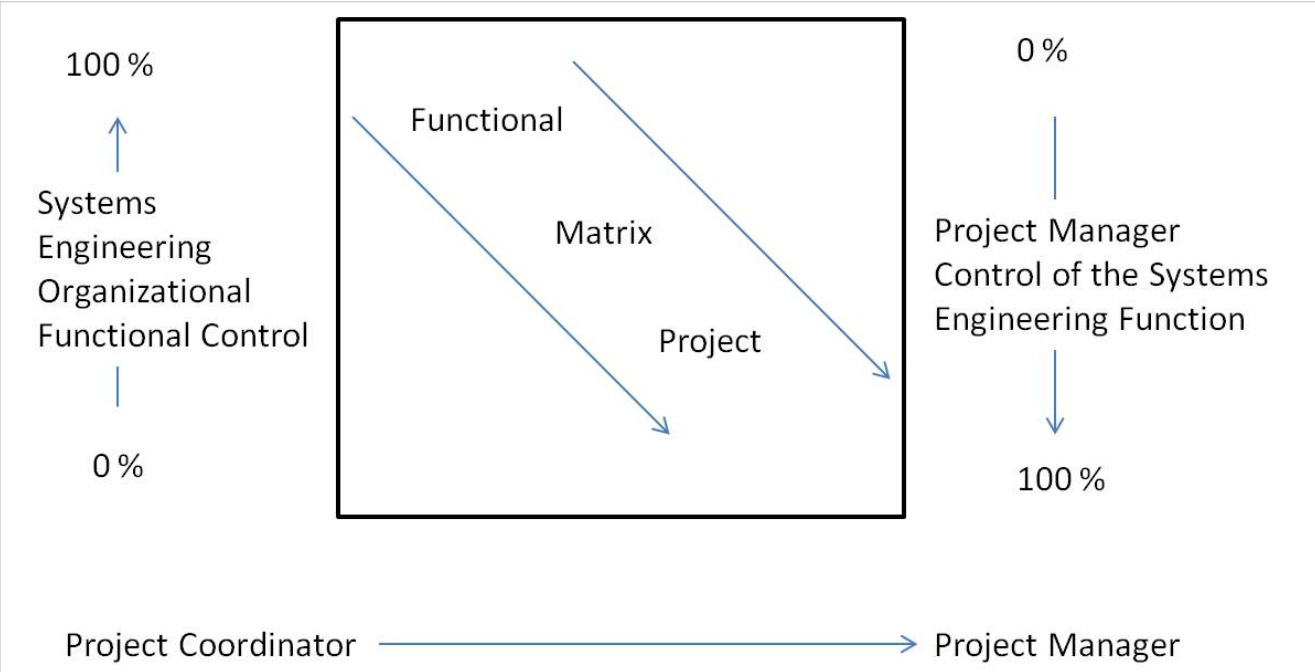


Figure 1. The Organizational Continuum (2). (SEBoK Original and Adapted from Fairley 2009). Reprinted with permission of the IEEE Computer Society. All other rights are reserved by the copyright owner.

In all cases, it is essential that the organizational and governance relationships be clarified and communicated to all project stakeholders and that the project manager and systems engineer work together in a collegial manner.

The Project Management Office (PMO) provides centralized control for a set of projects. The PMO is focused on meeting the business objectives leveraging a set of projects, while the project managers are focused on meeting the

objectives of those projects that fall under their purview. PMOs typically manage shared resources and coordinate communication across the projects, provide oversight and manage interdependencies, and drive project-related policies, standards, and processes. The PMO may also provide training and monitor compliance (PMI 2013).

Schedule-Driven versus Requirements-Driven Influences on Structure and Governance

This article addresses the influences on governance relationships between the project manager and the systems engineer. One factor that establishes this relationship is whether a project is schedule-driven or requirements-driven.

In general, a project manager is responsible for delivering an acceptable product/service on the specified delivery date and within the constraints of the specified schedule, budget, resources, and technology.

The systems engineer is responsible for collecting and defining the operational requirements, specifying the systems requirements, developing the system design, coordinating component development teams, integrating the system components as they become available, verifying that the system to be delivered is correct, complete and consistent to its technical specification, and validating the operation of the system in its intended environment.

From a governance perspective, the project manager is often thought of as being a movie producer who is responsible for balancing the schedule, budget, and resource constraints to meet customer satisfaction. The systems engineer is responsible for product content; ergo, the systems engineer is analogous to a movie director.

Organizational structures, discussed previously, provide the project manager and systems engineer with different levels of governance authority. In addition, schedule and requirements constraints can influence governance relationships. A schedule-driven project is one for which meeting the project schedule is more important than satisfying all of the project requirements; in these cases lower priority requirements may not be implemented in order to meet the schedule.

Classic examples of these types of projects are:

- a project that has an external customer with a contractual delivery date and an escalating late delivery penalty, and
- a project for which delivery of the system must meet a major milestone (e.g. a project for an announced product release of a cell phone that is driven by market considerations).

For schedule-driven projects, the project manager is responsible for planning and coordinating the work activities and resources for the project so that the team can accomplish the work in a coordinated manner to meet the schedule. The systems engineer works with the project manager to determine the technical approach that will meet the schedule. An Integrated Master Schedule (IMS) is often used to coordinate the project.

A requirements-driven project is one for which satisfaction of the requirements is more important than the schedule constraint. Classic examples of these types of projects are:

1. exploratory development of a new system that is needed to mitigate a potential threat (e.g. military research project) and
2. projects that must conform to government regulations in order for the delivered system to be safely operated (e.g., aviation and medical device regulations).

An Integrated Master Plan is often used to coordinate event-driven projects.

To satisfy the product requirements, the systems engineer is responsible for making technical decisions and making the appropriate technical trades. When the trade space includes cost, schedule, or resources, the systems engineer interacts with the project manager who is responsible for providing the resources and facilities needed to implement a system that satisfies the technical requirements.

Schedule-driven projects are more likely to have a management structure in which the project manager plays the central role, as depicted in Figure 1 of the Organizing the Team section in Team Capability. Requirement-driven projects are more likely to have a management structure in which the systems engineer plays the central role, as

depicted in Figure 2 of the Organizing the Team section in Team Capability.

Along with the Project Management Plan and the Systems Engineering Management Plan, IMP/IMS are critical to this process.

Related Structures

Integrated Product Teams (IPTs), Change Control Boards (CCBs), and Engineering Review Boards (ERBs) are primary examples of project structures that play a significant role in project governance and require coordination between the project manager, systems engineer and other members of the team.

Integrated Product Team

The Integrated Product Team (IPT) ensures open communication flow between the government and industry representatives as well as between the various product groups (see Good Practices in Planning). There is typically a top level IPT, sometimes referred to as the Systems Engineering and Integration Team (SEIT) (see Systems Engineering Organizational Strategy), that oversees the lower level IPTs. The SEIT can be led by either the project manager for a specific project or by the systems engineering functional manager or functional lead across many projects. Each IPT consists of representatives from the appropriate management and technical teams that need to collaborate on systems engineering, project management, and other activities to create a high-quality product. These representatives meet regularly to ensure that the technical requirements are understood and properly implemented in the design. Also see Team Capability for more information.

Change Control Board

An effective systems engineering approach includes a disciplined process for change control as part of the larger goal of configuration management. The primary objective of configuration management is to track changes to project artifacts that include software, hardware, plans, requirements, designs, tests, and documentation. Alternatively, a Change Control Board (CCB) with representatives from appropriate areas of the project is set up to effectively analyze, control and manage changes being proposed to the project. The CCB typically receives an Engineering Change Proposal (ECP) from design/development, production, or operations/support and initially reviews the change for feasibility. The ECP may also be an output of the Engineering Review Board (ERB) (see next section). If determined feasible, the CCB ensures there is an acceptable change implementation plan and proper modification and installation procedures to support production and operations.

There may be multiple CCBs in a large project. CCBs may be comprised of members from both the customer and the supplier. As with the IPTs, there can be multiple levels of CCB starting with a top level CCB with CCBs also existing at the subsystem levels. A technical lead typically chairs the CCB; however, the board includes representation from project management since the CCB decisions will have an impact on schedule, budget, and resources.

See Figure 2 under Configuration Management for a flow of the change control process adapted from Blanchard and Fabrycky (2011). See also Capability Updates, Upgrades, and Modernization, and topics included under Enabling Teams. See also the UK West Coast Modernization Project which provides an example where change control was an important success factor.

Engineering Review Board

Another example of a board that requires collaboration between technical and management is the Engineering Review Board (ERB). Examples of ERBs include the Management Safety Review Board (MSRB) (see Safety Engineering). Responsibilities of the ERB may include technical impact analysis of pending change requests (like the CCB), adjudication of results of engineering trade studies, and review of changes to the project baseline. In some cases, the ERB may be the management review board and the CCB may be the technical review board. Alternatively, in a requirement driven organization the ERB may have more influence while in a schedule driven organization the CCB may have more impact.

References

Works Cited

Blanchard, B.S., and W. J. Fabrycky. 2011. *Systems Engineering and Analysis.* 5th ed. Prentice-Hall International series in Industrial and Systems Engineering. Englewood Cliffs, NJ, USA: Prentice-Hall.

Fairley, R.E. 2009. *Managing and Leading Software Projects.* Hoboken, NJ, USA: IEEE Computer Society, John Wiley & Sons, Inc. Publication. ISBN: 978-0-470-29455-0.

PMI. 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 5th ed. Newtown Square, PA, USA: Project Management Institute (PMI).

Primary References

Forsberg, K., H. Mooz, and H. Cotterman. 2005. *Visualizing Project Management.* 3rd ed. New York, NY, USA: John Wiley & Sons.

Additional References

None.

Procurement and Acquisition

Lead Author: *Dick Fairley,* **Contributing Author:** Alice Squires

Procurement is the act of buying goods and services. Acquisition covers the conceptualization, initiation, design, development, testing, contracting, production, deployment, logistics support, modification, and disposal of weapons and other systems, as well as supplies or services (including construction) to satisfy organizational needs intended for use in, or in support of, defined missions (DAU 2010; DoD 2001).

Acquisition covers a much broader range of topics than procurement. Acquisition spans the whole life cycle of acquired systems. The procurement of appropriate systems engineering (SE) acquisition activities and levels of SE support is critical for an organization to meet the challenge of developing and maintaining complex systems.

The *Guide for Integrating Systems Engineering into DoD Acquisition Contracts* addresses how systems engineering activities are integrated into the various elements of acquisition and procurement (DoD 2006a).

Acquisition Process Model

Multiple acquisition process models exist. An acquisition process for major systems in industry and defense is shown in Figure 1. The process of acquisition is defined by a series of phases during which technology is defined and matured into viable concepts. These concepts are subsequently developed and readied for production, after which the systems produced are supported in the field.

Acquisition planning is the process of identifying and describing needs, capabilities, and requirements, as well as determining the best method for meeting those requirements (e.g., program acquisition strategy). This process includes procurement; thus, procurement is directly linked to the acquisition process model. The process model present in Figure 1 allows a given acquisition to enter the process at any of the development phases.

For example, a system using unproven technology would enter at the beginning stages of the process and would proceed through a lengthy period of technology maturation. On the other hand, a system based on mature and proven technologies might enter directly into engineering development or sometimes even production.

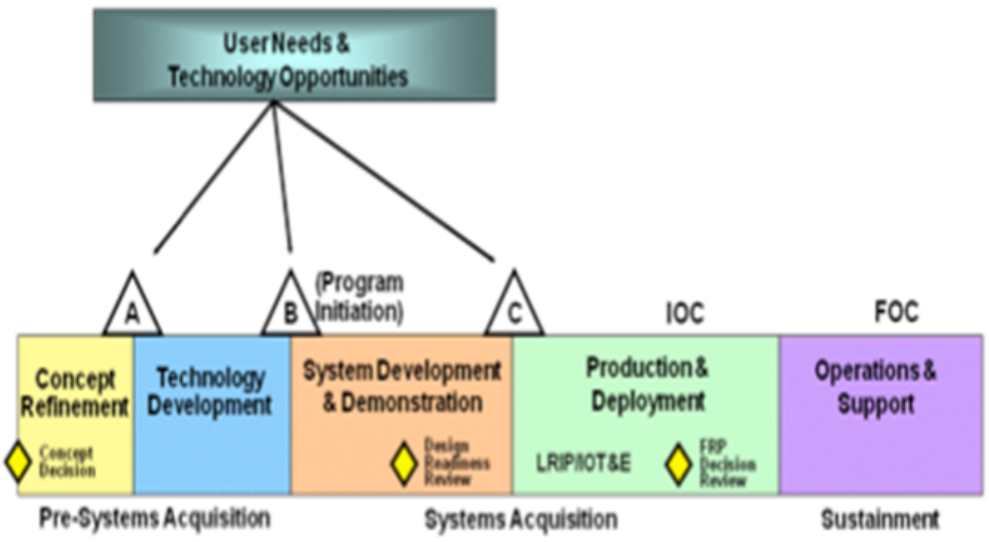


Figure 1. An Acquisition Process Model (DAU 2010). Released by Defense Acquisition University (DAU)/U.S. Department of Defense (DoD).

Systems Engineering Role in the Acquisition Process

The procurement of complex systems usually requires a close relationship between the offeror and supplier SE teams due to the breadth and depth of SE activities. SE is an overarching process that the program team applies in order to transition from a stated capability need to an affordable, operationally effective, and suitable system.

SE is important to every phase of the acquisition process. SE encompasses the application of SE processes across the acquisition life cycle and is intended to be an integrating mechanism for balanced solutions addressing capability needs, design considerations, and constraints. It is also intended to address limitations imposed by technology, budget, and schedule.

SE is an interdisciplinary approach; that is, it is a structured, disciplined, and documented technical effort to simultaneously design and develop system products and processes to satisfy the needs of the customer. Regardless of the scope and type of program, or at what point it enters the program acquisition life cycle, the technical approach to the program needs to be integrated with the acquisition strategy to obtain the best program solution.

Acquisition and procurement in the commercial sector have many characteristics in common with their counterparts in the realm of government contracting, although the processes in the commercial world are usually accomplished with fewer rigors than occur between government and contractor interactions. Offshore outsourcing is commonly practiced in the commercial software arena with the goal of reducing the cost of labor. Commercial organizations sometimes subcontract with other commercial organizations to provide missing expertise and to balance the ebb and flow of staffing needs.

In some cases, relations between the contracting organization and the subcontractor are strained because of the contracting organization's desire to protect its intellectual property and development practices from potential exposure to the subcontractor. Commercial organizations often have lists of approved vendors that are used to expedite the procurement of needed equipment, products, and services. In these situations, commercial organizations have processes to evaluate and approve vendors in ways that are analogous to the qualification of government contractors. Many commercial organizations apply SE principles and procedures even though they may not identify the personnel and job functions as "systems engineers" or "systems engineering."

Importance of the Acquisition Strategy in the Procurement Process

The acquisition strategy is usually developed during the front end of the acquisition life cycle. (For an example of this, see the Technology Development Phase in Figure 1.) The acquisition strategy provides the integrated strategy for all aspects of the acquisition program throughout the program life cycle.

In essence, the acquisition strategy is a high-level business and technical management approach designed to achieve program objectives within specified resource constraints. It acts as the framework for planning, organizing, staffing, controlling, and leading a program, as well as for establishing the appropriate contract mechanisms. It provides a master schedule for research, development, testing, production, fielding, and other SE related activities essential for program success, as well as for formulating functional strategies and plans.

The offeror's program team, including systems engineering, is responsible for developing and documenting the acquisition strategy, which conveys the program objectives, direction, and means of control based on the integration of strategic, technical, and resource concerns. A primary goal of the acquisition strategy is the development of a plan that will minimize the time and cost of satisfying an identified, validated need while remaining consistent with common sense and sound business practices. While the contract officer (CO) is responsible for all contracting aspects, including determining which type of contract is most appropriate, and following the requirements of existing regulations, directives, instructions, and policy memos of an organization, the program manager (PM) works with the CO to develop the best contract/procurement strategy and contract types.

Relating Acquisition to Request for Proposal and Technical Attributes

There are several formats for requesting proposals from offerors for building complex systems. Figure 2 relates acquisition program elements to a representative request for proposal (RFP) topical outline and key program technical attributes that have been used by the Department of Defense. In general, programs have a better chance of success when both the offeror and supplier understand the technical nature of the program and the need for the associated SE activities.

The offeror and supplier need to clearly communicate the technical aspect of the program throughout the procurement process. The offeror's RFP and the associated supplier proposal represent one of the formal communications paths. A partial list of key program technical attributes is presented in Figure 2.

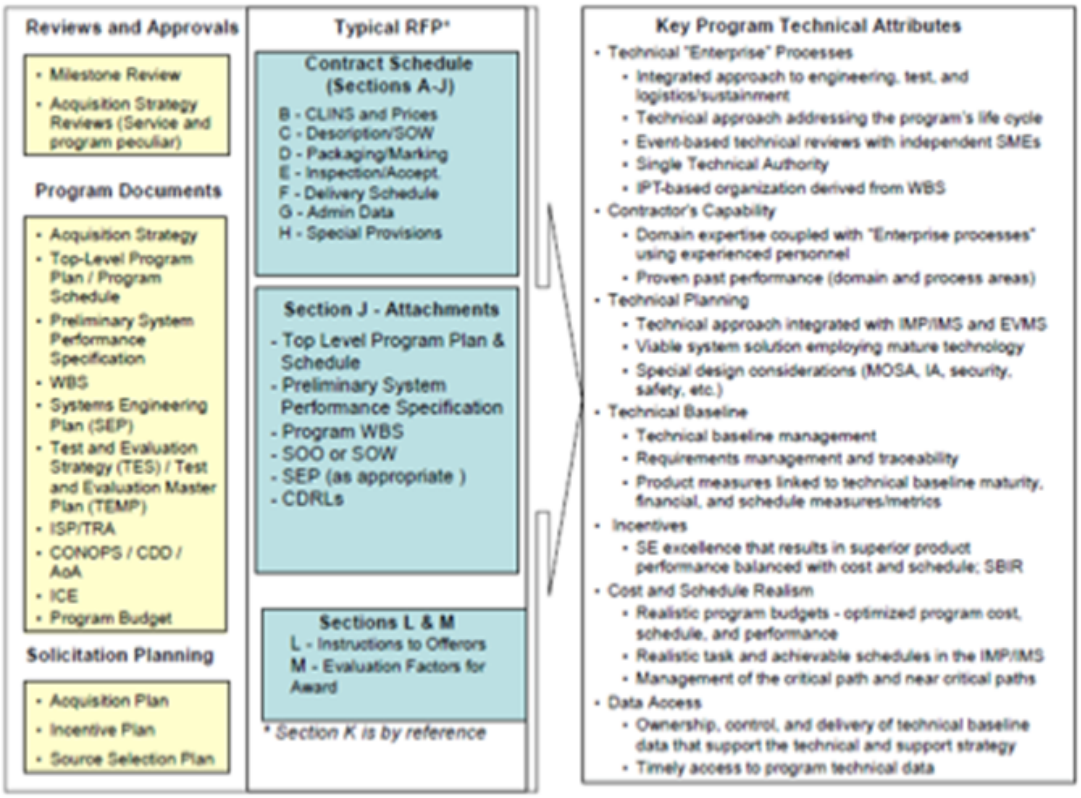


Figure 2. Relating Acquisition to Request for Proposal and Technical Attributes. (DoD 2006a). Released by the U.S. Office of the Secretary of Defense.

Contract-Related Activities and the Offeror's Systems Engineering and Project Management Roles

A clear understanding of the technical requirements is enhanced via the development of a Systems Engineering Plan (SEP). The SEP documents the systems engineering strategy for a project or program and acts as the blueprint for the conduct, management, and control of the technical aspects of the acquisition program (DoD 2011). The SEP documents the SE structure and addresses government and contractor boundaries. It summarizes the program's selected acquisition strategy and identifies and links to program risks. It also describes how the contractor's, and sometimes the subcontractor's and suppliers', technical efforts are to be managed.

Once the technical requirements are understood, a contract may be developed and followed by the solicitation of suppliers. The offeror's PM, chief or lead systems engineer, and CO must work together to translate the program's acquisition strategy and associated technical approach (usually defined in a SEP) into a cohesive, executable

contract(s).

Table 1 shows some key contracting-related tasks with indicators of the roles of the PM and LSE.

Table 1. Offeror's Systems Engineering and Program Management Roles (DoD 2006). Released by the U.S. Office of the Secretary of Defense.

Typical Contract-Related Activities	Systems Engineer and Project Manager Roles
1. Identify overall procurement requirements and associated budget. Describe the offer's needs and any constraints on the procurement.	1. Lead systems engineer (LSE) provides program technical requirements. PM provides any programmatic related requirements.
2. Identify technical actions required to successfully complete technical and procurement milestones. The program's SEP is the key source for capturing this technical planning.	2. LSE defines the technical strategy/approach and required technical efforts. This should be consistent with the program's Acquisition Strategy.
3. Document market research results and identify potential industry sources.	3. PM and LSE identify programmatic and technical information needed and assist in evaluating the results.
4. Prepare a Purchase Request, including product descriptions; priorities, allocations and allotments; architecture; government-furnished property or equipment (or Government-Off-The-Shelf (GOTS); government-furnished information; information assurance and security considerations; and required delivery schedules.	4. PM and LSE ensure the specific programmatic and technical needs are defined clearly (e.g., commercial-off-the-shelf (COTS) products).
5. Identify acquisition streamlining approach and requirements, budgeting and funding, management information requirements, environmental considerations, offeror's expected skill sets, and milestones. These should be addressed in the Acquisition Strategy.	5. The procurement team work together, but the CO has the prime responsibility. The PM is the owner of the program Acquisition Strategy. The LSE develops and reviews (and the PM approves) the technical strategy.
6. Plan the requirements for the contract Statement of Objectives (SOO) / Statement of Work (SOW) / specification, project technical reviews, acceptance requirements, and schedule.	6. LSE is responsible for the development of the technical aspects of the SOO/SOW.
7. Plan and conduct Industry Days as appropriate.	7. PM and LSE support the CO in planning the meeting agenda to ensure technical needs are discussed.
8. Establish contract cost, schedule, and performance reporting requirements. Determine an incentive strategy and appropriate mechanism (e.g., Award Fee Plan and criteria).	8. LSE provides technical resource estimates. LSE supports development of the Work Breakdown Structure (WBS) based on preliminary system specifications, determines event-driven criteria for key technical reviews, and determines what technical artifacts are baselined. The PM and LSE advise the CO in developing the metrics/criteria for an incentive mechanism.
9. Identify data requirements.	9. LSE identifies all technical Contractor Data Requirements List (CDRL) and technical performance expectations.
10. Establish warranty requirements, if applicable.	10. LSE works with the CO to determine cost-effective warranty requirements.
11. Prepare a Source Selection Plan (SSP) and RFP (for competitive contracts).	11. PM and LSE provide input to the SSP per the SOO/SOW.
12. Conduct source selection and award the contract to the successful offeror.	12. PM and LSE participate on evaluation teams.

Offeror and Supplier Interactions

There should be an environment of open communication prior to the formal source selection process. This ensures that the supplier understands the offeror's requirements and that the offeror understands the supplier's capabilities and limitations, as well as enhancing the supplier's involvement in the development of a program acquisition strategy. During the pre-solicitation phase, the offeror develops the solicitation and may ask suppliers to provide important insights into the technical challenges, program technical approach, and key business motivations.

For example, potential bidders could be asked for their assessment of a proposed system's performance based on the maturity level of new and existing technologies.

Contracts and Subcontracts

Typical types of contracts include the following:

- **Fixed Price**: In a fixed price contract the offeror proposes a single price for all products and services to implement the project. This single price is sometimes referred to as low bid or lump sum. A fixed price contract transfers the project risks to the supplier. When there is a cost overrun, the supplier absorbs it. If the supplier performs better than planned, their profit is higher. Since all risks are absorbed by the supplier, a fixed price bid may be higher to reflect this.
- **Cost-reimbursement** [Cost plus]: In a cost-reimbursement contract the offeror provides a fixed fee, but also reimburses the contractor for labor, material, overhead, and administration costs. Cost-reimbursement type contracts are used when there is a high level of project risk and uncertainty. With this type of contract, the risks reside primarily with the offeror. The supplier gets reimbursed for all of its costs. Additional costs that arise due to changes or rework are covered by the offeror. This type of contract is often recommended for the system definition of hardware and software development when there is a risk of stakeholder changes to the system.
- **Subcontracts**: A subcontractor performs work for another company as part of a larger project. A subcontractor is hired by a general contractor (also known as a prime or main contractor) to perform a specific set of tasks as part of the overall project. The incentive to hire subcontractors is either to reduce costs or to mitigate project risks. The systems engineering team is involved in establishing the technical contract requirements, technical selection criteria, acceptance requirements, and the technical monitoring and control processes.
- **Outsource contracts**: Outsourced contracts are used to obtain goods or services by contracting with an outside supplier. Outsourcing usually involves contracting a business function, such as software design and code development, to an external provider.
- **Exclusively Commercial Off-the-Shelf (COTS)**: Exclusively COTS contracts are completely satisfied with commercial solutions that require no modification for use. COTS solutions are used in the environment without modifying the COTS system. They are integrated into an existing user's platform or integrated into an existing operational environment. The systems engineering team is involved in establishing the technical contract requirements, technical acceptance, and technical selection criteria.
- **Integrated COTS**: Integrated COTS contracts use commercially available products and integrate them into existing user platforms or operational environments. In some cases, integrated COTS solutions modify the system's solution. The cost of integrating the commercial COTS product into the operational environment can exceed the cost of the COTS product itself. As a result, the systems engineering team is usually involved in establishing the technical outsourcing contract requirements, technical selection criteria, technical monitoring and control processes, and technical acceptance and integration processes.
- **COTS Modification**: COTS modification requires the most time and cost because of the additional work needed to modify the COTS product and integrate it into the system. Depending on how complex and critical the need is, the systems engineering team is usually involved in establishing the technical outsource contract requirements, technical selection criteria, technical monitoring and control processes, and technical acceptance requirements.

- **IT services**: IT services provide capabilities that can enable an enterprise, application, or Web service solution. IT services can be provided by an outsourced service provider. In many cases, the user interface for these Web services is as simple as a Web browser. Depending on how complex and critical the needs are, the systems engineering team can be involved in establishing the technical outsourcing contract requirements, technical selection criteria, and technical acceptance process.

References

Works Cited

DAU. 2010. *Defense Acquisition Guidebook (DAG).* Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/US Department of Defense (DoD).

DoD. 2011. *Systems Engineering Plan (SEP) Outline.* Washington, DC, USA: Office of the Undersecretary of Defense for Acquisition, Transportation, and Logistics (AT&L), US Department of Defense (DoD).

DoD. 2006. *Guide for Integrating Systems Engineering into DoD Acquisition Contracts.* Washington, DC, USA: Office of the Undersecretary of Defense for Acquisition, Transportation, and Logistics (AT&L), US Department of Defense (DoD).

DoD. 2001. *Systems Engineering Fundamentals.* Washington, DC, USA: Defense Acquisition University Press/US Department of Defense.

Primary References

DAU. 2010. *Defense Acquisition Guidebook (DAG).* Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/US Department of Defense (DoD).

DoD. 2011. *Systems Engineering Plan (SEP) Outline.* Washington, DC, USA: Office of the Undersecretary of Defense for Acquisition, Transportation, and Logistics (AT&L), US Department of Defense (DoD).

DoD. 2006. *Guide for Integrating Systems Engineering into DoD Acquisition Contracts.* Washington, DC, USA: Office of the Undersecretary of Defense for Acquisition, Transportation, and Logistics (AT&L), US Department of Defense (DoD).

Additional References

MITRE. 2011. "Acquisition Systems Engineering." *Systems Engineering Guide.* Accessed March 9, 2012. Available: http://www.mitre.org/work/systems_engineering/guide/acquisition_systems_engineering/.

Knowledge Area: Systems Engineering and Industrial Engineering

Systems Engineering and Industrial Engineering

Lead Author: *Johann Demmel*, ***Contributing Author:*** *Hillary Sillitto*

> *Industrial Engineering is concerned with the design, improvement and installation of integrated systems of people, materials, information, equipment and energy. It draws upon specialized knowledge and skill in the mathematical, physical, and social sciences together with the principles and methods of engineering analysis and design, to specify, predict, and evaluate the results to be obtained from such systems.* (IIE 1992)

Industrial engineering (IE) encompasses several aspects of systems engineering (SE) (i.e., production planning and analysis, continuous process improvement, etc.) and also many elements of the engineered systems domain (production control, supply chain management, operations planning and preparation, operations management, etc.), as depicted in Figure 3 of the article Scope and Context of the SEBoK.

This knowledge area covers the overarching aspects of industrial engineering and describes the synergies between IE and SE.

Overview of Industrial Engineering

Industrial engineers are trained to design and analyze the components of which man-machine systems are composed. They bring together individual elements that are designed via other engineering disciplines and properly synergize these subsystems together with the people components for a completely integrated man-machine system. Industrial engineers are focused on the improvement of any system that is being designed or evaluated. They make individual human tasks more productive and efficient by optimizing flow, eliminating unnecessary motions, utilizing alternate materials to improve manufacturing, improving the flow of product through processes, and optimizing the configuration of workspaces. Fundamentally, the industrial engineer is charged with reducing costs and increasing profitability through ensuring the efficient use of human, material, physical, and/or financial resources (Salvendy 2001).

A systems engineer leverages industrial engineering knowledge to provide:

- production planning and analysis
- systems integration
- lifecycle planning and estimating
- change analysis and management
- continuous process improvement
- quality assurance
- business case analysis / return on investment
- engineering management
- systems integration

Industrial engineers complement systems engineers with knowledge in:

- supply chain management

- budgeting and economic analysis
- production line preparation
- production
- production control
- testing
- staffing, organizing, directing
- cost, schedule, and performance monitoring
- risk monitoring and control
- operations planning and preparation
- operations management

Industrial Engineering Body of Knowledge

The current overview of the industrial engineering body of knowledge is provided in the *Handbook of Industrial Engineering* (Salvendy 2001) and *Maynard's Industrial Engineering Handbook* (Zandin 2001). The Institute of Industrial Engineers (IIE 1992) is currently in the process of developing a specific industrial engineering body of knowledge. Additionally, industrial engineering terminology defines specific terms related to the industrial engineering profession. Definitions used in this section are from this reference. Turner et al. (1992) provide an overview of industrial and systems engineering.

The elements of IE include the following:

Operations Engineering

Operations engineering involves the management and control aspects of IE and works to ensure that all the necessary requirements are in place to effectively execute a business. Key areas of knowledge in this field include: product and process life cycles, forecasting, project scheduling, production scheduling, inventory management, capacity management, supply chain, distribution, and logistics. Concepts such as materials requirements planning and enterprise resource planning find their roots in this domain.

Operations Research

Operations research is the organized and systematic analysis of complex situations, such as if there is a spike in the activities of organizations of people and resources. The analysis makes use of certain specific disciplinary methods, such as probability, statistics, mathematical programming, and queuing theory. The purpose of operations research is to provide a more complete and explicit understanding of complex situations, to promote optimal performance utilizing the all the resources available. Models are developed that describe deterministic and probabilistic systems and these models are employed to aid the decision maker. Knowledge areas in operations research include linear programming, network optimization, dynamic programming, integer programming, nonlinear programming, metaheuristics, decision analysis and game theory, queuing systems, and simulation. Classic applications include the transportation problem and the assignment problem.

Production Engineering / Work Design

Production engineering is the design of a production or manufacturing process for the efficient and effective creation of a product. Included in this knowledge area is classic tool and fixture design, selection of machines to produce product, and machine design. Closely related to production engineering, work design involves such activities as process, procedural and work area design, which are geared toward supporting the efficient creation of goods and services. Knowledge in work simplification and work measurement are crucial to work design. These elements form a key foundation, along with other knowledge areas in IE, for lean principles.

Facilities Engineering and Energy Management

Facilities engineering involves attempting to achieve the optimal organization in factories, buildings, and offices. In addition to addressing the aspects of the layout inside a facility, individuals in this field also possess knowledge of material and equipment handling as well as storage and warehousing. This area also involves the optimal placement and sizing of facilities according to the activities they are required to contain. An understanding of code compliance and use of standards is incorporated. The energy management aspect of this area encompasses atmospheric systems and lighting and electrical systems. Through the development of responsible management of resources in the energy management domain, industrial engineers have established a basis in sustainability.

Ergonomics

Ergonomics is the application of knowledge in the life sciences, physical sciences, social sciences, and engineering that studies the interactions between the human and the total working environment, such as atmosphere, heat, light and sound, as well as the interactions of all tools and equipment in the workplace. Ergonomics is sometimes referred to as *human factors*. Individuals in this field have a specialized knowledge in areas such as: anthropometric principles, standing/sitting, repetitive task analysis, work capacity and fatigue, vision and lighting, hearing, sound, noise, vibration, human information processing, displays and controls, and human-machine interaction. Members in this field also consider the organizational and social aspects of a project.

Engineering Economic Analysis

Engineering economic analysis concerns techniques and methods that estimate output and evaluate the worth of commodities and services relative to their costs. Engineering economic analysis is used to evaluate system affordability. Fundamental to this knowledge area are value and utility, classification of cost, time value of money and depreciation. These are used to perform cash flow analysis, financial decision making, replacement analysis, break-even and minimum cost analysis, accounting and cost accounting. Additionally, this area involves decision making involving risk and uncertainty and estimating economic elements. Economic analysis also addresses any tax implications.

Quality and Reliability

Quality is the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs. Reliability is the ability of an item to perform a required function under stated conditions for a stated period of time. The understanding of probability and statistics form a key foundation to these concepts. Knowledge areas in quality and reliability include: quality concepts, control charts, lot acceptance sampling, rectifying inspection and auditing, design of experiments, and maintainability. Six sigma has its roots in the quality domain; however, its applicability has grown to encompass a total business management strategy.

Engineering Management

Engineering management refers to the systematic organization, allocation, and application of economic and human resources in conjunction with engineering and business practices. Knowledge areas include: organization, people, teamwork, customer focus, shared knowledge systems, business processes, resource responsibility, and external influences.

Supply Chain Management

Supply chain management deals with the management of the input of goods and services from outside sources that are required for a business to produce its own goods and services. Information is also included as a form of input. Knowledge areas include: building competitive operations, planning and logistics, managing customer and supplier relationships, and leveraging information technology to enable the supply chain.

References

Works Cited

IIE. 1992. *Industrial Engineering Terminology*, revised edition. Norwood, GA, USA: Institute of Industrial Engineers (IIE). Accessed March 7, 2012. Available: http://www.iienet2.org/Details.aspx?id=645.

Salvendy, G. (ed.) 2001. *Handbook of Industrial Engineering, Technology and Operations Management,* 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc.

Turner, W.C., J.H. Mize, K.E. Case, and J.W. Nazemtz. 1992. *Introduction to Industrial and Systems Engineering*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall.

Zandin, K.B. (ed.) 2001. *Maynard's Industrial Engineering Handbook*, 5th ed. New York, NY, USA: McGraw-Hill.

Primary References

IIE. 1992. *Industrial Engineering Terminology*, revised edition. Norwood, GA, USA: Institute of Industrial Engineers (IIE). Accessed March 7, 2012. Available: http://www.iienet2.org/Details.aspx?id=645.

Salvendy, G. (ed.) 2001. *Handbook of Industrial Engineering, Technology and Operations Management,* 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc.

Zandin, K.B. (ed.) 2001. *Maynard's Industrial Engineering Handbook,* 5th ed. New York, NY, USA: McGraw-Hill.

Additional References

Operations Engineering

Hopp, W., and M. Spearman. 2001. *Factory Physics,* 3rd ed., New York, NY, USA: McGraw-Hill.

Heizer, J., and B. Render. 2001. *Operations Management,* 6th ed. Upper Saddle River, NJ, USA: Prentice Hall.

Mantel, S., J. Meredith, S. Shafer, and M. Sutton. 2008. *Project Management in Practice.* New York, NY, USA: John Wiley & Sons.

Operations Research

Banks, J., J. Carson, B. Nelson, and D. Nicol. 2005. *Discrete-Event System Simulation,* 4th ed. Upper Saddle River, NJ, USA: Prentice Hall.

Hillier, F., and G. Lieberman. 2010. *Introduction to Operations Research,* 9th ed. New York, NY, USA: McGraw Hill.

Kelton, W. David, R. Sadowski, and D. Sturrock. 2006. *Simulation with Arena,* 4th ed. New York, NY, USA: McGraw-Hill.

Law, A. 2007. *Simulation Modelling and Analysis,* 4th ed. New York, NY, USA: McGraw-Hill.

Winston, W. and J. Goldberg. 2004. *Operations Research Applications & Algorithms,* Independence, KY, USA: Thomson Brooks/Cole.

Production Engineering / Work Design

Freivalds, A. 2009. *Niebel's Methods, Standards, and Work Design,* 12th ed. New York, NY, USA: McGraw-Hill.

Groover, M. 2007. *Work Systems: The Methods, Measurement, and Management of Work,* Upper Saddle River, NJ, USA: Pearson-Prentice Hall.

Grover, M. 2007. *Fundamentals of Modern Manufacturing,* 3rd ed. New York, NY, USA: John Wiley & Sons.

Konz, S., and S. Johnson. 2008. *Work Design: Occupational Ergonomics,* 7th ed. Scottsdale, AZ, USA: Holcomb Hathaway.

Meyers, F., and J. Stewart. 2001. *Motion and Time Study for Lean Manufacturing,* 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall.

Facilities Engineering and Energy Management

Garcia-Diaz, A., and J. MacGregor Smith. 2008. *Facilities Planning and Design,* Upper Saddle River, NJ, USA: Pearson-Prentice Hall.

Tompkins, J., J. White, Y. Bozer, and J. Tanchoco. 2003. *Facilities Planning,* 3rd ed. New York, NY, USA: John Wiley & Sons.

Ergonomics

Chaffin, D., and G. Andersson. 1991. *Occupational Biomechanics.* New York, NY, USA: John Wiley & Sons.

Wickens, C., S. Gordon, and Y. Liu. 2004. *An Introduction to Human factors Engineering.* Upper Saddle River, NJ, USA: Pearson-Prentice Hall.

Engineering Economic Analysis

Blank, L.T., and A.J. Tarquin. 2011. *Engineering Economy,* 7th ed. New York, NY, USA: McGraw-Hill.

Newnan, D., T. Eschenbach, and J. Lavelle. 2011. *Engineering Economic Analysis,* 11th ed. New York, NY, USA: Oxford University Press.

Parl, C. 2007. *Fundamentals of Engineering Economics.* Upper Saddle River, NJ, USA: Prentice Hall.

Thuesen, G., and W. Fabrycky. 2001. *Engineering Economy,* 9th ed. Upper Saddle River, NJ, USA: Prentice Hall.

Quality & Reliability

Ebeling, C.E. 2005. *An Introduction to Reliability and Maintainability Engineering.* Long Grove, IL, USA: Waveland Press, Inc.

Hawkins, D., and D. Olwell. 1998. *Cumulative Sum Chars and Charting for Quality Improvement.* New York, NY, USA: Springer.

Kiemele, M., S. Schmidt, and R. Berdine. 1999. *Basic Statistics: Tools for Continuous Improvement*, 4th ed. Colorado Springs, CO, USA: Air Academy Press.

Montgomery, D., and G. Runger. 2007. *Applied Statistics and Probability for Engineers*, 4th ed. Hoboken, NJ, USA: John Wiley & Sons.

Montgomery, D. 2013. *Design & Analysis of Experiments,* 8th ed. Hoboken, NJ, USA: John Wiley & Sons.

Montgomery, D. 2009. *Introduction to Statistical Quality Control*, 6th ed. Hoboken, NJ, USA: John Wiley & Sons.

Quality Staff. 2006. *Data Quality Assessment: Statistical Methods for Practitioners.* Washington, D.C., USA: Environmental Protection Agency (EPA).

Engineering Management

Gido, J., and J. Clements. 2009. *Successful Project Management.* Cincinnati, OH, USA: South Western.

Kersner, H. 2009. *A Systems Approach to Planning, Scheduling, and Controlling,* 10th ed. New York, NY, USA: John Wiley & Sons.

Supply Chain Management

Jacobs, F., and R. Chase. 2010. *Operations and Supply Chain Management.* New York, NY, USA: McGraw-Hill.

Mentzer, J. 2004. *Fundamentals of Supply Chain Management: Twelve Drivers of Competitive Advantage.* Thousand Oaks, CA, USA: Sage.

< Previous Article | Parent Article | Next Article >

Systems Engineering and Specialty Engineering

Systems Engineering and Specialty Engineering

Lead Author: *Dave Olwell*, ***Contributing Authors:*** *Richard Turner, Dick Fairley, Scott Jackson, Alice Squires*

Every system has a set of properties that are largely a function of the system as a whole rather than just its constituent parts; e.g., security, reliability, cost, and resilience. This KA describes the engineering disciplines that enable the realization of those properties. Collectively, these disciplines are referred to as specialty engineering. Some consider specialty engineering to be a subdiscipline of SE itself; e.g. they consider security engineering, safety engineering, resilience engineering, etc. to be part of SE. Others consider them to be stand-alone disciplines in their own right. Either way, their mastery is important to a system engineer. Moreover, they are usually interdependent; e.g. increasing system reliability often requires using more expensive parts and adding redundant components. Hence, higher reliability often means higher cost to deliver a system – another system property. However, higher reliability might mean lower maintenance cost – another system property. A systems engineer typically makes many decisions and takes many actions with regard to system properties; e.g., specifying which properties are important for a particular system, stating how those properties will be measured, trading off conflicting properties, and verifying that a system has the specified properties. This version of the SEBoK includes topical articles on several specialty engineering disciplines such as Reliability, Availability, and Maintainability. A major revision of this KA is planned for an upcoming release which will provide a new comprehensive framework for describing and relating system properties as well as providing updates to the related articles.

Topics

Each part of the SEBoK is divided into knowledge areas (KAs), which are groupings of information with a related theme. The Kas, in turn, are divided into topics. This KA contains the following topics:

- Reliability, Availability, and Maintainability
- Human Systems Integration
- Security Engineering
- Electromagnetic Interference/Electromagnetic Compatibility
- System Resilience
- Manufacturability and Producibility
- Affordability
- Environmental Engineering
- Logistics Engineering

Specialty Requirements

The systems engineering team must ensure that specialty requirements are properly reviewed with regard to their impact on life cycle costs, development schedule, technical performance, and operational utility. For example, security requirements can impact operator workstations, electromagnetic interference requirements can impact the signal in the interfaces between subsystems, and mass-volume requirements may preclude the use of certain materials to reduce subsystem weight.

Engineering specialists audit the evolving design and resulting configuration items to ensure that the overall system performance also satisfies the specialty requirements. Including appropriate specialty engineers within each systems engineering team assures that all system requirements are identified and balanced throughout the development cycle.

Integration of Specialty Engineering

Integration of engineering specialties into a project or program is, or should be, a major objective of systems engineering management. With properly implemented procedures, the rigor of the systems engineering process ensures participation of the specialty disciplines at key points in the technical decision-making process. Special emphasis on integration is mandatory because a given design could in fact be accomplished without consideration of these “specialty” disciplines, leading to the possibility of system ineffectiveness or failure when an unexamined situation occurs in the operational environment.

For example, human factors considerations can contribute to reduced workloads and therefore lower error rates by operators in aircraft cockpits, at air-traffic consoles, or nuclear reactor stations. Similarly, mean-time-to-repair features can significantly increase overall system availability in challenging physical environments, such as mid-ocean or outer space. Specialty engineering requirements are often manifest as constraints on the overall system design space. The role of system engineering is to balance these constraints with other functionality in order to harmonize total system performance. The end goal is to produce a system that provides utility and effectiveness to the customer at an affordable price.

As depicted in Figure 1, systems engineering plays a leadership role in integrating traditional disciplines, specialty disciplines, and unique system product demands to define the system design. Relationships for this integration process are represented as interactions among three filters.

The first filter is a conceptual analysis that leverages traditional design consideration (structural, electronics, aerodynamics, mechanical, thermodynamics, etc.). The second filter evaluates the conceptual approach using specialty disciplines, such as safety, affordability, quality assurance, human factors, reliability and maintainability, producibility, packaging, test, logistics, and others, to further requirements development. Design alternatives that pass through these two processes go through a third filter that incorporates facility design, equipment design, procedural data, computer programs, and personnel to develop the final requirements for design selection and further detailed development.

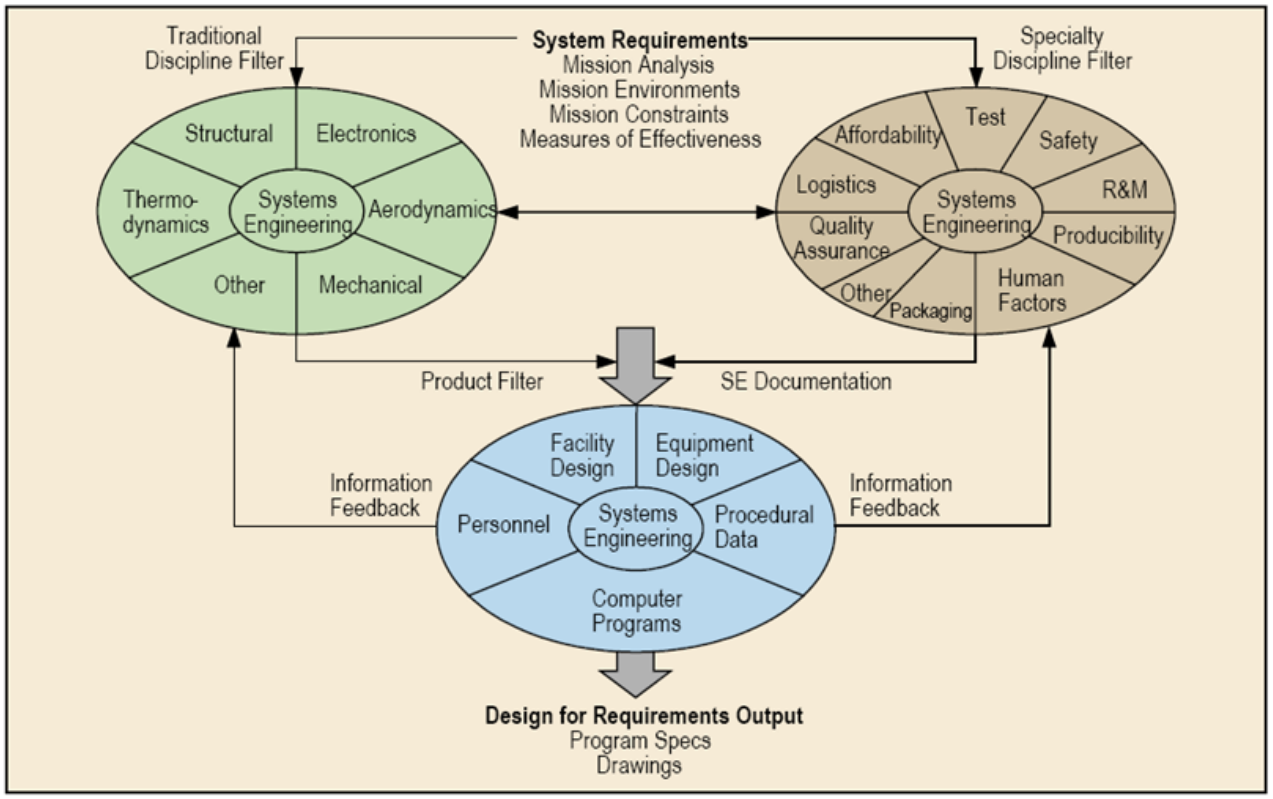


Figure 1. Integration Process for Specialty Engineering (USAF 2000). Released by the U.S. Air Force.

References

Works Cited

USAF. 2000. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems*, version 3.0. Hill AFB: Department of the Air Force Software Technology Support Center. May 2000. Accessed on September 11, 2011. Available at http://www.stsc.hill.af.mil/resources/tech%5Fdocs/.

Primary References

USAF. 2000. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems*, version 3.0. Hill AFB: Department of the Air Force Software Technology Support Center. May 2000. Accessed on September 11, 2011. Available at http://www.stsc.hill.af.mil/resources/tech%5Fdocs/.

Additional References

None.

Reliability, Availability, and Maintainability

Lead Authors: *Paul Phister, David Olwell*

Reliability, availability, and maintainability (RAM) are three system attributes that are of tremendous interest to systems engineers, logisticians, and users. Collectively, they affect economic life-cycle costs of a system and its utility.

Overview

Reliability, maintainability, and availability (RAM) are three system attributes that are of great interest to systems engineers, logisticians, and users. Collectively, they affect both the utility and the life-cycle costs of a product or system. The origins of contemporary reliability engineering can be traced to World War II. The discipline's first concerns were electronic and mechanical components (Ebeling, 2010). However, current trends point to a dramatic rise in the number of industrial, military, and consumer products with integrated computing functions. Because of the rapidly increasing integration of computers into products and systems used by consumers, industry, governments, and the military, reliability must consider both hardware, and software.

Maintainability models present some interesting challenges. The time to repair an item is the sum of the time required for evacuation, diagnosis, assembly of resources (parts, bays, tool, and mechanics), repair, inspection, and return. Administrative delay (such as holidays) can also affect repair times. Often these sub-processes have a minimum time to complete that is not zero, resulting in the distribution used to model maintainability having a threshold parameter.

A threshold parameter is defined as the minimum probable time to repair. Estimation of maintainability can be further complicated by queuing effects, resulting in times to repair that are not independent. This dependency frequently makes analytical solution of problems involving maintainability intractable and promotes the use of simulation to support analysis.

System Description

This section sets forth basic definitions, briefly describes probability distributions, and then discusses the role of RAM engineering during system development and operation. The final subsection lists the more common reliability test methods that span development and operation.

Basic Definitions

Reliability

Defined as the probability of a system or system element performing its intended function under stated conditions without failure for a given period of time (ASQ 2011). A precise definition must include a detailed description of the function, the environment, the time scale, and what constitutes a failure. Each can be surprisingly difficult to define as precisely as one might wish.

Maintainability

Defined as the probability that a system or system element can be repaired in a defined environment within a specified period of time. Increased maintainability implies shorter repair times (ASQ 2011).

Availability

Defined as the probability that a repairable system or system element is operational at a given point in time under a given set of environmental conditions. Availability depends on reliability and maintainability and is discussed in detail later in this topic (ASQ 2011).

A failure is the event(s), or inoperable state, in which any item or part of an item does not, or would not, perform as specified (GEIA 2008). The failure mechanism is the physical, chemical, electrical, thermal, or other process that results in failure (GEIA 2008). In computerized systems, a software defect or fault can be the cause of a failure (Laprie 1992) which may have been preceded by an error which was internal to the item. The failure mode is the way or the consequence of the mechanism through which an item fails (GEIA 2008, Laprie 1992.). The severity of the failure mode is the magnitude of its impact (Laprie, 1992).

Probability Distributions used in Reliability Analysis

Reliability can be thought of as the probability of the survival of a component until time *t*. Its complement is the probability of failure before or at time *t*. If we define a random variable *T* as the time to failure, then:

where *R(t)* is the reliability and *F(t)* is the failure probability. The failure probability is the cumulative distribution function (CDF) of a mathematical probability distribution. Continuous distributions used for this purpose include exponential, Weibull, log-normal, and generalized gamma. Discrete distributions such as the Bernoulli, Binomial, and Poisson are used for calculating the expected number of failures or for single probabilities of success.

The same continuous distributions used for reliability can also be used for maintainability although the interpretation is different (i.e., probability that a failed component is restored to service prior to time *t*). However, predictions of maintainability may have to account for processes such as administrative delays, travel time, sparing, and staffing and can therefore be extremely complex.

The probability distributions used in reliability and maintainability estimation are referred to as models because they only provide estimates of the true failure and restoration of the items under evaluation. Ideally, the values of the parameters used in these models would be estimated from life testing or operating experience. However, performing such tests or collecting credible operating data once items are fielded can be costly. Therefore, approximations sometimes use data from “similar systems”, “engineering judgment”, and other methods. As a result, those estimates based on limited data may be very imprecise. Testing methods to gather such data are discussed below.

RAM Considerations during Systems Development

RAM are inherent product or system attributes that should be considered throughout the development lifecycle. Reliability standards, textbook authors, and others have proposed multiple development process models (O’Connor 2014, Kapur 2014, Ebeling 2010, DoD 2005). The discussion in this section relies on a standard developed by a joint effort by the Electronic Industry Association and the U.S. Government and adopted by the U.S. Department of Defense (GEIA 2008) that defines 4 processes: understanding user requirements and constraints, design for reliability, production for reliability, and monitoring during operation and use (discussed in the next section).

Understanding User Requirements and Constraints

Understanding user requirements involves eliciting information about functional requirements, constraints (e.g., mass, power consumption, spatial footprint, life cycle cost), and needs that correspondent to RAM requirements. From these emerge system requirements that should include specifications for reliability, maintainability, and availability, and each should be conditioned on the projected operating environments. RAM requirements definition is as challenging but as essential to development success as the definition of general functional requirements.

Design for Reliability

System designs based on user requirements and system design alternatives can then be formulated and evaluated. Reliability engineering during this phase seeks to increase system robustness through measures such as redundancy, diversity, built-in testing, advanced diagnostics, and modularity to enable rapid physical replacement. In addition, it may be possible to reduce failure rates through measures such as use of higher strength materials, increasing the quality components, moderating extreme environmental conditions, or shortened maintenance, inspection, or overhaul intervals. Design analyses may include mechanical stress, corrosion, and radiation analyses for mechanical components, thermal analyses for mechanical and electrical components, and Electromagnetic Interference (EMI) analyses or measurements for electrical components and subsystems.

In most computer-based systems, hardware mean time between failures are hundreds of thousands of hours so that most system design measures to increase system reliability are focused on software. The most obvious way to improve software reliability is by improving its quality through more disciplined development efforts and tests. Methods for doing so are in the scope of software engineering but not in the scope of this section. However, reliability and availability can also be increased through architectural redundancy, independence, and diversity. Redundancy must be accompanied by measures to ensure data consistency, and managed failure detection and switchover. Within the software architecture, measures such as watchdog timers, flow control, data integrity checks (e.g., hashing or cyclic redundancy checks), input and output validity checking, retries, and restarts can increase reliability and failure detection coverage (Shooman, 2002).

System RAM characteristics should be continuously evaluated as the design progresses. Where failure rates are not known (as is often the case for unique or custom developed components, assemblies, or software), developmental testing may be undertaken to assess the reliability of custom-developed components. Evaluations based on quantitative analyses assess the numerical reliability and availability of the system and are usually based on reliability block diagrams, fault trees, Markov models, and Petri nets (O'Connor, 2011). Markov models and Petri nets are of particular value for computer-based systems that use redundancy. Evaluations based on qualitative analyses assess vulnerability to single points of failure, failure containment, recovery, and maintainability. The primary qualitative methods are the failure mode effects and criticality analyses (FMECA) (Kececioglu 1991). The development program Discrepancy Reporting (DR) or Failure Reporting and Corrective Action System (FRACAS) should also be used to identify failure modes which may not have been anticipated by the FMECA and to identify common problems that can be corrected through an improved design or development process.

Analyses from related disciplines during design time also affect RAM. Human factor analyses are necessary to ensure that operators and maintainers can interact with the system in a manner that minimizes failures and the restoration times when they occur. There is also a strong link between RAM and cybersecurity in computer-based systems. On the one hand, defensive measures reduce the frequency of failures due to malicious events. On the other hand, devices such as firewalls, policy enforcement devices, and access/authentication serves (also known as "directory servers") can also become single points of failure or performance bottlenecks that reduce system reliability and availability.

Production for Reliability

Many production issues associated with RAM are related to quality. The most important of these are ensuring repeatability and uniformity of production processes and complete unambiguous specifications for items from the supply chain. Other are related to design for manufacturability, storage, and transportation (Kapur 2014; Eberlin 2010). Large software intensive information systems are affected by issues related to configuration management, integration testing, and installation testing. Testing and recording of failures in the problem reporting and corrective action systems (PRACAS) or the FRACAS capture data on failures and improvements to correct failures. Depending on organizational considerations, this may be the same or a separate system as used during the design.

Monitoring During Operation and Use

After systems are fielded, their reliability and availability are monitored to assess whether the system or product has met its RAM objectives, identify unexpected failure modes, record fixes, and assess the utilization of maintenance resources and the operating environment. The FRACAS or a maintenance management database may be used for this purpose. In order to assess RAM, it is necessary to maintain an accurate record not only of failures but also of operating time and the duration of outages. Systems that report only on repair actions and outage incidents may not be sufficient for this purpose.

An organization should have an integrated data system that allows reliability data to be considered with logistical data, such as parts, personnel, tools, bays, transportation and evacuation, queues, and costs, allowing a total awareness of the interplay of logistical and RAM issues. These issues in turn must be integrated with management and operational systems to allow the organization to reap the benefits that can occur from complete situational awareness with respect to RAM.

Reliability and Maintainability Testing

Reliability Testing can be performed at the component, subsystem, and system level throughout the product or system lifecycle. Examples of hardware related categories of reliability testing are detailed in (Ebeling, 2010, O'Connor 2014).

- **Reliability Life Tests:** Reliability life tests are used to empirically assess the time to failure for non-repairable products and systems and the times between failure for repairable or restorable systems. Termination criteria for such tests can be based on a planned duration or planned number of failures. Methods to account for "censoring" of the failures or the surviving units enable a more accurate estimate of reliability.
- **Accelerated Life Tests:** Accelerated life testing is performed by subjecting the items under test (usually electronic parts) to increased temperatures well above the expecting operating temperature and extrapolating results using an Arhenius relation.
- **Highly Accelerated Life Testing/Highly Accelerated Stress Testing (HALT/HASS):** is performed by subjecting units under test (components or subassemblies) to extreme temperature and vibration tests with the objective of identifying failure modes, margins, and design weaknesses.
- **Parts Screening:** Parts screening is not really a test but a procedure to operate components for a duration beyond the "infant mortality" period during which less durable items fail and the more durable parts that remain are then assembled into the final product or system.
- **System Level Testing:** Examples of system level testing (including both hardware and software) are detailed in (O'Connor 2014, Ebeling 2010).
- **Stability Tests:** Stability tests are life tests for integrated hardware and software systems. The goal of such testing is to determine the integrated system failure rate and assess operational suitability. Test conditions must include accurate simulation of the operating environment (including workload) and a means of identifying and recording failures.

- **Reliability Growth Tests:** Reliability growth testing is part of a reliability growth program in which items are tested throughout the development and early production cycle with the intent of assessing reliability increases due to improvements in the manufacturing process (for hardware) or software quality (for software).
- **Failure/Recovery Tests:** Such testing assesses the fault tolerance of a system by measuring probability of switchover for redundant systems. Failures are simulated and the ability of the hardware and software to detect the condition and reconfigure the system to remain operational are tested.
- **Maintainability Tests:** Such testing assesses the system diagnostics capabilities, physical accessibility, and maintainer training by simulating hardware or software failures that require maintainer action for restoration.

Because of its potential impact on cost and schedule, reliability testing should be coordinated with the overall system engineering effort. Test planning considerations include the number of test units, duration of the tests, environmental conditions, and the means of detecting failures.

Data Issues

True RAM models for a system are generally never known. Data on a given system is assumed or collected, used to select a distribution for a model, and then used to fit the parameters of the distribution. This process differs significantly from the one usually taught in an introductory statistics course.

First, the normal distribution is seldom used as a life distribution, since it is defined for all negative times. Second, and more importantly, reliability data is different from classic experimental data. Reliability data is often censored, biased, observational, and missing information about covariates such as environmental conditions. Data from testing is often expensive, resulting in small sample sizes. These problems with reliability data require sophisticated strategies and processes to mitigate them.

One consequence of these issues is that estimates based on limited data can be very imprecise.

Discipline Management

In most large programs, RAM experts report to the system engineering organization. At project or product conception, top level goals are defined for RAM based on operational needs, lifecycle cost projections, and warranty cost estimates. These lead to RAM derived requirements and allocations that are approved and managed by the system engineering requirements management function. RAM testing is coordinated with other product or system testing through the testing organization, and test failures are evaluated by the RAM function through joint meetings such as a Failure Review Board. In some cases, the RAM function may recommend design or development process changes as a result of evaluation of test results or software discrepancy reports, and these proposals must be adjudicated by the system engineering organization, or in some cases, the acquiring customer if cost increases are involved.

Post-Production Management Systems

Once a system is fielded, its reliability and availability should be tracked. Doing so allows the producer/owner to verify that the design has met its RAM objectives, to identify unexpected failure modes, to record fixes, to assess the utilization of maintenance resources, and to assess the operating environment.

One such tracking system is generically known as a FRACAS system (Failure Reporting and Corrective Action System). Such a system captures data on failures and improvements to correct failures. This database is separate from a warranty data base, which is typically run by the financial function of an organization and tracks costs only.

A FRACAS for an organization is a system, and itself should be designed following systems engineering principles. In particular, a FRACAS system supports later analyses, and those analyses impose data requirements. Unfortunately, the lack of careful consideration of the backward flow from decision to analysis to model to required data too often leads to inadequate data collection systems and missing essential information. Proper prior planning

prevents this poor performance.

Of particular importance is a plan to track data on units that have not failed. Units whose precise times of failure are unknown are referred to as censored units. Inexperienced analysts frequently do not know how to analyze censored data, and they omit the censored units as a result. This can bias an analysis.

An organization should have an integrated data system that allows reliability data to be considered with logistical data, such as parts, personnel, tools, bays, transportation and evacuation, queues, and costs, allowing a total awareness of the interplay of logistical and RAM issues. These issues in turn must be integrated with management and operational systems to allow the organization to reap the benefits that can occur from complete situational awareness with respect to RAM.

Discipline Relationships

Interactions

RAM interacts with nearly all aspects of the system development effort. Specific dependencies and interactions include:

- **Systems Engineering:** RAM interacts with systems engineering as described in the previous section.
- **Product Management (Life Cycle Cost and Warranty):** RAM interacts with the product or system lifecycle cost and warranty management organizations by assisting in the calculation of expected repair rates, downtimes, and warranty costs. RAM may work with those organizations to perform tradeoff analyses to determine the most cost-efficient solution and to price service contracts.
- **Quality Assurance:** RAM may also interact with the procurement and quality assurance organizations with respect to selection and evaluation of materials, components, and subsystems.

Dependencies

- **Systems Safety:** RAM and system safety engineers have many common concerns with respect to managing the failure behavior of a system (i.e., single points of failure and failure propagation). RAM and safety engineers use similar analysis techniques, with safety being concerned about failures affecting life or unique property and RAM being concerned with those failures as well as lower severity events that disrupt operations. RAM and system safety are both concerned with failures occurring during development and test – FRACAS is the primary methodology used for RAM; hazard tracking is the methodology used for system safety.
- **Cybersecurity:** In systems or products integrating computers and software, cybersecurity and RAM engineers have common concerns relating to the availability of cyber defenses and system event monitoring. However, there are also tradeoffs with respect to access control, boundary devices, and authentication where security device failures could impact the availability of the product or system to users.
- **Software and Hardware Engineering:** Design and RAM engineers have a common goal of creating dependable products and systems. RAM interacts with the software and hardware reliability functions through design analyses such as failure modes and effects analyses, reliability predictions, thermal analyses, reliability measurement, and component specific analyses. RAM may recommend design changes as a result of these analyses that may have to be adjudicated by program management, the customer, or systems engineering if there are cost or schedule impacts.
- **Testing:** RAM interacts with the testing program during planning to assess the most efficient (or feasible) test events to perform life testing, failure/recovery testing, and stability testing as well as to coordinate requirements for reliability or stress tests. RAM also interacts with the testing organization to assess test results and analyze failures for the implications on product or system RAM.

- **Logistics:** RAM works with logistics in providing expected failure rates and downtime constraints in order for logistics engineers to determine staffing, sparing, and special maintenance equipment requirements.

Discipline Standards

Because of the importance of reliability, availability, and maintainability, as well as related attributes, there are hundreds of standards associated. Some are general but more are specific to domains such as automotive, aviation, electric power distribution, nuclear energy, rail transportation, software, etc.Standards are produced by both governmental agencies, professional associations and international standards bodies such as:

- The International Electrotechnical Commission (IEC), Geneva, Switzerland and the closely associated International Standards Organization (ISO)
- The Institute of Electrical and Electronic Engineers (IEEE), New York, NY, USA
- The Society of Automotive Engineers (SAE), Warrendale, PA, USA
- Governmental Agencies – primarily in military and space systems

The following table lists selected standards from each of these agencies. Because of differences in domains and because many standards handle the same topic in slightly different ways, selection of the appropriate standards requires consideration of previous practices (often documented as contractual requirements), domain specific considerations, certification agency requirements, end user requirements (if different from the acquisition or producing organization), and product or system characteristics.

Table 1. Selected Reliability, Availability, Maintainability standards (SEBoK Original)

Organization	Number, Title, and Year	Domain	Comment
IEC	IEC 60812, Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA), 2006	General	
IEC	IEC 61703, Mathematical expressions for reliability, availability, maintainability and maintenance, 2001	General	
IEC	IEC 62308, Equipment reliability - Reliability assessment methods, 2006	General	
IEC	IEC 62347, Guidance on system dependability specifications, 2006	General	
IEC	IEC 62278, Railway applications – Specification and demonstration of reliability, availability, maintainability and safety (RAMS), 2002	Railways	
IEEE	IEEE Std 352-1987, IEEE Guide for General Principles of Reliability Analysis of Nuclear Power Generating Station Safety Systems, 1987	Nuclear Energy	
IEEE	IEEE Std 1044-2009, IEEE Standard Classification for Software Anomalies, 2009	Software	
IEEE	IEEE Std 1633-2008, IEEE Recommended Practice on Software Reliability, 2008	Software	
SAE	ARP 4754A, Guidelines for the Development of Civil Aircraft and Systems, 2010	Aviation	
SAE	ARP 5890, Guidelines for Preparing Reliability Assessment Plans for Electronic Engine Controls, 2011	Aviation	
SAE	J1213/2- Use of Model Verification and Validation in Product Reliability and Confidence Assessments, 2011	General	

SAE	SAE-GEIA-STD-0009, Reliability Program Standard for Systems Design, Development, and Manufacturing, 2008	General	Used by the U.S. Dept. of Defense as the primary reliability standard (replaces MIL-STD-785B)
SAE	JA 1002, Software Reliability Program Standard, 2012	Software	
U.S. Government	NASA-STD-8729.1, Planning, Developing and Managing an Effective Reliability And Maintainability (R&M) Program	Space Systems	
U.S. Government	MIL HDBK 470A, Designing and Developing Maintainable Products and Systems, 1997	Defense Systems	
U.S. Government	MIL HDBK 217F (Notice 2), Reliability Prediction of Electronic Equipment, 1995	Defense Systems	Although formally titled a “Handbook” and more than 2 decades old, the values and methods constitute a de facto standard for some U.S. military acquisitions
U.S. Government	MIL-STD-1629A, Procedures for Performing a Failure Mode Effects and Criticality Analysis - Revision A, 1980		The parent of FMEA standards produced by the IEEE, SAE, ISO, and many other agencies. Still valid and in use after 4 decades.

Personnel Considerations

Becoming a reliability engineer requires education in probability and statistics as well as the specific engineering domain of the product or system under development or in operation. A number of universities throughout the world have departments of reliability engineering (which also address maintainability and availability) and more have research groups and courses in reliability and safety – often within the context of another discipline such as computer science, systems engineering, civil engineering, mechanical engineering, or bioengineering. Because most academic engineering programs do not have a full reliability department, most engineers working in reliability have been educated in other disciplines and acquire the additional skills through additional coursework or by working with other qualified engineers. A certification in reliability engineering is available from the American Society for Quality (ASQ 2016). However, only a minority of engineers working in the discipline have this certification.

Metrics

The three basic metrics of RAM are (not surprisingly) Reliability, Maintainability, and Availability. Reliability can be characterized in terms of the parameters, mean, or any percentile of a reliability distribution. However, in most cases, the exponential distribution is used, and a single value, the mean time to failure (MTTF) for non-restorable systems, or mean time between failures (MTBF for restorable systems are used). The metric is defined as:

where is the total operating time and is the number of failures.

Maintainability is often characterized in terms of the exponential distribution and the mean time to repair and be similarly calculated, i.e.,

Where is the total down time and is the number of outages.

As was noted above, accounting for downtime requires definitions and specificity. Down time might be counted only for corrective maintenance actions, or it may include both corrective and preventive maintenance actions. Where the lognormal rather than the exponential distribution is used, a mean down time can still be calculated, but both the log of the downtimes and the variance must be known in order to fully characterize maintainability. Availability can be calculated from the total operating time and the downtime, or in the alternative, as a function of MTBF and MTTR (Mean Time To Repair.)

As was the case with maintainability, availability may be qualified as to whether it includes only unplanned failures and repairs (inherent availability) or downtime due to all causes including administrative delays, staffing outages, or

spares inventory deficiencies (operational availability).

Probabilistic metrics describe system performance for RAM. Quantiles, means, and modes of the distributions used to model RAM are also useful.

Availability has some additional definitions, characterizing what downtime is counted against a system. For **inherent availability**, only downtime associated with corrective maintenance counts against the system. For **achieved availability**, downtime associated with both corrective and preventive maintenance counts against a system. Finally, **operational availability** counts all sources of downtime, including logistical and administrative, against a system.

Availability can also be calculated instantaneously, averaged over an interval, or reported as an asymptotic value. **Asymptotic availability** can be calculated easily, but care must be taken to analyze whether or not a system settles down or settles up to the asymptotic value, as well as how long it takes until the system approaches that asymptotic value.

Reliability importance measures the effect on the system reliability of a small improvement in a component's reliability. It is defined as the partial derivative of the system reliability with respect to the reliability of a component.

Criticality is the product of a component's reliability, the consequences of a component failure, and the frequency with which a component failure results in a system failure. Criticality is a guide to prioritizing reliability improvement efforts.

Many of these metrics cannot be calculated directly because the integrals involved are intractable. They are usually estimated using simulation.

Models

There are a wide range of models that estimate and predict reliability (Meeker and Escobar 1998). Simple models, such as exponential distribution, can be useful for "back of the envelope" calculations.

System models are used to (1) combine probabilities or their surrogates, failure rates and restoration times, at the component level to find a system level probability or (2) to evaluate a system for maintainability, single points of failure, and failure propagation. The three most common are reliability block diagrams, fault trees, and failure modes and effects analyses.

There are more sophisticated probability models used for life data analysis. These are best characterized by their failure rate behavior, which is defined as the probability that a unit fails in the next small interval of time, given it has lived until the beginning of the interval, and divided by the length of the interval.

Models can be considered for a fixed environmental condition. They can also be extended to include the effect of environmental conditions on system life. Such extended models can in turn be used for accelerated life testing (ALT), where a system is deliberately and carefully overstressed to induce failures more quickly. The data is then extrapolated to usual use conditions. This is often the only way to obtain estimates of the life of highly reliable products in a reasonable amount of time (Nelson 1990).

Also useful are **degradation models**, where some characteristic of the system is associated with the propensity of the unit to fail (Nelson 1990). As that characteristic degrades, we can estimate times of failure before they occur.

The initial developmental units of a system often do not meet their RAM specifications. **Reliability growth models** allow estimation of resources (particularly testing time) necessary before a system will mature to meet those goals (Meeker and Escobar 1998).

Maintainability models describe the time necessary to return a failed repairable system to service. They are usually the sum of a set of models describing different aspects of the maintenance process (e.g., diagnosis, repair, inspection, reporting, and evacuation). These models often have threshold parameters, which are minimum times until an event can occur.

Logistical support models attempt to describe flows through a logistics system and quantify the interaction between maintenance activities and the resources available to support those activities. Queue delays, in particular, are a major source of down time for a repairable system. A logistical support model allows one to explore the trade space between resources and availability.

All these models are abstractions of reality, and so at best approximations to reality. To the extent they provide useful insights, they are still very valuable. The more complicated the model, the more data necessary to estimate it precisely. The greater the extrapolation required for a prediction, the greater the imprecision.

Extrapolation is often unavoidable, because high reliability equipment typically can have long life and the amount of time required to observe failures may exceed test times. This requires strong assumptions be made about future life (such as the absence of masked failure modes) and that these assumptions increase uncertainty about predictions. The uncertainty introduced by strong model assumptions is often not quantified and presents an unavoidable risk to the system engineer.

There are many ways to characterize the reliability of a system, including fault trees, reliability block diagrams, and failure mode effects analysis.

A **Fault Tree** (Kececioglu 1991) is a graphical representation of the failure modes of a system. It is constructed using logical gates, with AND, OR, NOT, and K of N gates predominating. Fault trees can be complete or partial; a partial fault tree focuses on a failure mode or modes of interest. They allow "drill down" to see the dependencies of systems on nested systems and system elements. Fault trees were pioneered by Bell Labs in the 1960s.

A Failure Mode Effects Analysis is a table that lists the possible failure modes for a system, their likelihood, and the effects of the failure. A Failure Modes Effects Criticality Analysis scores the effects by the magnitude of the product of the consequence and likelihood, allowing ranking of the severity of failure modes (Kececioglu 1991).

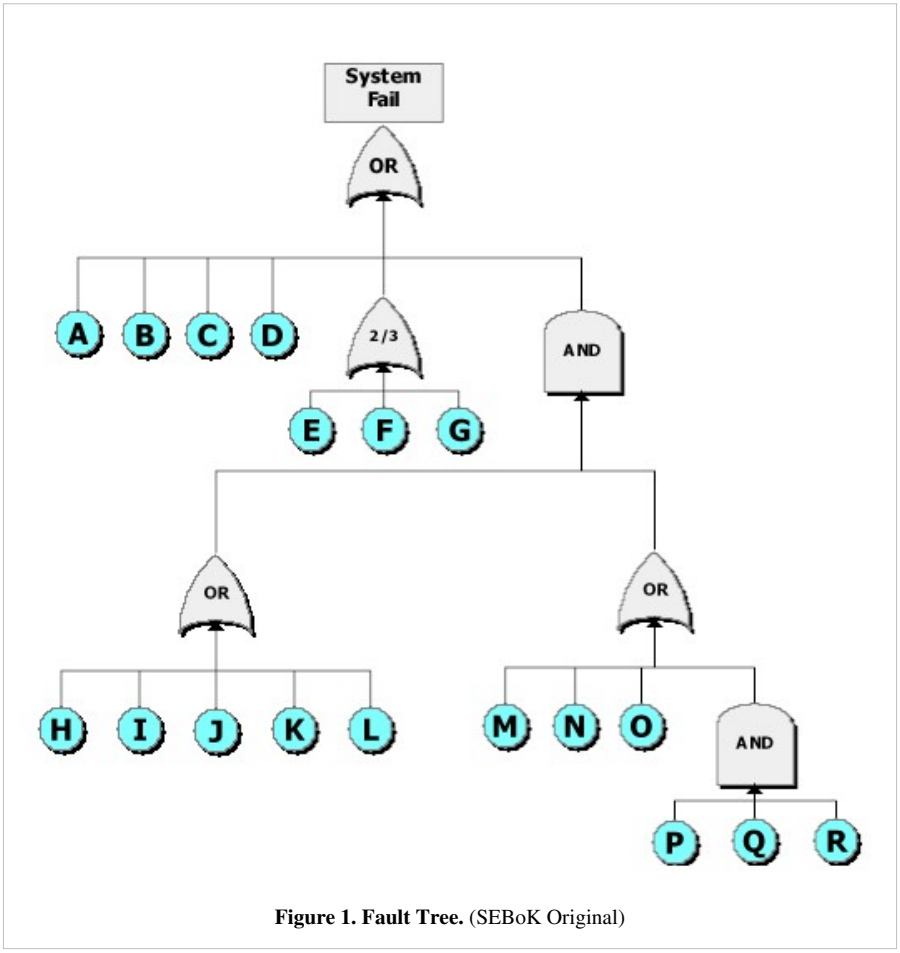


Figure 1. Fault Tree. (SEBoK Original)

A **Reliability Block Diagram** (RBD) is a graphical representation of the reliability dependence of a system on its components. It is a directed, acyclic graph. Each path through the graph represents a subset of system components. As long as the components in that path are operational, the system is operational. Component lives are usually assumed to be independent in an RBD. Simple topologies include a series system, a parallel system, a *k* of *n* system, and combinations of these.

RBDs are often nested, with one RBD serving as a component in a higher-level model. These hierarchical models allow the analyst to have the appropriate resolution of detail while still permitting abstraction.

RBDs depict paths that lead to success, while fault trees depict paths that lead to failure.

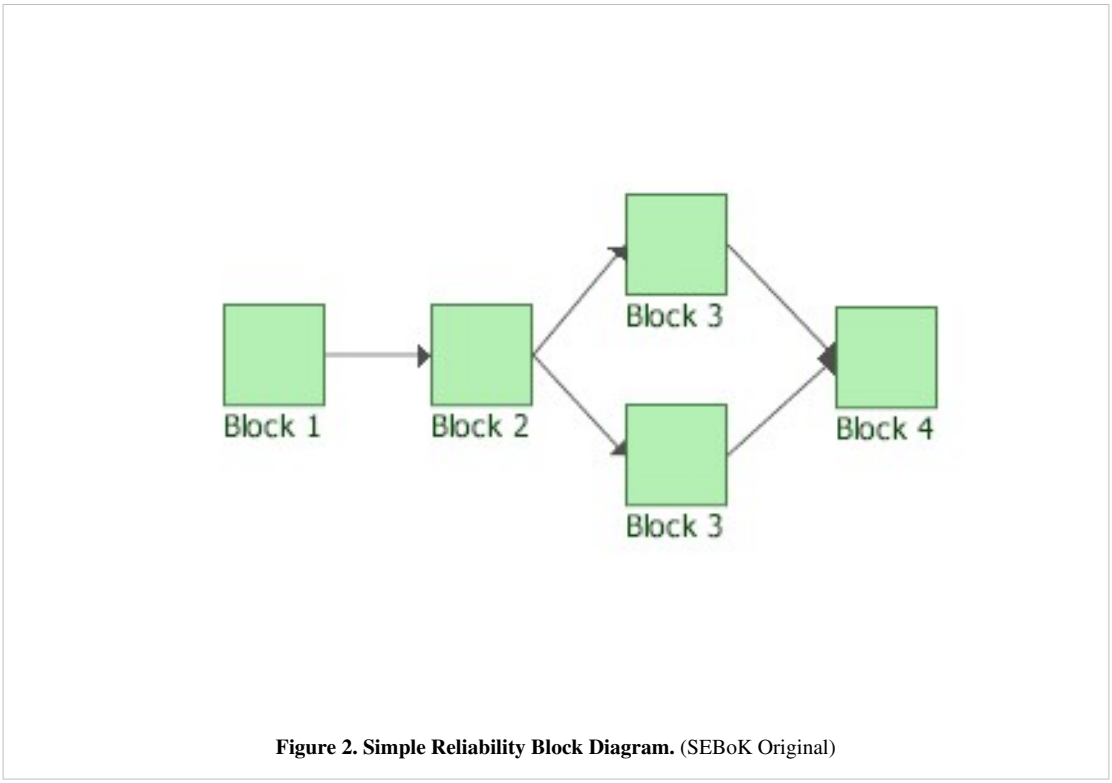


Figure 2. Simple Reliability Block Diagram. (SEBoK Original)

A **Failure Mode Effects Analysis** is a table that lists the possible failure modes for a system, their likelihood, and the effects of the failure. A **Failure Modes Effects Criticality Analysis** scores the effects by the magnitude of the product of the consequence and likelihood, allowing ranking of the severity of failure modes (Kececioglu 1991).

System models require even more data to fit them well. “Garbage in, garbage out” (GIGO) particularly applies in the case of system models.

Tools

The specialized analyses required for RAM drive the need for specialized software. While general purpose statistical languages or spreadsheets can, with sufficient effort, be used for reliability analysis, almost every serious practitioner uses specialized software.

Minitab (versions 13 and later) includes functions for life data analysis. Win Smith is a specialized package that fits reliability models to life data and can be extended for reliability growth analysis and other analyses. Relex has an extensive historical database of component reliability data and is useful for estimating system reliability in the design phase.

There is also a suite of products from ReliaSoft (2007) that is useful in specialized analyses. Weibull++ fits life models to life data. ALTA fits accelerated life models to accelerated life test data. BlockSim models system reliability, given component data.

Discipline Specific Tool Families

Reliasoft [1] and PTC Windchill Product Risk and Reliability [2] produce a comprehensive family of tools for component reliability prediction, system reliability predictions (both reliability block diagrams and fault trees), reliability growth analysis, failure modes and effects analyses, FRACAS databases, and other specialized analyses. In addition to these comprehensive tool families, there are more narrowly scoped tools. Minitab (versions 13 and later) includes functions for life data analysis.

General Purpose Statistical Analysis Software with Reliability Support

Some general-purpose statistical analysis software includes functions for reliability data analysis. Minitab [3] has a module for reliability and survival analysis. SuperSmith [4] is a more specialized package that fits reliability models to life data and can be extended for reliability growth analysis and other analyses.

R [5] is a widely used open source and well-supported general purpose statistical language with specialized packages that can be used for fitting reliability models, Bayesian analysis, and Markov modeling.

Special Purpose Analysis Tools

Fault tree generation and analysis tools include CAFTA [6] from the Electric Power Research Institute and OpenFTA [7], an open source software tool originally developed by Auvation Software.

PRISM [8] is an open source probabilistic model checker that can be used for Markov modeling (both continuous and discrete time) as well as for more elaborate analyses of system (more specifically, "timed automata") behaviors such as communication protocols with uncertainty.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

References

Works Cited

American Society for Quality (ASQ). 2011. *Glossary: Reliability.* Accessed on September 11, 2011. Available at http://asq.org/glossary/r.html.

American Society for Quality (ASQ). 2016. ''Reliability Engineering Certification – CRE''. Available at: http://asq.org/cert/reliability-engineer.

DoD. 2005. ''DOD Guide for Achieving Reliability, Availability, and Maintainability.'' Arlington, VA, USA: U.S. Department of Defense (DoD). Accessed on September 11, 2011. Available at: http://www.acq.osd.mil/se/docs/RAM_Guide_080305.pdf

Ebeling, Charles E., 2010. ''An Introduction to Reliability and Maintainability Engineering''. Long Grove Illinois, U.S.A: Waveland Press.

GEIA. 2008. ‘’Reliability Program Standard for Systems Design, Development, and Manufacturing’’. Warrendale, PA, USA: Society of Automotive Engineers (SAE), SAE-GEIA-STD-0009.

IEEE. 2008. ‘’IEEE Recommended Practice on Software Reliability’’. New York, NY, USA: Institute of Electrical and Electronic Engineers (IEEE). IEEE Std 1633-2008.

Kececioglu, D. 1991. *Reliability Engineering Handbook,* Volume 2. Upper Saddle River, NJ, USA: Prentice Hall.

Laprie, J.C., A. Avizienis, and B. Randell. 1992. ‘’Dependability: Basic Concepts and Terminology’’. Vienna, Austria: Springer-Verlag.

Nelson, W. 1990. ‘’Accelerated Testing: Statistical Models, Test Plans, and Data Analysis.’’ New York, NY, USA: Wiley and Sons.

O'Connor, D.T., and A. Kleyner. 2012. ‘’Practical Reliability Engineering’’, 5th Edition. Chichester, UK: J. Wiley & Sons, Ltd.

ReliaSoft. 2007. *Failure Modes and Effects Analysis (FMEA) and Failure Modes, Effects and Criticality Analysis (FMECA).* Accessed on September 11, 2011. Available at http://www.weibull.com/basics/fmea.htm.

Shooman, Martin. 2002. ‘’Reliability of Computer Systems and Networks’., New York, NY, USA: John Wiley & Sons.

Primary References

Blischke, W.R. and D.N. Prabhakar Murthy. 2000. *Reliability Modeling, Prediction, and Optimization.* New York, NY, USA: Wiley and Sons.

Dezfuli, H, D. Kelly, C. Smith, K. Vedros, and W. Galyean. 2009. “Bayesian Inference for NASA Risk and Reliability Analysis” National Aeronautics and Space Administration, NASA/SP-2009-569,. Available at: http:// www.hq.nasa.gov/office/codeq/doctree/SP2009569.pdf.

DoD. 2005. *DOD Guide for Achieving Reliability, Availability, and Maintainability.* Arlington, VA, USA: U.S. Department of Defense (DoD). Accessed on September 11, 2011. Available at: http://www.acq.osd.mil/se/docs/ RAM_Guide_080305.pdf [9]

Kececioglu, D. 1991. *Reliability Engineering Handbook,* Volume 2. Upper Saddle River, NJ, USA: Prentice Hall.

Lawless, J.F. 1982. *Statistical Models and Methods for Lifetime Data.* New York, NY, USA: Wiley and Sons.

Lyu, M. 1996. ‘’Software Reliability Engineering’’. New York, NY: IEEE-Wiley Press. Available at: http://www. cse.cuhk.edu.hk/~lyu/book/reliability/index.html.

Martz, H.F. and R.A. Waller. 1991. *Bayesian Reliability Analysis.* Malabar, FL, USA: Kreiger.

Meeker, W.Q. and L.A. Escobar. 1998. *Statistical Methods for Reliability Data.* New York, NY, USA: Wiley and Sons.

DoD. 2011. ‘’MIL-HDBK-189C, Department of Defense Handbook: Reliability Growth Management (14 JUN 2011).’’ Arlington, VA, USA: U.S. Department of Defense (DoD). Available at: http:/ / everyspec. com/ MIL-HDBK/MIL-HDBK-0099-0199/MIL-HDBK-189C_34842

1998. ‘’MIL-HDBK-338B, Electronic Reliability Design Handbook’’ U.S. Department of Defense Air Force Research Laboratory IFTB, Available at: http://www.weibull.com/mil_std/mil_hdbk_338b.pdf.

U.S. Naval Surface Weapons Center Carderock Division, NSWC-11. ‘’Handbook of Reliability Prediction Procedures for Mechanical Equipment.’’ Available at:http:/ / reliabilityanalyticstoolkit. appspot. com/ static/ Handbook_of_Reliability_Prediction_Procedures_for Mechanical_Equipment_NSWC-11.pdf.

Additional References

IEEE. 2013. ''IEEE Recommended Practice for Collecting Data for Use in Reliability, Availability, and Maintainability Assessments of Industrial and Commercial Power Systems, IEEE Std 3006.9-2013.'' New York, NY, USA: IEEE.

''NIST/SEMATECH Engineering Statistics Handbook 2013'' Available online at http://www.itl.nist.gov/div898/handbook/.

Olwell, D.H. 2011. "Reliability Leadership." *Proceedings of the 2001 Reliability and Maintainability M Symposium.* Philadelphia, PA, USA: IEEE. Accessed 7 March 2012 at [IEEE web site. [10]]

Reliability Analytics Toolkit, http://reliabilityanalyticstoolkit.appspot.com/ (web page containing 31 reliability and statistical analyses calculation aids), Seymour Morris, Reliability Analytics, last visited July 4, 2016

ReliaSoft. 2007. "Availability." Accessed on September 11, 2011. Available at: http://www.weibull.com/SystemRelWeb/availability.htm.

SAE. 2000a. *Aerospace Recommended Practice ARP5580: Recommended Failure Modes and Effects Analysis (FMEA) Practices for Non-Automobile Applications.* Warrendale, PA, USA: Society of Automotive Engineers (SAE) International.

SAE. 2000b. *Surface Vehicle Recommended Practice J1739: (R) Potential Failure Mode and Effects Analysis in Design (Design FMEA), Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA), and Potential Failure Mode and Effects Analysis for Machinery (Machinery FMEA).* Warrendale, PA, USA: Society of Automotive Engineers (SAE) International.

Relevant Videos

- Availability [11]
- Measuring Reliability [12]
- Reliability, Availability [13]

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

References

[1] http://www.reliasoft.com/products.htm
[2] http://www.ptc.com/product-lifecycle-management/windchill/product-risk-and-reliability
[3] https://www.minitab.com/en-us/products/minitab/look-inside/
[4] http://www.barringer1.com/wins.htm
[5] https://www.r-project.org/
[6] http://teams.epri.com/RR/News%20Archives/CAFTAFactSheet.pdf
[7] http://www.openfta.com/
[8] http://www.prismmodelchecker.org/
[9] http://www.acq.osd.mil/se/docs/RAM_Guide_080305.pdf
[10] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=902456
[11] https://www.youtube.com/watch?v=D9FFXeliHJc
[12] https://www.youtube.com/watch?v=eckn-f2R_LQ
[13] https://www.youtube.com/watch?v=jznHA-C07dg

Human Systems Integration

Lead Author: *Ben Schwartz,* ***Contributing Authors:*** *Guy Boy, Alice Squires, Paul Phister, Stuart Booth, Dick Fairley*

Human systems integration (HSI) is "the management and technical discipline of planning, enabling, coordinating, and optimizing all human-related considerations during system design, development, test, production, use and disposal of systems, subsystems, equipment and facilities." (SAE, 2019). Though used by industries around the world, HSI was initiated by the U.S. Department of Defense as part of the "total system approach" to acquisition. The goal of HSI is to "optimize total system performance (hardware, software, and human), operational effectiveness, and suitability, survivability, safety, and affordability." (DoD, 2003.) HSI activities must be initiated "early in system development (during stakeholder requirements generation) and continuously through the development process to realize the greatest benefit to the final system solution and substantial lifecycle cost savings." (INCOSE Systems Engineering Handbook, 2015).

HSI generally incorporates the following domains as integration considerations: manpower, personnel, training, human-centered design, human factors engineering, life-critical systems that include occupational health, environment, safety, habitability, and human survivability. Some organizations use a slightly different domain set.

Overview

Historically, insufficient systems engineering resources were dedicated to ensuring proper integration of humans with the rest of the system. Most projects were technology-centered with human considerations being addressed through training. Technological systems were hard to use and maintain, resulting in large manpower and training costs, reduced system performance, and increased risk of catastrophic loss, among other impacts. The U.S. Army was among the first to address this with the Manpower and Personnel Integration (MANPRINT) program in 1986. MANPRINT emphasized the consideration of the HSI domains throughout the system acquisition as a standard part of the systems engineering effort. The approach has since been adopted by the broader U.S. Department of Defense, by international militaries, and by civilian government agencies around the world. (Booher, 2003). Some organizations, particularly the U.K. Ministry of Defence, use the term Human Factors Integration (HFI). HSI applies systems engineering processes, tools, and techniques to ensure that human considerations are given proper weight in all system development activities. HSI should not be confused with Human Factors Engineering (HFE); HFE is a domain of HSI focusing on designing human interfaces. HSI is about mutual integration of technology, organizations and people.

System Description

HSI is more than human factors, human-computer interaction, or systems engineering. It is a technical and managerial set of processes that involves the consideration and integration of multiple domains. In addition, HSI involves complexity analysis and organization design and management. Various organizations represent the HSI domains differently as the number and names of the domains are aligned with existing organizational structures. Booher (2003) presents the seven US Army domains. The Canadian Forces have a different number of domains while the UK Ministry of Defence has another. All the technical work of the domains is present while the number and names and the domains is the same. According to the Defense Acquisition University, the HSI domains are:

Manpower: Manpower describes the number and mix of personnel required to carry out a task, multiple tasks, or mission in order to operate, maintain, support, and provide training for a system. Manpower factors are those variables that define manpower requirements. These variables include job tasks, operation/maintenance rates, associated workload, and operational conditions (e.g., risk of operator injury) (DAU 2010).

Environment: Environment includes the physical conditions in and around the system, as well as the operational context within which the system will be operated and supported. Environmental attributes include temperature, humidity, noise, vibration, radiation, shock, air quality, among many others. This "environment" affects the human's ability to function as a part of the system (DAU 2010).

Habitability: Habitability factors are those living and working conditions that are necessary to sustain the morale, safety, health, and comfort of the user population. They directly contribute to personnel effectiveness and mission accomplishment and often preclude recruitment and retention problems. Examples include: lighting, space, ventilation, and sanitation; noise and temperature control (i.e., heating and air conditioning); religious, medical, and food services availability; and berthing, bathing, and personal hygiene. Habitability consists of those characteristics of systems, facilities (temporary and permanent), and services necessary to satisfy personnel needs. Habitability factors are those living and working conditions that result in levels of personnel morale, safety, health, and comfort adequate to sustain maximum personnel effectiveness, support mission performance, and avoid personnel retention problems (DAU 2010).

Human-Centered Design: Human-Centered Design (HCD) combines creativity from virtual to tangible), agile prototyping, formative evaluation, rigorous demonstration and validation. It is based on design thinking, expertise, experience, organization design and management, advanced interaction media, and complexity analysis, and more specifically on considering human-systems integration using modeling and human-in-the-loop simulation from the very beginning of the design process, as well as during the whole life cycle of a system (Boy, 2017).

Human Factors Engineering: Human factors engineering is primarily concerned with designing human-machine interfaces consistent with the physical, cognitive, and sensory abilities of the user population. Human-machine interfaces include:

- functional interfaces (functions and tasks, and allocation of functions to human performance or automation);
- informational interfaces (information and characteristics of information that provide the human with the knowledge, understanding, and awareness of what is happening in the tactical environment and in the system);
- environmental interfaces (the natural and artificial environments, environmental controls, and facility design);
- co-operational interfaces (provisions for team performance, cooperation, collaboration, and communication among team members and with other personnel);
- organizational interfaces (job design, management structure, command authority, and policies and regulations that impact behavior);
- operational interfaces (aspects of a system that support successful operation of the system such as procedures, documentation, workloads, and job aids);
- cognitive interfaces (decision rules, decision support systems, provisions for maintaining situational awareness, mental models of the tactical environment, provisions for knowledge generation, cognitive skills and attitudes, and memory aids); and
- physical interfaces (hardware and software elements designed to enable and facilitate effective and safe human performance such as controls, displays, workstations, worksites, accesses, labels and markings, structures, steps and ladders, handholds, maintenance provisions, etc.) (DAU 2010).

Human Survivability: In the defense domain, survivability factors consist of those system design features that reduce the risk of fratricide, detection, and the probability of being attacked, and that enable personnel to withstand man-made hostile environments without aborting the mission or objective, or suffering acute chronic illness, disability, or death. Survivability attributes are those that contribute to the survivability of manned systems (DAU 2010).

Occupational Health: Occupational health factors are those system design features that serve to minimize the risk of injury, acute or chronic illness, or disability, and/or reduce job performance of personnel who operate, maintain, or support the system. Prevalent issues include noise, chemical safety, atmospheric hazards (including those associated with confined space entry and oxygen deficiency), vibration, ionizing and non-ionizing radiation, and

human factors issues that can create chronic disease and discomfort such as repetitive motion diseases. Many occupational health problems, particularly noise and chemical management, overlap with environmental impacts. Human factors stress that creating a risk of chronic disease and discomfort overlaps with occupational health considerations (DAU 2010).

Personnel: Personnel factors are those human aptitudes (i.e., cognitive, physical, and sensory capabilities), knowledge, skills, abilities, and experience levels that are needed to properly perform job tasks. Personnel factors are used to develop occupational specialties for system operators, maintainers, trainers, and support personnel (DAU 2010). The selection and assignment of personnel is critical to the success of a system, as determined by the needs set up by various work-related requirements.

Safety: The design features and operating characteristics of a system that serve to minimize the potential for human or machine errors or failure that cause injurious accidents (DAU, 2010). Safety also encompasses the administrative procedures and controls associated with the operations, maintenance, and storage of a system.

Training: Training is the learning process by which personnel individually or collectively acquire or enhance pre-determined job-relevant knowledge, skills, and abilities by developing their cognitive, physical, sensory, and team dynamic abilities. The "training/instructional system" integrates training concepts and strategies, as well as elements of logistic support to satisfy personnel performance levels required to operate, maintain, and support the systems. It includes the "tools" used to provide learning experiences, such as computer-based interactive courseware, simulators, actual equipment (including embedded training capabilities on actual equipment), job performance aids, and Interactive Electronic Technical Manuals (DAU 2010).

Discipline Management

In a contractor project organization, the human systems integrator is typically a member of the senior engineering staff reporting to either the systems engineering lead or chief engineer. HSI activities are documented in the Systems Engineering Management Plan (SEMP). Larger programs may have a stand-alone HSI Plan (HSIP) compatible with and referenced by the SEMP. HSI activities are tailored to the needs of the project and the project lifecycle (NASA, 2016). Most projects implement a Joint HSI Working Group between the customer and contractor. This enables sharing of priorities, knowledge, and effort to allow each group to achieve their objectives.

Discipline Relationships

Interactions

Interactions include:

- SE: HSI is an integral part of the systems engineering effort and the integrator participates in all relevant systems engineering activities during the whole life cycle of the system being considered.
- HSI domain experts: Domain experts collaborate with the human systems integrator to achieve HSI objectives, though this may or may not be a direct reporting relationship.
- The contractor and customer may each have a human systems integrator and various domain experts; each role should collaborate with their counterparts to the appropriate extent.
- HSI domain experts may participate in integrated product teams (IPTs)/design teams as full participants or consultants as appropriate for the needs of the project.
- HSI shares many concerns with Reliability, Availability, and Maintainability (RAM). The integrator and/or domain experts may collaborate with RAM specialists as appropriate.
- The integrator and/or domain experts should work with the Test & Evaluation team to ensure that HSI is represented in test and evaluation events.
- HSI shares many concerns with logistics and supportability, the integrator and/or domain experts may collaborate with this team as appropriate.

Dependencies

HSI depends on sufficient scope of work and authorization from the project. Proper planning and leadership buy-in is a key enabler.

Discipline Standards

Note: These are standards relevant to the practice of HSI specifically and not each of the HSI domains, which have their own standards and practices.

- National Aeronautics and Space Administration (NASA). 2015. Human Systems Integration Practitioner's Guide. NASA/SP-2015-3709. Houston: Johnson Space Center.
- SAE International. 2019. Standard Practice for Human Systems Integration. SAE6906. Warrendale, PA: SAE International.
- U.K. Ministry of Defence. 2016. Defence Standard 00-251: Human Factors Integration for Defence Systems. Glasglow: Defence Equipment and Support.
- U.S. Army. 2015. Regulation 602-2: Human Systems Integration in the System Acquisition Process. Washington: Headquarters, Department of the Army.
- U.S. Department of Defense. 2011. Data Item Description: Human Systems Integration Program Plan. DI-HFAC-81743A.
- U.S. Navy. 2017. Opnav Instruction 5310.23A: Navy Personnel Human Systems Integration. Washington: Office of the Chief of Naval Operations: Department of the Navy.

Personnel Considerations

HSI is conducted by a human systems integrator. The integrator is part of the systems engineering team responsible for conducting systems engineering related to human and organizational considerations and for coordinating the work of the HSI domain experts. HSI uses the same techniques and approaches as systems engineering with additional consideration for non-materiel aspects of the system. Therefore, the integrator must be well-versed in the SE process and have a working understanding of each of the domains. The integrator does not need to be an expert in any of the domains. The human systems integrator's responsibilities include:

- providing inputs to the SEMP and/or creating an HSI Plan (HSIP) compatible with the SEMP and the project lifecycle (NASA Systems Engineering Handbook, 2016)
- tailoring the scope of HSI efforts to the needs of the project and system lifecycle
- ensuring HSI domains are given appropriate consideration across all programmatic and engineering activities
- assisting domain personnel in planning domain activities
- facilitating execution of domain tasks and collaboration among domains
- making tradeoffs among domains to optimize the attainment of HSI goals
- optimizing the impact of domains on the acquisition program from the perspectives of performance, sustainability, and cost
- integrating the results of domain activities and representing them to the rest of the acquisition program from a total HSI perspective
- facilitating interactions among domains within the scope of HSI, and between HSI and the rest of the program
- tracking, statusing, and assessing HSI risks, issues and opportunities that have surfaced during the execution of the program

Metrics

Human-System Measures of Effectiveness

A measure of effectiveness (MOE) is a metric corresponding to the accomplishment of the mission objective. MOEs measure system performance in a representative mission context, including with representative users. Effectiveness is typically achieved through a combination of hardware, software, and human components, thus there are not typically HSI-specific MOEs. MOEs may be decomposed into measures of performance (MOP) and measures of suitability (MOS). There may be HSI-specific MOPs and MOSs. For example, an MOE for an air defense radar might be positive detection probability, with an MOP for the radar's effective resolution and an MOP for the operator's ability to identify the target. It is the human system integrator's responsibility to ensure that relevant MOPs and MOSs are identified and incorporated into modeling, simulation, test, and evaluation efforts. The integrator and domain experts may contribute to these efforts as appropriate.

Models

HSI Process Models

HSI shares common systems engineering models with general systems engineering (e.g. the SE Vee process model). Additionally, a number of HSI-specific models and processes exist. A particularly good resource is A User-Centered Systems Engineering Framework by Ehrhart and Sage (in Booher 2003).

Human Performance and Domain Models

A variety of human performance models exist for cognition, behavior, anthropometry, strength, fatigue, attention, situation awareness, etc. Additionally, a variety of models exist for each HSI domain. The integrator should have a good understanding of the types of models available and the appropriate applications. In a project utilizing model-based systems engineering, the system model should include humans. The integrator should ensure sufficient fidelity to meet the needs of the project. Human-in-the-loop simulations should be encouraged during the design process as during the whole life cycle of a product.

Tools

HSI shares common tools with systems engineering. A sample of HSI-specific tools include:

- Command Control and Communications - Techniques for Reliable Assessment of Concept Execution (C3TRACE) [1] developed by U.S. Army Research Labs.
- Comprehensive Human Integration Evaluation Framework (CHIEF) [2] developed by U.S. Navy.
- Human Analysis and Requirements Planning System (HARPS) [3] developed by U.S. Navy Space and Naval Warfare Systems Command.
- Human Systems Integration Framework (HSIF) [4] developed by U.S. Air Force.
- Improved Performance and Research Integration Tool (IMPRINT) [1] developed by U.S. Army Research Labs.

Additionally, each HSI domain has specific tools and approaches for their unique efforts and considerations.

Practical Considerations

Pitfalls

Many organizations assign a human factors engineer to the human systems integrator role. This can be a mistake if the individual is not well versed in the SE process. Relegating HSI to a "specialty engineering" team deprives the integrator of sufficient scope and authority to accomplish their mission.

Proven Practices

Ensure the human systems integrator is a knowledgeable systems engineer with the respect of the other systems engineers and with a good understanding of each of the HSI domains. A human systems integrator should be involved in program planning activities to ensure sufficient budget and schedule. They should be involved in technical planning activities to create sufficient scope in the SEMP/HISPP, identify HSI-related risks and opportunities, recommend HSI trade studies, etc. There is significant overlap and trade space among the HSI domains, therefore the domain experts, led by the integrator, should collaborate throughout the project to optimize the impact of HSI.

Other Considerations

None at this time.

References

Works Cited

Booher, H.R. (ed.). 2003. *Handbook of Human Systems Integration.* Hoboken, NJ, USA: Wiley.

Boy, G.A. 2017. "Human-Centered Design as an Integrating Discipline." ''Journal of Systemics, Cybernetics and Informatics.'' International Institute of Informatics and Systemics. Volume 15, Number 1, pp. 25-32. ISSN: 1690-4524.

DAU. 2010. *Defense Acquisition Guidebook (DAG).* Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/U.S. Department of Defense (DoD). February 19, 2010.

National Aeronautics and Space Administration (NASA). 2016. NASA Systems Engineering Handbook Rev. 2. https://www.nasa.gov/connect/ebooks/nasa-systems-engineering-handbook

SAE International. 2019. Standard Practice for Human Systems Integration (SAE6906). Warrendale, PA: SAE. https://saemobilus.sae.org/content/sae6906

U.S. Department of Defense (DoD). 2003. DoD Directive 5000.01, The Defense Acquisition System. https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodd/500001p.pdf

US Air Force. 2009. *Air Force Human Systems Integration Handbook.* Brooks City-Base, TX, USA: Directorate of Human Performance Integration. Available at http:/ / www. wpafb. af. mil/ shared/ media/ document/ AFD-090121-054.pdf. [5]

Primary References

None at this time.

Additional References

Blanchard, B. S., and W. J. Fabrycky. 2011. *Systems Engineering and Analysis*. 5th ed. Prentice-Hall International series in Industrial and Systems Engineering. Englewood Cliffs, NJ, USA: Prentice-Hall.

Boy, G.A. (2013). ''Orchestrating Human-Centered Design.'' Springer, U.K. ISBN 978-1-4471-4338-3.

Helander, Martin, Landauer, T.K, and Prabhu, P.V. 1997. *Handbook of Human-Computer Interaction.* Amsterdam, Netherlands: Elsevier.

Pew, R.W. and A.S. Mavor. 2007. *Human-System Integration in the System Development Process: A New Look.* Washington, DC, USA: National Academies Press.

Simpkiss, B. (2009). ''AFHSIO-001: Human Systems Integration Requirements Pocket Guide.'' Falls Church, VA: Air Force Human Systems Integration Office.

U.S. Department of Defense (DoD). 2003. DoD Instruction 5000.02, Operation of The Defense Acquisition System.

Wickens, C.D., Lee, J. D, Liu, Y., and Becker, S.E. Gordon. 2004. *An Introduction to Human Factors Engineering.* Englewood Cliffs, NJ, USA: Prentice-Hall.

Woodson, W.E, Tillman, B. and Tillman, P. 1992. "Human Factors Design Handbook: Information and Guidelines for the Design of Systems, Facilities, Equipment, and Products for Human Use." 2nd Ed. New York, NY, USA: McGraw Hill.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

References

[1] https://www.arl.army.mil/www/default.cfm?page=3200
[2] http://calhoun.nps.edu/handle/10945/42696
[3] https://www.dau.mil/cop/log/pages/topics/Manpower%20and%20Personnel.aspx
[4] https://www.acq.osd.mil/se/webinars/2015_10_06-SoSECIE-Risser-Lacson-brief.pdf
[5] http://www.wpafb.af.mil/shared/media/document/AFD-090121-054.pdf

Safety Engineering

Lead Author: *Dick Fairley*, ***Contributing Author:*** *Alice Squires*

In the most general sense, safety is freedom from harm. As an engineering discipline, system safety is concerned with minimizing hazards that can result in a mishap with an expected severity and with a predicted probability. These events can occur in elements of life-critical systems as well as other system elements. MIL-STD-882E defines system safety as “the application of engineering and management principles, criteria, and techniques to achieve acceptable risk, within the constraints of operational effectiveness and suitability, time, and cost, throughout all phases of the system life cycle" (DoD 2012). MIL-STD-882E defines standard practices and methods to apply as engineering tools in the practice of system safety. These tools are applied to both hardware and software elements of the system in question.

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

System safety engineering focuses on identifying hazards, their causal factors, and predicting the resultant severity and probability. The ultimate goal of the process is to reduce or eliminate the severity and probability of the identified hazards, and to minimize risk and severity where the hazards cannot be eliminated. MIL STD 882E defines a hazard as "a real or potential condition that could lead to an unplanned event or series of events (i.e., mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment." (DoD 2012).

While systems safety engineering attempts to minimize safety issues throughout the planning and design of systems, mishaps do occur from combinations of unlikely hazards with minimal probabilities. As a result, safety engineering is often performed in reaction to adverse events after deployment. For example, many improvements in aircraft safety come about as a result of recommendations by the National Air Traffic Safety Board based on accident investigations. Risk is defined as “a combination of the severity of the mishap and the probability that the mishap will occur" (DoD 2012, 7). Failure to identify risks to safety and the according inability to address or "control" these risks can result in massive costs, both human and economic (Roland and Moriarty 1990)."

System Description

Information to be supplied at a later date.

Discipline Management

Information to be supplied at a later date.

Discipline Relationships

Interactions

Information to be supplied at a later date.

Dependencies

Information to be supplied at a later date.

Discipline Standards

Information to be supplied at a later date.

Personnel Considerations

System Safety specialists are typically responsible for ensuring system safety. Air Force Instruction (AFI) provides the following guidance:

> *9.1. System safety disciplines apply engineering and management principles, criteria, and techniques throughout the life cycle of a system within the constraints of operational effectiveness, schedule, and costs.*
>
> *9.1.1. System safety is an inherent element of system design and is essential to supporting system requirements. Successful system safety efforts depend on clearly defined safety objectives and system requirements.*
>
> *9.1.2. System safety must be a planned, integrated, comprehensive effort employing both engineering and management resources.*

(USAF 1998, 91-202)

Safety personnel are responsible for the integration of system safety requirements, principles, procedures, and processes into the program and into lower system design levels to ensure a safe and effective interface. Two common mechanisms are the Safety Working Group (SWG) and the Management Safety Review Board (MSRB). The SWG enables safety personnel from all integrated product teams (IPTs) to evaluate, coordinate, and implement a safety approach that is integrated at the system level in accordance with MIL-STD-882E (DoD 2012). Increasingly, safety reviews are being recognized as an important risk management tool. The MSRB provides program level oversight and resolves safety related program issues across all IPTs. Table 1 provides additional information on safety.

Table 1. Safety Ontology. (SEBoK Original)

Ontology Element Name	Ontology Element Attributes	Relationships to Safety
Failure modes	Manner of failure	Required attribute
Severity	Consequences of failure	Required attribute
Criticality	Impact of failure	Required attribute
Hazard Identification	Identification of potential failure modes	Required to determine failure modes
Risk	Probability of a failure occurring	Required attribute
Mitigation	Measure to take corrective action	Necessary to determine criticality and severity

Table 1 indicates that achieving System safety involves a close tie between Safety Engineering and other specialty Systems Engineering disciplines such as Reliability and Maintainability Engineering.

System safety engineering focuses on identifying hazards, their causal factors, and predicting the resultant severity and probability. The ultimate goal of the process is to reduce or eliminate the severity and probability of the identified hazards, and to minimize risk and severity where the hazards cannot be eliminated. MIL STD 882E defines a hazard as "a real or potential condition that could lead to an unplanned event or series of events (i.e., mishap) resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment." (DoD 2012).

While systems safety engineering attempts to minimize safety issues throughout the planning and design of systems, mishaps do occur from combinations of unlikely hazards with minimal probabilities. As a result, safety engineering is often performed in reaction to adverse events after deployment. For example, many improvements in aircraft

safety come about as a result of recommendations by the National Air Traffic Safety Board based on accident investigations. Risk is defined as “a combination of the severity of the mishap and the probability that the mishap will occur" (DoD 2012, 7). Failure to identify risks to safety, and the according inability to address or "control" these risks, can result in massive costs, both human and economic (Roland and Moriarty 1990)."

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be supplied at a later date.

References

Works Cited

DoD. 2012. *Standard practice for System Safety.* Arlington, VA, USA: Department of Defense (DoD). MIL-STD 882E. Accessed 4 November 2014 at http://assistdoc1.dla.mil/qsDocDetails.aspx?ident_number=36027.

Roland, H.E. and B. Moriarty. 1990. *System Safety Engineering and Management.* Hoboken, NJ, USA: Wiley-IEEE.

USAF. 1998. *The US Air Force Mishap Prevention Program.* Washington, DC, USA: US Air Force, Air Force Instruction (AFI).

Primary References

None.

Additional References

Bahr, N.J. 2001. "System Safety Engineering and Risk Assessment." In *International Encyclopedia of Ergonomics and Human Factors.* Vol. 3. Ed. Karwowski, Waldemar. New York, NY, USA: Taylor and Francis.

ISSS. "System Safety Hazard Analysis Report." The International System Safety Society (ISSS). DI-SAFT-80101B. http://www.system-safety.org/Documents/DI-SAFT-80101B_SSHAR.DOC.

ISSS. "Safety Assessment Report." The International System Safety Society (ISSS). DI-SAFT-80102B. http://www.system-safety.org/Documents/DI-SAFT-80102B_SAR.DOC.

ISSS. "Engineering Change Proposal System Safety Report." The International System Safety Society (ISSS). DI-SAFT-80103B. http://www.system-safety.org/Documents/DI-SAFT-80103B_ECPSSR.DOC.

ISSS. "Waiver or Deviation System Safety Report." The International System Safety Society (ISSS). DI-SAFT-80104B. http://www.system-safety.org/Documents/DI-SAFT-80104B_WDSSR.DOC.

ISSS. "System Safety Program Progress Report." The International System Safety Society (ISSS). DI-SAFT-80105B. http://www.system-safety.org/Documents/DI-SAFT-80105B_SSPPR.DOC.

ISSS. "Health Hazard Assessment Report." The International System Safety Society (ISSS). DI-SAFT-80106B. http://www.system-safety.org/Documents/DI-SAFT-80106B_HHAR.DOC.

ISSS. "Explosive Ordnance Disposal Data." The International System Safety Society (ISSS). DI-SAFT-80931B. http://www.system-safety.org/Documents/DI-SAFT-80931B_EODD.pdf.

ISSS. "Explosive Hazard Classification Data." The International System Safety Society (ISSS). DI-SAFT-81299B. http://www.system-safety.org/Documents/DI-SAFT-81299B_EHCD.pdf.

ISSS. "System Safety Program Plan (SSPP)." The International System Safety Society (ISSS). DI-SAFT-81626. http://www.system-safety.org/Documents/DI-SAFT-81626_SSPP.pdf.

ISSS. "Mishap Risk Assessment Report." The International System Safety Society (ISSS). DI-SAFT-81300A. http://www.system-safety.org/Documents/DI-SAFT-81300A_MRAR.DOC.

Joint Software System Safety Committee. 1999. *Software System Safety Handbook.* Accessed 7 March 2012 at http://www.system-safety.org/Documents/Software_System_Safety_Handbook.pdf.

Leveson, N. 2011. ''Engineering a Safer World: Systems Thinking Applied to Safety.'' Cambridge, Mass: MIT Press.

Leveson, N. G. 2012. "Complexity and safety." In *Complex Systems Design & Management,* ed. Omar Hammami, Daniel Krob, and Jean-Luc Voirin, 27–39. Springer, Berlin, Heidelberg. http:/ / dx. doi. org/ 10. 1007/ 978-3-642-25203-7_2.

NASA. 2004. *NASA Software Safety Guidebook.* Accessed 7 March 2012 at [[1]].

Roland, H. E., and Moriarty, B. 1985. *System Safety Engineering and Management.* New York, NY, USA: John Wiley.

SAE. 1996. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.* ARP 4761. Warrendale, PA, USA: Society of Automotive Engineers. Accessed 28 August 2012 at [http://standards.sae.org/arp4761/ [2]].

SAE. 1996. *Certification Considerations for Highly-Integrated or Complex Aircraft Systems.* ARP 4754. Warrendale, PA, USA: Society of Automotive Engineers. Accessed 28 August 2012 at [http://standards.sae.org/arp4754/ [3]].

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

References

[1] http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf
[2] http://standards.sae.org/arp4761/
[3] http://standards.sae.org/arp4754/

Security Engineering

Lead Author: *Dick Fairley*, ***Contributing Author:*** *Alice Squires*

Security engineering is concerned with building systems that remain secure despite malice or error. It focuses on the tools, processes, and methods needed to design and implement complete systems that proactively and reactively mitigate vulnerabilities. Security engineering is a primary discipline used to achieve system assurance.

The term System Security Engineering (SSE) is used to denote this specialty engineering field and the US Department of Defense define it as: *"an element of system engineering that applies scientific and engineering principles to identify security vulnerabilities and minimize or contain risks associated with these vulnerabilities"* (DODI5200.44, 12).

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

Security engineering incorporates a number of cross-disciplinary skills, including cryptography, computer security, tamper-resistant hardware, applied psychology, supply chain management, and law. Security requirements differ greatly from one system to the next. System security often has many layers built on user authentication, transaction accountability, message secrecy, and fault tolerance. The challenges are protecting the right items rather than the wrong items and protecting the right items but not in the wrong way.

Security engineering is an area of increasing emphasis in the defense domain. Baldwin et al. (2012) provide a survey of the issues and a detailed reference list.

The primary objective of System Security Engineering (SSE) is to minimize or contain defense system vulnerabilities to known or postulated security threats and to ensure that developed systems protect against these threats. Engineering principles and practices are applied during all system development phases to identify and reduce these system vulnerabilities to the identified system threats.

The basic premise of SSE is recognition that an initial investment in “engineering out” security vulnerabilities and “designing-in” countermeasures is a long-term benefit and cost saving measure. Further, SSE provides a means to ensure adequate consideration of security requirements, and, when appropriate, that specific security-related designs are incorporated into the overall system design during the engineering development program. Security requirements include: physical; personnel; procedural; emission; transmission; cryptographic; communications; operations; and, computer security.

There may be some variation in the SSE process from program to program, due mainly to the level of design assurance—that is, ensuring that appropriate security controls have been implemented correctly as planned—required of the contractor. These assurance requirements are elicited early in the program (where they can be adequately planned), implemented, and verified in due course of the system development.

The System Security Engineering Management Plan (SSEMP) is a key document to develop for SSE. The SSEMP identifies the planned security tasks for the program and the organizations and individuals responsible for security aspects of the system. The goals of the SSEMP are to ensure that pertinent security issues are raised at the appropriate points in the program, to ensure adequate precautions are taken during design, implementation, test, and fielding, and to ensure that only an acceptable level of risk is incurred when the system is released for fielding. The SSEMP forms the basis for an agreement with SSE representing the developer, the government program office, the certifier, the accreditor, and any additional organizations that have a stake in the security of the system. The SSEMP identifies the major tasks for certification & accreditation (C&A), document preparation, system evaluation, and engineering; identifies the responsible organizations for each task; and presents a schedule for the completion of

those tasks.

SSE security planning and risk management planning includes task and event planning associated with establishing statements of work and detailed work plans as well as preparation and negotiation of SSE plans with project stakeholders. For each program, SSE provides the System Security Plan (SSP) or equivalent. An initial system security Concept of Operations (CONOPS) may also be developed. The SSP provides: the initial planning of the proposed SSE work scope; detailed descriptions of SSE activities performed throughout the system development life cycle; the operating conditions of the system; the security requirements; the initial SSE risk assessment (includes risks due to known system vulnerabilities and their potential impacts due to compromise and/or data loss); and, the expected verification approach and validation results.

These plans are submitted with the proposal and updated as required during engineering development. In the case where a formal C&A is contracted and implemented, these plans comply with the government's C&A process, certification responsibilities, and other agreement details, as appropriate. The C&A process is the documented agreement between the customer and contractor on the certification boundary. Upon agreement of the stakeholders, these plans guide SSE activities throughout the system development life cycle.

System Assurance

NATO AEP-67 (Edition 1), Engineering for System Assurance in NATO Programs, defines system assurance as:

…the justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle... This confidence is achieved by system assurance activities, which include a planned, systematic set of multi-disciplinary activities to achieve the acceptable measures of system assurance and manage the risk of exploitable vulnerabilities. (NATO 2010, 1)

The NATO document is organized based on the life cycle processes in ISO/IEC 15288:2008 and provides process and technology guidance to improve system assurance.

Software Assurance

Since most modern systems derive a good portion of their functionality from software, software assurance becomes a primary consideration in systems assurance. The Committee on National Security Systems (CNSS) (2010, 69) defines software assurance as a "level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner."

Goertzel, et. al (2008, 8) point out that "the reason software assurance matters is that so many business activities and critical functions—from national defense to banking to healthcare to telecommunications to aviation to control of hazardous materials—depend on the on the correct, predictable operation of software."

System Description

Robust security design explicitly rather than implicitly defines the protection goals. The Certified Information Systems Security Professional (CISSP) Common Body of Knowledge (CBK) partitions robust security into ten domains (Tipton 2006):

1. Information security governance and risk management addresses the framework, principles, policies, and standards that establish the criteria and then assess the effectiveness of information protection. Security risk management contains governance issues, organizational behavior, ethics, and security awareness training.

2. Access control is the procedures and mechanisms that enable system administrators to allow or restrict operation and content of a system. Access control policies determine what processes, resources, and operations users can invoke.

3. Cryptography can be defined as the principles and methods of disguising information to ensure its integrity, confidentiality, and authenticity during communications and while in storage. Type I devices are certified by the US National Security Agency (NSA) for classified information processing. Type 2 devices are certified by NSA for proprietary information processing. Type 3 devices are certified by NSA for general information processing. Type 4 devices are produced by industry or other nations without any formal certification.

4. Physical (environmental) security addresses the actual environment configuration, security procedures, countermeasures, and recovery strategies to protect the equipment and its location. These measures include separate processing facilities, restricted access into those facilities, and sweeps to detect eavesdropping devices.

5. Security architecture and design contains the concepts, processes, principles, and standards used to define, design, and implement secure applications, operating systems, networks, and equipment. The security architecture must integrate various levels of confidentiality, integrity, and availability to ensure effective operations and adherence to governance.

6. Business continuity and disaster recovery planning are the preparations and practices which ensure business survival given events, natural or man-made, which cause a major disruption in normal business operations. Processes and specific action plans must be selected to prudently protect business processes and to ensure timely restoration.

7. Telecommunications and network security are the transmission methods and security measures used to provide integrity, availability, and confidentiality of data during transfer over private and public communication networks.

8. Application development security involves the controls applied to application software in a centralized or distributed environment. Application software includes tools, operating systems, data warehouses, and knowledge systems.

9. Operations security is focused on providing system availability for end users while protecting data processing resources both in centralized data processing centers and in distributed client/server environments.

10. Legal, regulations, investigations, and compliance issues include the investigative measures to determine if an incident has occurred and the processes for responding to such incidents.

One response to the complexity and diversity of security needs and domains that contribute to system security is “defense in depth,” a commonly applied architecture and design approach. Defense in depth implements multiple layers of defense and countermeasures, making maximum use of certified equipment in each layer to facilitate system accreditation.

Discipline Management

Information to be supplied at a later date.

Discipline Relationships

Interactions

Information to be supplied at a later date.

Dependencies

Web-based Resource

A good online resource for system and software assurance is the US Department of Homeland Security’s Build Security In [1] web site (DHS 2010), which provides resources for best practices, knowledge, and tools for engineering secure systems.

Discipline Standards

Information to be supplied at a later date.

Personnel Considerations

Information to be supplied at a later date.

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be supplied at a later date.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

References

Works Cited

Baldwin, K., J. Miller, P. Popick, and J. Goodnight. 2012. *The United States Department of Defense Revitalization of System Security Engineering Through Program Protection.* Proceedings of the 2012 IEEE Systems Conference, 19-22 March 2012, Vancouver, BC, Canada. Accessed 28 August 2012 at http://www.acq.osd.mil/se/docs/IEEE-SSE-Paper-02152012-Bkmarks.pdf [2].

CNSS. 2010. *National Information Assurance Glossary", Committee on National Security Systems Instruction (CNSSI) no. 4009". Fort Meade, MD, USA: The Committee on National Security Systems.*

DHS. 2010. *Build Security In.* Washington, DC, USA: US Department of Homeland Security (DHS). Accessed September 11, 2011. Available: https://buildsecurityin.us-cert.gov.

DODI5200.44, United States Department of Defense, *Protection of Mission Critical Functions to Achieve Trusted Systems and Networks*, Department of Defense Instruction Number 5200.44, November 2012, Accessed 3 November 2014 at Defense Technical Information Center http://www.dtic.mil/whs/directives/corres/pdf/520044p.pdf [3].

Goertzel, K., et al. 2008. *Enhancing the Development Life Cycle to Produce Secure Software: A Reference Guidebook on Software Assurance.* Washington, DC, USA: Data and Analysis Center for Software (DACS)/US Department of Homeland Security (DHS).

NATO. 2010. *Engineering for System Assurance in NATO programs.* Washington, DC, USA: NATO Standardization Agency. DoD 5220.22M-NISPOM-NATO-AEP-67.

Tipton, H.F. (ed.). 2006. *Official (ISC)2 guide to the CISSP CBK,* 1st ed. Boston, MA, USA: Auerbach Publications.

Primary References

Anderson, R.J. 2008. *Security Engineering: A Guide to Building Dependable Distributed Systems,* 2nd Ed. New York, NY, USA: John Wiley & Sons. Accessed October 24, 2014 at http://www.cl.cam.ac.uk/~rja14/book.html

DAU. 2012. "Defense Acquisition Guidebook (DAG): Chapter 13 -- Program Protection." Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/U.S. Department of Defense (DoD). November 8, 2012. Accessed October 24, 2014 at https://dag.dau.mil/ [4]

ISO. 2008. "Information technology -- Security techniques -- Systems Security Engineering -- Capability Maturity Model® (SSE-CMM®)," Second Edition. Geneva, Switzerland: International Organization for Standardization (ISO), ISO/IEC 21827:2008.

ISO/IEC. 2013. "Information technology — Security techniques — Information security management systems — Requirements," Second Edition. Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC), ISO/IEC 27001:2013.

Kissel, R., K. Stine, M. Scholl, H. Rossman, J. Fahlsing, J. Gulick. 2008. "Security Considerations in the System Development Life Cycle," Revision 2. Gaithersburg, MD. National Institute of Standard and Technology (NIST), NIST 800-64 Revision 2:2008. Accessed October 24, 2014 at the Computer Security Resource Center [5]

Ross, R., J.C. Oren, M. McEvilley. 2014. "Systems Security Engineering: An Integrated Approach to Building Trustworthy Resilient Systems." Gaithersburg, MD. National Institute of Standard and Technology (NIST) Special Publication (SP), NIST SP 800-160:2014 (Initial Public Draft). Accessed October 24, 2014 at the Computer Security Resource Center http://csrc.nist.gov/publications/drafts/800-160/sp800_160_draft.pdf [6]

Additional References

Allen, Julia; Barnum, Sean; Ellison, Robert; McGraw, Gary; and Mead, Nancy. 2008. *Software security engineering: a guide for project managers.* New York, NY, USA: Addison Wesley Professional.

ISO. 2005. *Information technology -- Security techniques -- Code of practice for information security management.* Geneva, Switzerland: International Organization for Standardization (ISO). ISO/IEC 27002:2005.

Jurjens, J. 2005. "Sound Methods and effective tools for model-based security engineering with UML." *Proceedings of the 2005 International Conference on Software Engineering.* Munich, GE: ICSE, 15-21 May.

MITRE. 2012. "Systems Engineering for Mission Assurance." In *Systems Engineering Guide.* Accessed 19 June 2012 at MITRE http:/ / www. mitre. org/ work/ systems_engineering/ guide/ enterprise_engineering/ se_for_mission_assurance/ [7].

NIST SP 800-160. *Systems Security Engineering - An Integrated Approach to Building Trustworthy Resilient Systems*. National Institute of Standards and Technology, U.S. Department of Commerce, Special Publication 800-160. Accessed October 24, 2014 at the Computer Security Resource Center http://csrc.nist.gov/publications/ drafts/800-160/sp800_160_draft.pdf [6].

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

References

[1] https://buildsecurityin.us-cert.gov/
[2] http://www.acq.osd.mil/se/docs/IEEE-SSE-Paper-02152012-Bkmarks.pdf
[3] http://www.dtic.mil/whs/directives/corres/pdf/520044p.pdf
[4] https://dag.dau.mil/
[5] http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf
[6] http://csrc.nist.gov/publications/drafts/800-160/sp800_160_draft.pdf
[7] http://www.mitre.org/work/systems_engineering/guide/enterprise_engineering/se_for_mission_assurance/

Electromagnetic Interference/Electromagnetic Compatibility

Lead Author: *Paul Phister*, ***Contributing Authors:*** *Scott Jackson, Richard Turner, John Snoderly, Alice Squires*

Electromagnetic Interference (EMI) is the disruption of operation of an electronic device when it is in the vicinity of an electromagnetic field in the radio frequency (RF) spectrum. Many electronic devices fail to work properly in the presence of strong RF fields. The disturbance may interrupt, obstruct, or otherwise degrade or limit the effective performance of the circuit. The source may be any object, artificial or natural, that carries rapidly changing electrical currents.

Electromagnetic Compatibility (EMC) is the ability of systems, equipment, and devices that utilize the electromagnetic spectrum to operate in their intended operational environments without suffering unacceptable degradation or causing unintentional degradation because of electromagnetic radiation or response. It involves the application of sound electromagnetic spectrum management; system, equipment, and device design configuration that ensures interference-free operation; and clear concepts and doctrines that maximize operational effectiveness (DAU 2010, Chapter 7).

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

Spectrum

Each nation has the right of sovereignty over the use of its spectrum and must recognize that other nations reserve the same right. It is essential that regional and global forums exist for the discussion and resolution of spectrum development and infringement issues between bordering and proximal countries that might otherwise be difficult to resolve.

The oldest, largest, and unquestionably the most important such forum, with 193 member countries, is the International Telecommunications Union (ITU) agency of the United Nations, which manages spectrum at a global level. As stated in Chapter 3 of the NTIA Manual, “The International Telecommunication Union (ITU)...is responsible for international frequency allocations, worldwide telecommunications standards and telecommunication development activities” (NTIA 2011, 3-2). The broad functions of the ITU are the regulation, coordination and development of international telecommunications.

The spectrum allocation process is conducted by many different international telecommunication geographical committees. Figure 1 shows the various international forums represented worldwide.

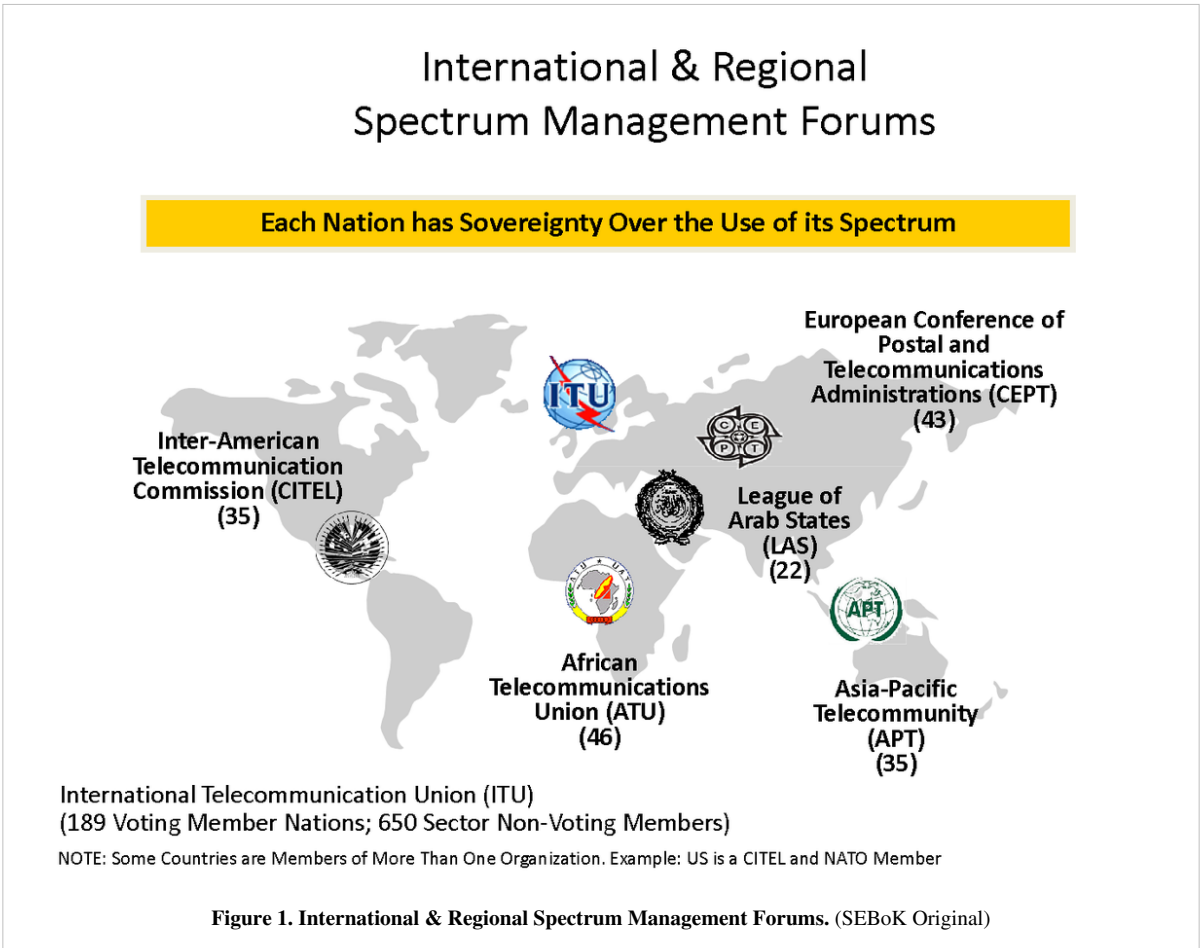


Figure 1. International & Regional Spectrum Management Forums. (SEBoK Original)

Assigning frequencies is very complicated, as shown in the radio spectrum allocation chart in Figure 2. Sometimes, commercial entities try to use frequencies that are actually assigned to US government agencies, such as the Department of Defense (DoD). One such incident occurred when an automatic garage door vendor installed doors on homes situated near a government installation. Random opening and closing of the doors created a problem for the vendor that could have been avoided.

Four ITU organizations affect spectrum management (Stine and Portigal 2004):

1. World Radio-communication Conference (WRC)
2. Radio Regulations Board (RRB)
3. Radio-communications Bureau (RB)
4. Radio-communication Study Groups (RSG)

The WRC meets every four years to review and modify current frequency allocations. The RB registers frequency assignments and maintains the master international register. The RRB approves the Rules of Procedures used by the BR to register frequency assignments and adjudicates interference conflicts among member nations. The SG analyzes spectrum usage in terrestrial and space applications and makes allocation recommendations to the WRC. Most member nations generally develop national frequency allocation polices that are consistent with the Radio Regulations (RR). These regulations have treaty status.

Dual Management of Spectrum in the US

Whereas most countries have a single government agency to perform the spectrum management function, the US has a dual management scheme intended to insure that

- decisions concerning commercial interests are made only after considering their impact on government systems; and
- government usage supports commercial interests.

The details of this scheme, established by the Communications Act of 1934, are as follows:

- the Federal Communications Commission (FCC) is responsible for all non-government usage
- the FCC is directly responsible to Congress;
- the president is responsible for federal government usage, and by executive order, delegates the federal government spectrum management to the National *Telecommunications and Information Administration (NTIA); and
- the NTIA is under the authority of the Secretary of Commerce.

The FCC regulates all non-federal government telecommunications under Title 47 of the Code of Federal Regulations. For example, see FCC (2009, 11299-11318). The FCC is directed by five Commissioners appointed by the president and confirmed by the Senate for five-year terms. The Commission staff is organized by function. The responsibilities of the six operating Bureaus include processing applications for licenses, analyzing complaints, conducting investigations, implementing regulatory programs, and conducting hearings (http://www.fcc.gov).

The NTIA performs spectrum management function through the Office of Spectrum Management (OSM), governed by the Manual of Regulations and Procedures for Federal Radio Frequency Management. The IRAC develops and executes policies, procedures, and technical criteria pertinent to the allocation, management, and usage of spectrum. The Spectrum Planning and Policy Advisory Committee (SPAC) reviews the reviews IRAC plans, balancing considerations of manufacturing, commerce, research, and academic interests.

Within the DoD, spectrum planning and routine operation activities are cooperatively managed. Spectrum certification is a mandated process designed to ensure that

1. frequency band usage and type of service in a given band are in conformance with the appropriate national and international tables of frequency allocations;
2. equipment conforms to all applicable standards, specifications, and regulations; and
3. approval is provided for expenditures to develop equipment dependent upon wireless communications.

Host Nation Coordination and Host Nation Approval

In peacetime, international spectrum governance requires military forces to obtain host nation permission — Host Nation Coordination (HNC)/Host Nation Approval (HNA) — to operate spectrum-dependent systems and equipment within a sovereign nation. For example, international governance is honored and enforced within the United States by the US departments of State, Defense, and the user service.

In wartime, international spectrum governance is not honored between warring countries; however, the sovereign spectrum rights of bordering countries must be respected by military forces executing their assigned missions. For example, HNA is solicited by US naval forces to use spectrum-dependent systems and equipment in bordering countries' airspace and/or on bordering countries' soil. HNA must be obtained before the operation of spectrum-dependent systems and equipment within a sovereign nation. The combatant commander is responsible for coordinating requests with sovereign nations within his or her area of responsibility. Because the combatant commander has no authority over a sovereign nation, the HNC/HNA process can be lengthy and needs to be started early in the development of a system. Figure 2 illustrates a spectrum example.

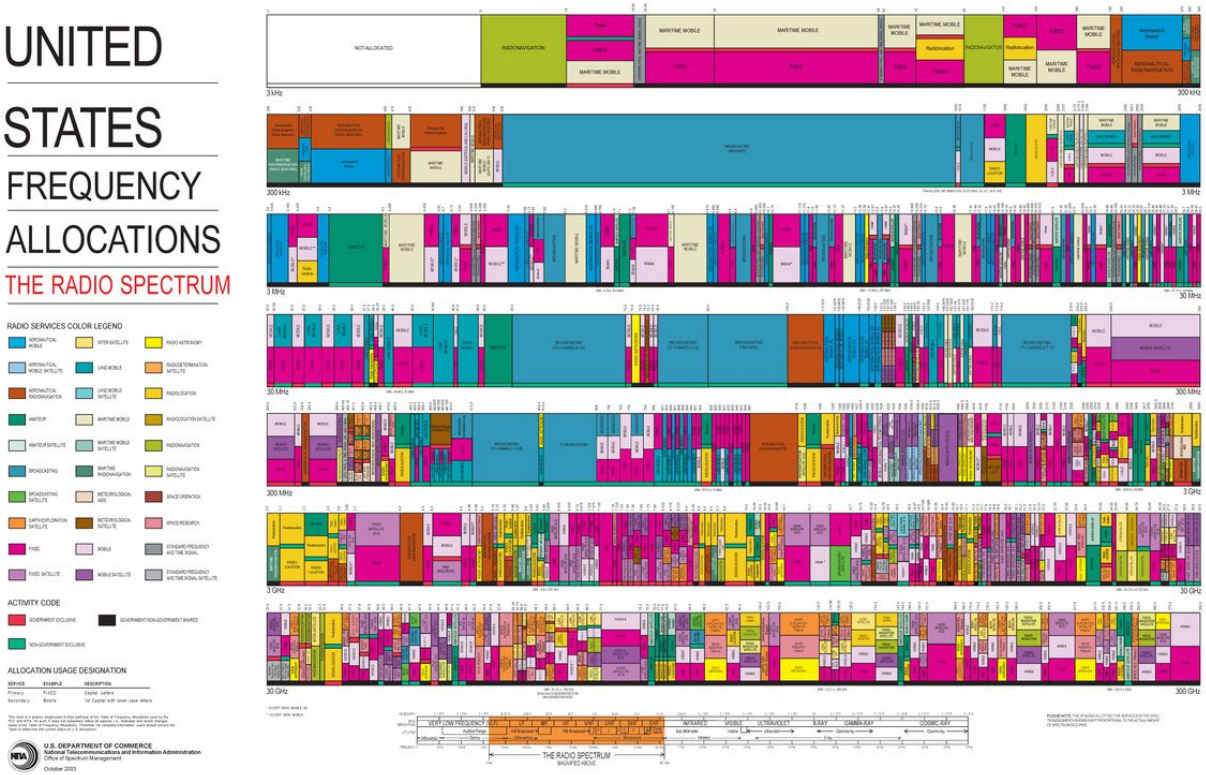


Figure 2. The Radio Spectrum (Department of Commerce 2003). Released by the U.S. Department of Commerce. Source is available at http://www.ntia.doc.gov/files/ntia/publications/2003-allochrt.pdf (Retrieved September 15, 2011)

Practical Considerations

EMI/EMC is difficult to achieve for systems that operate world-wide because of the different frequencies in which products are designed to operate in each of the telecommunication areas. Billions of US dollars have been spent in retrofitting US DoD equipment to operate successfully in other countries.

It is important to note that the nuclear radiation environment is drastically more stressing than, and very different from, the space radiation environment.

System Description

Narrowband and Broadband Emissions

To help in analyzing conducted and radiated interference effects, EMI is categorized into two types—narrowband and broadband—which are defined as follows:

- **Narrowband Emissions**

  A narrowband signal occupies a very small portion of the radio spectrum… Such signals are usually continuous sine waves (CW) and may be continuous or intermittent in occurrence… Spurious emissions, such as harmonic outputs of narrowband communication transmitters, power-line hum, local oscillators, signal generators, test equipment, and many other man made sources are narrowband emitters. (Bagad 2009, G-1)

- **Broadband Emissions**

  A broadband signal may spread its energy across hundreds of megahertz or more… This type of signal is composed of narrow pulses having relatively short rise and fall times. Broadband signals are further

> *divided into random and impulse sources. These may be transient, continuous or intermittent in occurrence. Examples include unintentional emissions from communication and radar transmitters, electric switch contacts, computers, thermostats, ignition systems, voltage regulators, pulse generators, and intermittent ground connections.* (Bagad 2009, G-1)

TEMPEST

TEMPEST is a codename used to refer to the field of emission security. The National Security Agency (NSA) investigations conducted to study compromising emission (CE) were codenamed TEMPEST. National Security Telecommunications Information Systems Security Issuance (NSTISSI)-7000 states:

> *Electronic and electromechanical information-processing equipment can produce unintentional intelligence-bearing emanations, commonly known as TEMPEST. If intercepted and analyzed, these emanations may disclose information transmitted, received, handled, or otherwise processed by the equipment.* (NSTISS 1993, 3)

These compromising emanations consist of electrical, mechanical, or acoustical energy intentionally or unintentionally emitted by sources within equipment or systems which process national security information. Electronic communications equipment needs to be secured from potential eavesdroppers while allowing security agencies to intercept and interpret similar signals from other sources. The ranges at which these signals can be intercepted depends upon the functional design of the information processing equipment, its installation, and prevailing environmental conditions.

Electronic devices and systems can be designed, by means of Radiation Hardening techniques, to resist damage or malfunction caused by ionizing and other forms of radiation (Van Lint and Holmes Siedle 2000). Electronics in systems can be exposed to ionizing radiation in the Van Allen radiation belts around the Earth's atmosphere, cosmic radiation in outer space, gamma or neutron radiation near nuclear reactors, and electromagnetic pulses (EMP) during nuclear events.

A single charged particle can affect thousands of electrons, causing electronic noise that subsequently produces inaccurate signals. These errors could affect safe and effective operation of satellites, spacecraft, and nuclear devices. Lattice displacement is permanent damage to the arrangement of atoms in element crystals within electronic devices. Lattice displacement is caused by neutrons, protons, alpha particles, and heavy ions. Ionization effects are temporary damages that create latch-up glitches in high power transistors and soft errors like bit flips in digital devices. Ionization effects are caused by charged particles.

Most radiation-hardened components are based on the functionality of their commercial equivalents. Design features and manufacturing variations are incorporated to reduce the components' susceptibility to interference from radiation. Physical design techniques include insulating substrates, package shielding, chip shielding with depleted boron, and magneto-resistive RAM. Logical design techniques include error-correcting memory, error detection in processing paths, and redundant elements at both circuit and subsystem levels (Dawes 1991). Nuclear hardness is expressed as susceptibility or vulnerability for given environmental conditions. These environmental conditions include peak radiation levels, overpressure, dose rates, and total dosage.

Discipline Management

Information to be supplied at a later date.

Discipline Relationships

Interactions

Information to be supplied at a later date.

Dependencies

Information to be supplied at a later date.

Discipline Standards

Information to be supplied at a later date.

Personnel Considerations

Information to be supplied at a later date.

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be provided at a later date.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

References

Works Cited

Bagad, V.S. 2009. *Electronic Product Design,* 4th ed. Pune, India: Technical Publications Pune.

DAU. 2010. *Defense Acquisition Guidebook (DAG).* Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/U.S. Department of Defense (DoD). February 19, 2010.

NSTISS. 1993. *Tempest Countermeasures for Facilities.* Ft. Meade, MD, USA: National Security Telecommunications and Information Systems Security (NSTISSI). 29 November, 1993. NSTISSI No. 7000.

NTIA. 2011. *Manual of Regulations and Procedures for Federal Radio Frequency Management,* May 2011 Revision of the 2008 Edition. Washington, DC, USA: National Telecommunications and Information Administration, U.S. Department of Commerce.

Stine, J. and D. Portigal. 2004. *An Introduction to Spectrum Management.* Bedford, MA, USA: MITRE Technical Report Spectrum. March 2004.

Van Lint, V.A.J. and A.G. Holmes Siedle. 2000. *Radiation Effects in Electronics: Encyclopedia of Physical Science and Technology.* New York, NY, USA: Academic Press.

Primary References

DAU. 2010. *Defense Acquisition Guidebook (DAG).* Ft. Belvoir, VA, USA: Defense Acquisition University (DAU)/U.S. Department of Defense (DoD). February 19, 2010.

Additional References

None.

< Previous Article | Parent Article | Next Article >

System Resilience

Lead Author: *John Brtis*, ***Contributing Authors:*** *Scott Jackson, Alice Squires, Richard Turner*

According to the Oxford English Dictionary on Historical Principles (1973), resilience is “the act of rebounding or springing back.” This definition most directly fits the situation of materials which return to their original shape after deformation. For human-made, or engineered systems the definition of resilience can be extended to include the ability to maintain capability in the face of a disruption. The US government definition for resilient infrastructure systems is the "ability of systems, infrastructures, government, business, communities, and individuals to resist, tolerate, absorb, recover from, prepare for, or adapt to an adverse occurrence that causes harm, destruction, or loss of national significance” (DHS 2010).

The name **Resilience Engineering** was coined in the book Resilience Engineering: Concepts and Precepts (Hollnagel et al 2006). The authors make clear in this book that Resilience Engineering has to do with the resilience of the organizations that design and operate engineered systems and not with the systems themselves The term **System Resilience** used in this article is primarily concerned with the techniques used to consider the resilience of engineered systems directly. To fully achieve this SE also needs to consider the resilience of those organizational and human systems which enable the life cycle of an engineered system. The techniques or design principles used to assess and improve the resilience of engineered systems across their life cycle are elaborated by Jackson and Ferris (2013).

Overview

Resilience is a relatively new term in the SE realm, appearing only in the 2006 time frame and becoming popularized in 2010. The recent application of “resilience” to engineered systems has led to confusion over its meaning and a proliferation of alternative definitions. (One expert claims that well over 100 unique definitions of resilience have appeared.) While the details of definitions will continue to be discussed and debated, the information here should provide a working understanding of the meaning and implementation of resilience, sufficient for a system engineer to effectively address it.

Definition

It is difficult to identify a single definition that – word for word – satisfies all. However, it is possible to gain general agreement of what is meant by resilience of engineered systems; viz., resilience is the ability to provide required capability in the face of adversity.

Scope of the Means

In applying this definition, one needs to consider the range of means by which resilience is achieved: The means of achieving resilience include avoiding, withstanding, and recovering from adversity. These may also be considered the fundamental objectives of resilience (Brtis and McEvilley 2019). Classically, resilience includes “withstanding” and “recovering” from adversity. For the purpose of engineered systems, “avoiding” adversity is considered a legitimate means of achieving resilience (Jackson and Ferris 2016). Also, it is believed that resilience should consider the system’s ability to “evolve and adapt” to future threats and unknown-unknowns.

Scope of the Adversity

Adversity is any condition that may degrade the desired capability of a system. We propose that the SE must consider all sources and types of adversity; e.g., from environmental sources, due to normal failure, as well as from opponents, friendlies and neutral parties. Adversity from human sources may be malicious or accidental. Adversities may be expected or not. Adversity may include "unknown unknowns." The techniques for achieving resilience discussed below are applicable to both hostile and non-hostile adversities in both civil and military domains. Non-hostile adversities will dominate in the civil domain and hostile adversities will predominate in the military domain.

Notably, a single incident may be the result of multiple adversities, such as a human error committed in the attempt to recover from another adversity.

Jackson and Ferris Taxonomy

Figure 1 depicts the loss and recovery of the functionality of a system. In the taxonomy proposed by Jackson and Ferris (2013) four attributes can lead to a resilient system and may possess four attributes: robustness, adaptability, tolerance, and integrity — and fourteen design techniques and 20 support techniques that can achieve these attributes. These four attributes are adapted from Hollnagel, Woods, and Leveson (2006), and the design techniques are extracted from Hollnagel et al. and are elaborated based on Jackson and Ferris (2013) for civil systems.

Other sources for example, DHS (2010) lists the following additional attributes: rapidly, affordability and learning capacity.

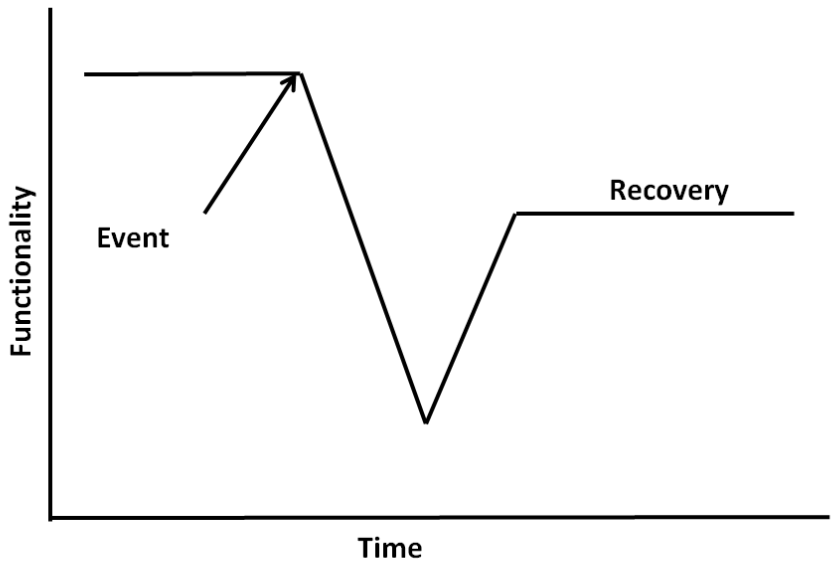


Figure 1. Disruption Diagram. (SEBoK Original)

The Robustness Attribute

Robustness is the attribute of a system that allows it to withstand a threat in the normal operating state. Resilience allows that the capacity of a system may be exceeded, forcing the system to rely on the remaining attributes to achieve recovery. The following design techniques tend to achieve robustness:

- absorption
- physical redundancy
- functional redundancy

The Adaptability Attribute

Adaptability is the attribute of a system that allows it to restructure itself in the face of a threat. Adaptability can apply to any phase of the event including detecting and avoiding the adversity and restructuring to return to normal operation. The following design techniques apply to the adaptability attribute:

- restructuring
- human in the loop
- complexity avoidance
- drift correction

The Tolerance Attribute

Tolerance is the attribute of a system that allows it to degrade gracefully following an encounter with adversity. The following design techniques apply to the tolerance attribute.

- modularity
- loose coupling
- neutral state
- reparability
- defense in depth

The Integrity Attribute

According to the (On-line Dictiobary 2019) integrity is the property of being whole or cohesive, all of which are terms used to describe systems. (Hitchins 2009) states that coherence is a property of systems. According to Adams et al. (2014) and Checkland (1999), another term used to describe wholeness is holism. In addition, the INCOSE Fellows state in Sillitto and Dori (2017) that systems are “whole” or “complete.”

The resilience principle that ensures that coherence is the internode interaction principle. This principle was identified by Jackson and Ferris (2013). The internode interaction principle calls for all nodes of a system to interact with other relevant nodes of a system. This interaction with all the other relevant nodes enables the system to sustain a viable architecture in the face of adversity. Coherence is built into the system prior to the utilization stage. The period prior to the utilization stage is when the adversities addressed by the system design are identified.

MITRE Taxonomy

Brits (2016) and Brits and McEvilley (2019) builds on the cyber resilience work of Bodeau and Graubart (2011) and proposes an objectives-based three layer taxonomy for thinking about resilience. The three layers include: 1) three fundamental objectives of resilience, 2) eleven means objectives of resilience, and, 3) 23 engineering techniques for achieving resilience.

The three fundamental objectives, which identify the intrinsic values of resilience, are:

- Avoid adversity
- Withstand adversity
- Recover from adversity

The eleven means objectives are not ends in themselves - as are the fundamental objective - but do tend to result in the achievement of the fundamental objectives: The means objectives are:

- adapt
- anticipate
- understand
- disaggregate
- prepare
- prevent
- continue
- constrain
- redeploy
- transform
- re-architect

The 23 engineering techniques that tend to achieve the fundamental objectives are:

- adaptive response
- analytic monitoring
- coordinated defense
- deception
- distribution
- detection avoidance
- diversification
- dynamic positioning
- dynamic representation
- effect tolerance
- non-persistence
- privilege restriction
- proliferation
- protection
- realignment
- reconfiguring
- redundancy
- replacement
- segmentation
- substantiated integrity
- substitution
- threat suppression
- unpredictability

The relationships between the three layers of this taxonomy are many-to-many relationships.

The Jackson and Ferris taxonomy comes from the civil resilience perspective and the MITRE taxonomy comes from the military perspective. Jackson and Brtis (2017) have shown that many of the techniques of the two taxonomies are equivalent and that some techniques are unique to each domain.

Techniques for Achieving Resilience

Techniques for achieving resilience have been identified for both the civil and military domains. Jackson and Ferris (2013) have identified techniques for the civil domain and Brtis (2016) has identified techniques for the military domain. There is overlap between these two sets and also some differences. Jackson and Brtis (2017) compare the techniques in the two domains and show their commonalities and their differences.

Techniques for the Civil Domain

34 techniques and support techniques for the civil domain described by Jackson and Ferris (2013) include both design and process techniques that will be used to define a system of interest in an effort to make it resilient. These techniques were extracted from many sources. Prominent among these sources is Hollnagel et al (2006). Other sources include Leveson (1995), Reason (1997), Perrow (1999), and Billings (1997). Some techniques were implied in case study reports, such as the 9/11 Commission report (2004) and the US-Canada Task Force report (2004) following the 2003 blackout. These techniques include very simple and well-known techniques as physical redundancy and more sophisticated techniques as loose coupling. Some of these techniques are domain dependent, such as loose coupling, which is important in the power distribution domain as discussed by Perrow (1999). These techniques will be the input to the state model of Jackson, Cook, and Ferris (2015) to determine the characteristics of a given system for a given threat. In the resilience literature the term technique is used to describe both scientifically accepted techniques and also heuristics, design rules determined from experience as described by Rechtin (1991). Jackson and Ferris (2013) showed that it is often necessary to invoke these techniques in combinations to best enhance resilience. This concept is called defense in depth. Pariès (2011) illustrates how defense in depth was used to achieve resilience in the 2009 ditching of US Airways Flight 1549. Uday and Marais (2015) apply the above techniques to the design of a system-of-systems. Henry and Ramirez-Marquez (2016) describe the state of the U.S. East Coast infrastructure in resilience terms following the impact of Hurricane Sandy in 2012. Bodeau and Graubert (2011) propose a framework for understanding and addressing cyber-resilience. They propose a taxonomy comprised of four goals, eight objectives, and fourteen cyber-resilience techniques. Many of these goals, objectives and practices can be applied to non-cyber resilience. Jackson and Ferris (2013) have collected 14 design techniques from various authoritative sources. These techniques are applicable primarily to civil systems including civil infrastructure, aviation, and power grids. In addition to the 14 design techniques, Jackson and Ferris (2013) also identify 20 support techniques that are narrower in scope than the above design techniques.

Techniques for the Military Domain

Brtis (2016), in the third level of his taxonomy discussed above identifies 23 engineering techniques for achieving resilience in the military domain. Jackson and Brtis (2017) have shown that many of the civil and military techniques are equivalent though some are unique to each domain.

The Resilience Process

Implementation of resilience in a system requires the execution of both analytic and holistic processes. In particular, the use of architecting with the associated heuristics is required. Inputs are the desired level of resilience and the characteristics of a threat or disruption. Outputs are the characteristics of the system, particularly the architectural characteristics and the nature of the elements (e.g., hardware, software, or humans). Artifacts depend on the domain of the system. For technological systems, specification and architectural descriptions will result. For enterprise systems, enterprise plans will result. Both analytic and holistic methods, including the techniques of architecting, are

required. Analytic methods determine required robustness. Holistic methods determine required adaptability, tolerance, and integrity. One pitfall is to depend on just a single technique to achieving resilience. The technique of defense in depth suggests that multiple techniques may be required to achieve resilience.

Resilience Requirements

Brtis and McEvilley (2019) investigated the content and structure needed to specify resilience requirements. The content of a resilience requirement flows almost directly from the definition of resilience: "the ability to deliver required capability in the face of adversity." Specifying resilience requires that several parameters be identified. The aggregation of these parameters can be considered to be a "resilience scenario," represented in Figure 2.

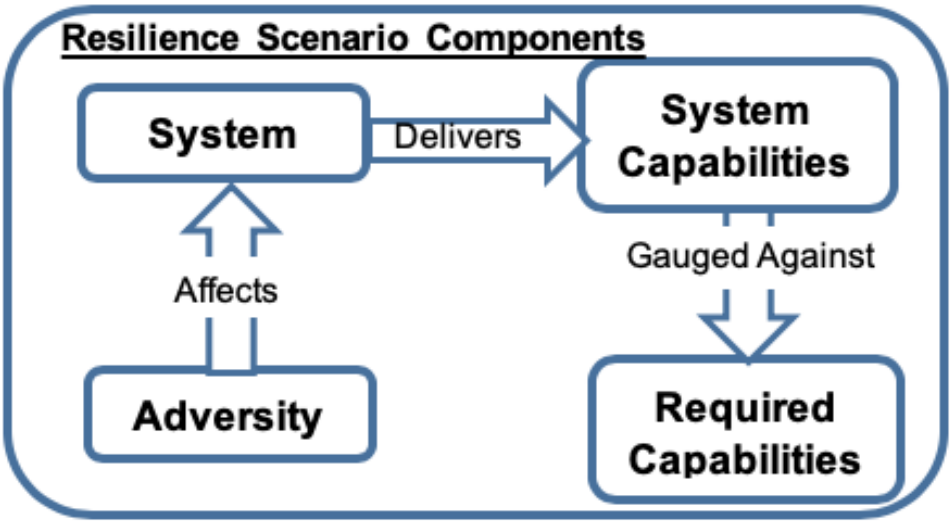


Figure 2. Relationships of resilience requirements components, comprising a resilience scenario. (Brtis and McEvilley 2019, used with permission)

The following must be known in order to specify a resilience requirement:

- The capability(s) of interest (note: a system may deliver several capabilities each of which may have different levels of resilience.).
  - The measure(s) (and units) of the capability(s).
  - The target value(s) (required amount) of the capability(s).
    - Note: there may be several salient levels of "required" capability. (e.g., nominal, degraded mode, minimum useful, objective, threshold, etc.).
- System modes of operation (e.g., operational, training, exercise, maintenance, update…)
- The adversity(s) being considered for this resilience scenario.
- The ways that the adversity(s) affect(s) the system and how the system reacts in terms of its ability to deliver capability.
- The timeframe of interest.
  - Note: An adversity and its affecting the system may be acute or chronic, single or multiple.
- The required resilience (performance) of the capability in the face of each identified resilience scenario (e.g., expected availability, maximum allowed degradation, maximum length of degradation, etc.).
  - Note there may be several "required" resilience goals (e.g., threshold, objective, As Resilient as Practicable (ARAP)).

Importantly, many of these parameters may vary over the timeframe of the scenario. Brtis and McEvilley (2019) provides a formal data structure for the above identified parameters.

Resilience requirements are unique because they are requirements about requirements. For example, the capabilities of interest can be functional requirements of the system. Resilience extends such requirements with a resilience scenario, adding environmental requirements (adversities) and performance requirements (the desired resilience).

Affordable Resilience

"Affordable Resilience" means to achieve an effective balance across affordability and technical attributes in order to adapt to changing conditions as well as to prepare for and avoid, withstand, and rapidly recover from disruption as required to satisfy the needs of multiple stakeholders throughout a system's life cycle. Technical attributes include robustness, flexibility, adaptability, tolerance, and integrity.

Note that the priority of resilience attributes for systems is typically domain-dependent-- for example, public transportation systems may emphasize affordable safety; electronic funds transfer systems may emphasize affordable cyber security; and unmanned space exploration systems may emphasize affordable survivability to withstand previously-unknown environments.

Life cycle considerations should address not only risks and adversities associated with known and unknown disruptions over time but also opportunities for seeking gain in known and unknown future environments. This often requires balancing the time value of money vs. the time value of resilience to achieve affordable resilience.

- For resilience attributes for engineered systems see Jackson and Ferris (2013).
- For engineering resilient systems see Neches and Madni (2013).
- For frameworks for affordability see Wheaton and Madni (2015)
- For four elements of the research strategy for SE Transformation see Boehm (2013)
- See section 10.9 Resilience Engineering in INCOSE (2015)
- See section 2.4 Quantifying Project Opportunity in Browning (2014)

System Description

System resilience is the ability of an engineered system to provide required capability in the face of adversity. Resilience in the realm of systems engineering involves identifying: 1) the capabilities that are required of the system, 2) the adverse conditions under which the system is required to deliver those capabilities, and 3) the systems engineering to ensure that the system can provide the required capabilities.

Put simply, resilience is achieved by a systems engineering focus on adverse conditions.

Resilience of Processes

It is important to recognize that processes are systems – in fact Systems Engineering is a system. Discussions relating to the resilience of such “process” systems include seven key resiliencies that successful sociotechnical systems intending to accomplish system engineering must have, Warfield (2008). Ashby’s Law of Requisite Variety and Pareto’s Law of Requisite Saliency are the most familiar. The scope and time of arrival of Contract Change Orders that require system engineering attention pose significant risk. Ones that occur during detailed design that affect the requirements baseline and system design baseline and occur faster than can be accommodated are particularly threatening.

Discipline Management

Most enterprises, both military and commercial, include organizations generally known as Advanced Design. These organizations are responsible for defining the architecture of a system at the very highest level of the system architecture. This architecture reflects the resilience techniques described in Jackson and Ferris (2013) and Brtis (2016) and the processes associated with that system. In many domains, such as fire protection, no such organization will exist. However, the system architecture will still need to be defined by the highest level of management in that

organization. In addition, some aspects of resilience will be established by government imposed requirements.

Discipline Relationships

Interactions

Outputs

The primary outputs of the resilience discipline are a subset of the principles described by Jackson and Ferris (2013) which have been determined to be appropriate for a given system, threat, and desired state of resilience as determined by the state-transition analysis described below. The processes requiring these outputs are the system design and system architecture processes.

Inputs

Inputs to the state model described in Jackson, Cook, and Ferris (2015) include (1) type of system of interest, (2) nature of threats to the system (earthquakes, terrorist threats, human error, etc.) (3) techniques for potential architectural design, and (4) predicted probability of success of individual techniques.

Dependencies

The techniques identified for the achieving resilience may also be used by other systems engineering areas of concern such as safety, reliability, human factors, availability, maintainability, human factors, security, and others. For example, the physical redundancy technique may help achieve resilience, reliability, and safety. Resilience design and analysis should be conducted in concert with the various ‘ilities. The goal being to create a system which -- from the beginning -- meets the requirements for resilience and other ‘ilities.

Discipline Standards

ASIS (2009) has published a standard pertaining to the resilience of organizational systems.

NIST 800-160 considers resilience of physical systems.

Personnel Considerations

Humans are important components of systems for which resilience is desired. This aspect is reflected in the human in the loop technique identified by Jackson and Ferris (2013). Decisions made by the humans are at the discretion of the humans in real time. Apollo 11 described by Eyles (2009) is a good example.

Metrics

Uday and Marais (2015) performed a survey of resilience metrics. Those identified include:

- Time duration of failure
- Time duration of recovery
- Ratio of performance recovery to performance loss
- A function of speed of recovery
- Performance before and after the disruption and recovery actions
- System importance measures

Jackson (2016) developed a metric to evaluate various systems in four domains: aviation, fire protection, rail, and power distribution, for the principles that were lacking in ten different case studies. The principles are from the set identified by Jackson and Ferris (2013) and are represented in the form of a histogram plotting principles against frequency of omission. The data in these gaps were taken from case studies in which the lack of principles was

inferred from recommendations by domain experts in the various cases cited.

Brtis (2016) surveyed and evaluated a number of potential resilience metrics and identified the following: [Note: This reference is going through approval for public release and should be referenceable by the end of July 2016.]

- Maximum outage period
- Maximum brownout period
- Maximum outage depth
- Expected value of capability: the probability-weighted average of capability delivered
- Threat resiliency (the time integrated ratio of the capability provided divided by the minimum needed capability)
- Expected availability of required capability (the likelihood that for a given adverse environment the required capability level will be available)
- Resilience levels (the ability to provide required capability in a hierarchy of increasingly difficult adversity)
- Cost to the opponent
- Cost-benefit to the opponent
- Resource resiliency (the degradation of capability that occurs as successive contributing assets are lost)

Brtis found that multiple metrics may be required, depending on the situation. However, if one had to select a single most effective metric for reflecting the meaning of resilience, it would be the expected availability of the required capability. Expected availability of the required capability is the probability-weighted sum of the availability summed across the scenarios under consideration. In its most basic form, this metric can be represented mathematically as:

where,

R = Resilience of the required capability (Cr);

n = the number of exhaustive and mutually exclusive adversity scenarios within a context (n can equal 1);

Pi = the probability of adversity scenario I;

Cr(t)_i = timewise availability of the required capability during scenario I; --- 0 if below the required level --- 1 if at or above the required value (Where circumstances dictate this may take on a more complex, non-binary function of time.);

T = length of the time of interest.

Models

The state-transition model described by Jackson et al (2015) describes a system in its various states before, during, and after an encounter with a threat. The model identifies seven different states as the system passes from a nominal operational state to minimally acceptable functional state as shown in the figure below. In addition, the model identifies 28 transition paths from state to state. To accomplish each transition the designer must invoke one or more of the 34 principles or support principles described by Jackson and Ferris (2013). The designs implied by these principles can then be entered into a simulation to determine the total effectiveness of each design.

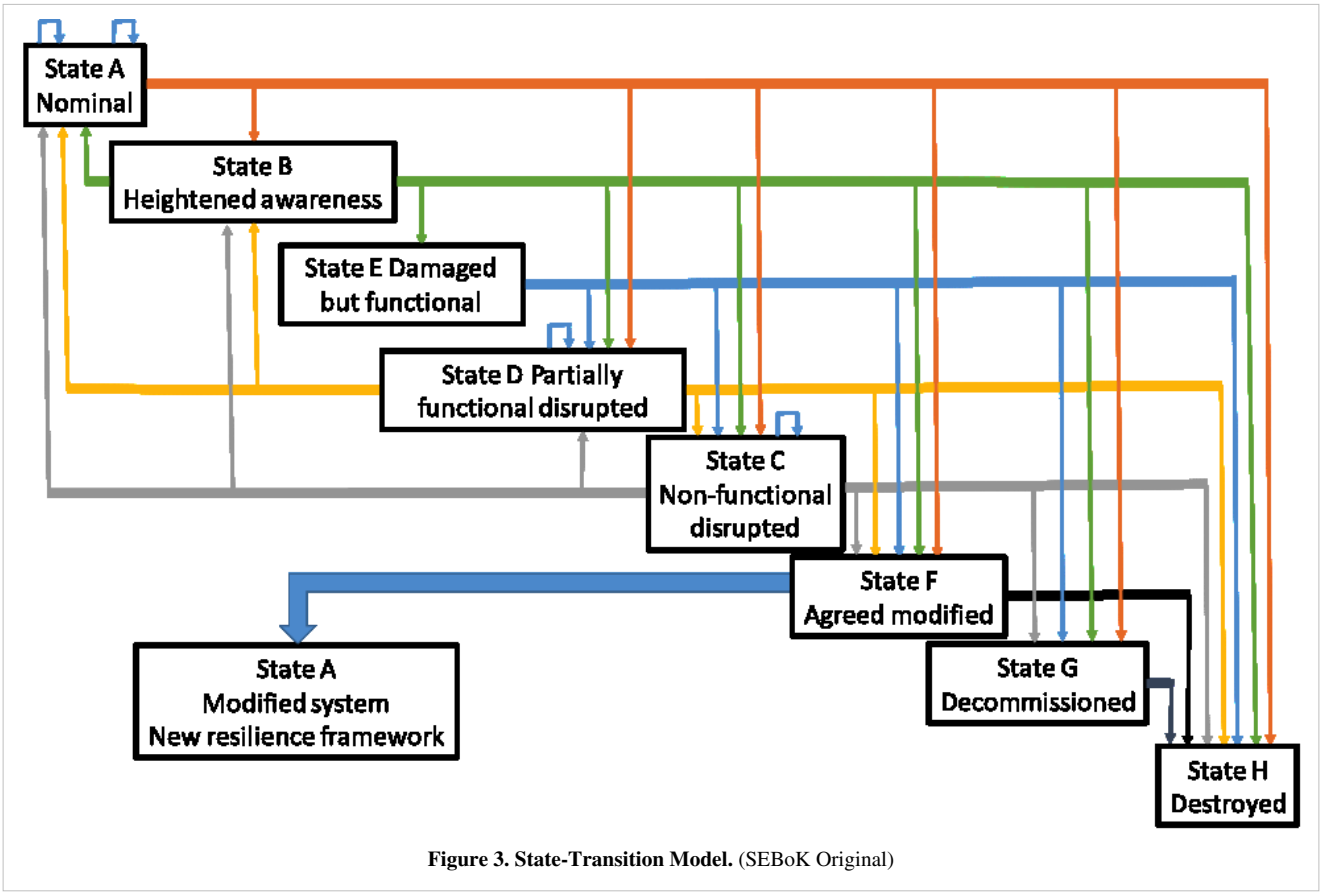


Figure 3. State-Transition Model. (SEBoK Original)

Tools

No tools dedicated to resilience have been identified.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Resilience is difficult to achieve for infrastructure systems because the nodes (cities, counties, states, and private entities) are reluctant to cooperate with each other. Another barrier to resilience is cost. For example, achieving redundancy in dams and levees can be prohibitively expensive. Other aspects, such as communicating on common frequencies, can be low or moderate cost; even there, cultural barriers have to be overcome for implementation.

References

Works Cited

9/11 Commission. (2004). 9/11 Commission Report.

Adams, K. M., Hester, P. T., Bradley J. M., Meyers, T. J., and Keating, C. B. (2014). "Systems Theory as the Foundation for Understanding Systems." Systems Engineering 17 (1):112-123.

ASIS International. (2009). Organizational Resilience: Security, Preparedness, and Continuity Management Systems--Requirements With Guidance for Use. Alexandria, VA, USA: ASIS International.

Billings, C. (1997). Aviation Automation: The Search for Human-Centered Approach. Mahwah, NJ: Lawrence Erlbaum Associates.

Bodeau, D. K, and Graubart, R. (2011). Cyber Resiliency Engineering Framework, MITRE Technical Report #110237, The MITRE Corporation.

Boehm, B. (PI), (2013). "Tradespace and Affordability – Phase 2 Final Technical Report", SERC-2013-TR-039-2, December 31 2013, Copyright © 2013 Stevens Institute of Technology, Systems Engineering Research Center, Published online at https://apps.dtic.mil/dtic/tr/fulltext/u2/a608178.pdf

Browning, T. R. (2014). "A Quantitative Framework for Managing Project Value, Risk, and Opportunity," in IEEE Transactions on Engineering Management, vol. 61, no. 4, pp. 583-598, Nov. 2014. doi: 10.1109/TEM.2014.2326986

Brtis, J. S. (2016). How to Think About Resilience, MITRE Technical Report, MITRE Corporation.

Brtis, J. S., and McEvilley, M. A. (2019). Systems Engineering for Resilience, MITRE Technical Report, The MITRE Corporation.

Checkland, P. 1999. Systems Thinking, Systems Practice. New York: John Wiley & Sons.

Hitchins, D. 2009. "What are the General Principles Applicable to Systems?" Insight, 59-63.

Hollnagel, E., D. Woods, and N. Leveson (eds). 2006. *Resilience Engineering: Concepts and Precepts.* Aldershot, UK: Ashgate Publishing Limited.

INCOSE (2015). Systems Engineering Handbook, a Guide for System Life Cycle Processes and Activities. San Diego, Wiley.

Jackson, S., & Ferris, T. (2013). Resilience Principles for Engineered Systems. Systems Engineering, 16(2), 152-164. doi:10.1002/sys.21228.

Jackson, S., Cook, S. C., & Ferris, T. (2015). A Generic State-Machine Model of System Resilience. Insight, 18.

Jackson, S. & Ferris, T. L. (2016). Proactive and Reactive Resilience: A Comparison of Perspectives.

Jackson, W. S. (2016). Evaluation of Resilience Principles for Engineered Systems. Unpublished PhD, University of South Australia, Adelaide, Australia.

Leveson, N. (1995). Safeware: System Safety and Computers. Reading, Massachusetts: Addison Wesley.

Madni, Azad,, & Jackson, S. (2009). Towards a conceptual framework for resilience engineering. Institute of Electrical and Electronics Engineers (IEEE) Systems Journal, 3(2), 181-191.

Neches, R. and Madni, A. M. (2013), Towards affordably adaptable and effective systems. Syst. Engin., 16: 224-234. doi:10.1002/sys.21234

On-line Dictiobary. 2019. "Definition of Integrity." Google.com, accessed 27 June.

Pariès, J. (2011). Lessons from the Hudson. In E. Hollnagel, J. Pariès, D. D. Woods & J. Wreathhall (Eds.), Resilience Engineering in Practice: A Guidebook. Farnham, Surrey: Ashgate Publishing Limited.

Perrow, C. (1999). Normal Accidents: Living With High Risk Technologies. Princeton, NJ: Princeton University Press.

Reason, J. (1997). Managing the Risks of Organisational Accidents. Aldershot, UK: Ashgate Publishing Limited.

Rechtin, E. (1991). Systems Architecting: Creating and Building Complex Systems. Englewood Cliffs, NJ: CRC Press.

Sillitto, H. G., and Dori, D.. 2017. "Defining 'System': A Comprehensive Approach." IS 2017, Adelaide, Australia.

US-Canada Power System Outage Task Force. (2004). Final Report on the August 14, 2003 Blackout in the United States and Canada: Causes and Recommendations. Washington-Ottawa.

Uday, P., & Morais, K. (2015). Designing Resilient Systems-of-Systems: A Survey of Metrics, Methods, and Challenges. Systems Engineering, 18(5), 491-510.

Warfield, J. N. (2008). "A Challenge for Systems Engineers: To Evolve Toward Systems Science" INCOSE INSIGHT, Volume 11, Issue 1, January 2008.

Wheaton, M. J. & Madni, A. M. (2015). "Resiliency and Affordability Attributes in a System Integration Tradespace", AIAA SPACE 2015 Conference and Exposition, 31 Aug-2 Sep 2015, Pasadena California, Published online at https://doi.org/10.2514/6.2015-4434.

Primary References

Hollnagel, E., Woods, D. D., & Leveson, N. (Eds.). (2006). Resilience Engineering: Concepts and Precepts. Aldershot, UK: Ashgate Publishing Limited.

Jackson, S., & Ferris, T. (2013). Resilience Principles for Engineered Systems. Systems Engineering, 16(2), 152-164.

Jackson, S., Cook, S. C., & Ferris, T. (2015). Towards a Method to Describe Resilience to Assist in System Specification. Paper presented at the INCOSE Systems 2015.

Jackson, S.: Principles for Resilient Design - A Guide for Understanding and Implementation. Available at https://www.irgc.org/irgc-resource-guide-on-resilience Accessed 18th August 2016

Madni, Azad,, & Jackson, S. (2009). Towards a conceptual framework for resilience engineering. Institute of Electrical and Electronics Engineers (IEEE) Systems Journal, 3(2), 181-191.

Additional References

ASIS International. 2009. Organisational Resilience: Security, Preparedness, and Continuity Management Systems--Requirements With Guidance for Use. Alexandria, VA, USA: ASIS International.

Billings, Charles. 1997. Aviation Automation: The Search for Human-Centered Approach. Mahwah, NJ: Lawrence Erlbaum Associates.

Bodeu, D. K., and R. Graubart. 2011. Cyber Resiliency Engineering Framework.

Eyles, Don. 2009. "1202 Computer Error Almost Aborted Lunar Landing." Massachusetts Intitute of Technology, MIT News, accessed 6 April http://njnnetwork.com/2009/07/1202-computer-error-almost-aborted-lunar-landing/

Henry, Devanandham, and Emmanual Ramirez-Marquez. 2016. "On the Impacts of Power Outages during Hurrican Sandy -- A Resilience Based Analysis." Systems Engineering 19 (1):59-75.

OED. 1973. The Shorter Oxford English Dictionary on Historical Principles. edited by C. T. Onions. Oxford: Oxford Univeristy Press. Original edition, 1933.

Pariès, Jean. 2011. "Lessons from the Hudson." In Resilience Engineering in Practice: A Guidebook, edited by Erik Hollnagel, Jean Pariès, David D. Woods and John Wreathhall, 9-27. Farnham, Surrey: Ashgate Publishing Limited.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Manufacturability and Producibility

Lead Authors: *Dick Fairley, Kevin Forsberg*, ***Contributing Authors:*** *Paul Phister, Alice Squires, Richard Turner*

Manufacturability and producibility is an engineering specialty. The machines and processes used to build a system must be architected and designed. A systems engineering approach to manufacturing and production is necessary because manufacturing equipment and processes can sometimes cost more than the system being built (Maier and Rechtin 2002). Manufacturability and producibility can be a discriminator between competing system solution concepts and therefore must be considered early in the study period, as well as during the maturing of the final design solution.

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

The system being built might be intended to be one-of-a-kind, or to be reproduced multiple times. The manufacturing system differs for each of these situations and is tied to the type of system being built. For example, the manufacture of a single-board computer would be vastly different from the manufacture of an automobile. Production involves the repeated building of the designed system. Multiple production cycles require the consideration of production machine maintenance and downtime.

Manufacturing and production engineering involve similar systems engineering processes specifically tailored to the building of the system. Manufacturability and producibility are the key attributes of a system that determine the ease of manufacturing and production. While manufacturability is simply the ease of manufacture, producibility also encompasses other dimensions of the production task, including packaging and shipping. Both these attributes can be improved by incorporating proper design decisions that take into account the entire system life cycle (Blanchard and Fabrycky 2005).

System Description

Information to be supplied at a later date.

Discipline Management

Information to be supplied at a later date.

Discipline Relationships

Interactions

Information to be supplied at a later date.

Dependencies

Information to be supplied at a later date.

Discipline Standards

Information to be supplied at a later date.

Personnel Considerations

Information to be supplied at a later date.

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be supplied at a later date.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

References

Works Cited

Maier, M., and E. Rechtin. 2002. *The Art of Systems Architecting,* 2nd ed. Boca Raton, FL, USA: CRC Press.

Blanchard, B.S., and W.J. Fabrycky. 2005. *Systems Engineering and Analysis,* 4th ed. Prentice-Hall International Series in Industrial and Systems Engineering. Englwood Cliffs, NJ, USA: Prentice-Hall.

Primary References

None.

Additional References

Anderson, D. 2010. *Design for Manufacturability & Concurrent Engineering; How to Design for Low Cost, Design in High Quality, Design for Lean Manufacture, and Design Quickly for Fast Production.* Cambria, CA, USA: CIM Press.

Boothroyd, G., P. Dewhurst, and W. Knight. 2010. *Product Design for Manufacture and Assembly.* 3rd Ed. Boca Raton, FL, USA: CRC Press.

Bralla, J. 1998. *Design for Manufacturability Handbook.* New York, NY, USA: McGraw Hill Professional.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Affordability

Lead Author: *Paul Phister*, ***Contributing Author:*** *Ray Madachy*

A system is affordable to the degree that system performance, cost, and schedule constraints are balanced over the system life, while mission needs are satisfied in concert with strategic investment and organizational needs (INCOSE 2011). Design for affordability is the practice of considering affordability as a design characteristic or constraint.

Increasing competitive pressures and the scarcity of resources demand that systems engineering (SE) improve affordability. Several recent initiatives have made affordability their top technical priority. They also call for a high priority to be placed on research into techniques — namely, improved systems autonomy and human performance augmentation — that promise to reduce labor costs, provide more efficient equipment to reduce supply costs, and create adaptable systems whose useful lifetime is extended cost-effectively.

However, methods for cost and schedule estimation have not changed significantly to address these new challenges and opportunities. There is a clear need for:

- new methods to analyze tradeoffs between cost, schedule, effectiveness, and resilience;
- new methods to adjust priorities and deliverables to meet budgets and schedules; and
- more affordable systems development processes.

All of this must be accomplished in the context of the rapid changes underway in technology, competition, operational concepts, and workforce characteristics.

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

Historically, cost and schedule estimation has been decoupled from technical SE tradeoff analyses and decision reviews. Most models and tools focus on evaluating either cost-schedule performance or technical performance, but not the tradeoffs between the two. Meanwhile, organizations and their systems engineers often focus on affordability to minimize acquisition costs. They are then drawn into the easiest-first approaches that yield early successes, at the price of being stuck with brittle, expensive-to-change architectures that increase technical debt and life cycle costs.

Two indications that the need for change is being recognized in systems engineering are that the *INCOSE SE Handbook* now includes affordability as one of the criteria for evaluating requirements (INCOSE 2011) and that

there is a trend in SE towards stronger focus on maintainability, flexibility, and evolution (Blanchard, Verma, and Peterson 1995).

There are pitfalls for the unwary. Autonomous systems experience several hazardous failure modes, including:

- **system instability due to positive feedback —** where an agent senses a parameter reaching a control limit and gives the system a strong push in the other direction, causing the system to rapidly approach the other control limit, causing the agent (or another) to give it an even stronger push in the original direction, and so on
- **self-modifying autonomous agents which fail** after several self-modifications **—** the failures are difficult to debug because the agent's state has been changing
- **autonomous agents performing weakly at commonsense reasoning** about system control decisions by human operators, and so tend to reach incorrect conclusions and make incorrect decisions about controlling the system
- **multiple agents making contradictory decisions** about controlling the system, and lacking the ability to understand the contradiction or to negotiate a solution to resolve it

Modularization of the system's architecture around its most frequent sources of change (Parnas 1979) is a key SE principle for affordability. This is because when changes are needed, their side effects are contained in a single systems element, rather than rippling across the entire system.

This approach creates the need for three further improvements:

- refocusing the system requirements, not only on a snapshot of current needs, but also on the most likely sources of requirements change, or evolution requirements;
- monitoring and acquiring knowledge about the most frequent sources of change to better identify requirements for evolution; and
- evaluating the system's proposed architecture to assess how well it supports the evolution requirements, as well as the initial snapshot requirements.

This approach can be extended to produce several new practices. Systems engineers can:

- identify the commonalities and variability across the families of products or product lines, and develop architectures for creating (and evolving) the common elements *once* with plug-compatible interfaces for inserting the variable elements (Boehm, Lane, and Madachy 2010);
- extrapolate principles for service-oriented system elements that are characterized by their inputs, outputs, and assumptions, and that can easily be composed into systems in which the sources of change were not anticipated; and
- develop classes of smart or autonomous systems whose many sensors identify needed changes, and whose autonomous agents determine and effect those changes in microseconds, or much more rapidly than humans can, reducing not only reaction time, but also the amount of human labor needed to operate the systems, thus improving affordability.

System Description

Information to be supplied at a later date.

Discipline Management

Information to be supplied at a later date.

Discipline Relationships

Interactions

Information to be supplied at a later date.

Dependencies

Information to be supplied at a later date.

Discipline Standards

Information to be supplied at a later date.

Personnel Considerations

Autonomous systems need human supervision, and the humans involved require better methods for trend analysis and visualization of trends (especially undesired ones).

There is also the need, with autonomous systems, to extend the focus from life cycle costs to total ownership costs, which encompass the costs of failures, including losses in sales, profits, mission effectiveness, or human quality of life. This creates a further need to evaluate affordability in light of the value added by the system under consideration. In principle, this involves evaluating the system's total cost of ownership with respect to its mission effectiveness and resilience across a number of operational scenarios. However, determining the appropriate scenarios and their relative importance is not easy, particularly for multi-mission systems of systems. Often, the best that can be done involves a mix of scenario evaluation and evaluation of general system attributes, such as cost, schedule, performance, and so on.

As for these system attributes, different success-critical stakeholders will have different preferences, or utility functions, for a given attribute. This makes converging on a mutually satisfactory choice among the candidate system solutions a difficult challenge involving the resolution of the multi-criteria decision analysis (MCDA) problem among the stakeholders (Boehm and Jain 2006). This is a well-known problem with several paradoxes, such as Arrow's impossibility theorem that describes the inability to guarantee a mutually optimal solution among several stakeholders, and several paradoxes in stakeholder preference aggregation in which different voting procedures produce different winning solutions. Still, groups of stakeholders need to make decisions, and various negotiation support systems enable people to better understand each other's utility functions and to arrive at mutually satisfactory decisions, in which no one gets everything that they want, but everyone is at least as well off as they are with the current system.

Also see System Analysis for considerations of cost and affordability in the technical design space.

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be supplied at a later date.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

Primary References

Works Cited

Blanchard, B., D. Verma, and E. Peterson. 1995. *Maintainability: A Key to Effective Serviceability and Maintenance Management.* New York, NY, USA: Wiley and Sons.

Boehm, B., J. Lane, and R. Madachy. 2010. "Valuing system flexibility via total ownership cost analysis." Proceedings of the NDIA SE Conference, October 2010, San Diego, CA, USA.

Boehm, B., and A. Jain. 2006. "A value-based theory of systems engineering." Proceedings of the International Council on Systems Engineering (INCOSE) International Symposium (IS), July 9-13, 2006, Orlando, FL, USA.

INCOSE. 2012. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2. p. 79.

Parnas, D.L. 1979. "Designing Software for Ease of Extension and Contraction." *IEEE Transactions on Software Engineering*. 5 (2): 128-138.

Primary References

INCOSE. 2012. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2. p. 79.

Blanchard, B., D. Verma, and E. Peterson. 1995. *Maintainability: A Key to Effective Serviceability and Maintenance Management.* New York, NY, USA: Wiley and Sons.

Parnas, D.L. 1979. "Designing Software for Ease of Extension and Contraction." *IEEE Transactions on Software Engineering*. 5 (2): 128-138.

Additional References

Kobren, Bill. 2011. "Supportability as an affordability enabler: A critical fourth element of acquisition success across the system life cycle." *Defense AT&L: Better Buying Power.* Accessed August 28, 2012. Available online at: http://www.dau.mil/pubscats/ATL%20Docs/Sep-Oct11/Kobren.pdf.

Myers, S.E., P.P. Pandolfini, J.F. Keane, O. Younossi, J.K. Roth, M.J. Clark, D.A. Lehman, and J.A. Dechoretz. 2000. "Evaluating affordability initiatives." *Johns Hopkins APL Tech. Dig.* 21 (3): 426–437.

< Previous Article | Parent Article | Next Article >

SEBoK v. 2.2, released 15 May 2020

Environmental Engineering

Lead Authors: *Paul Phister, David Olwell*

Environmental engineering addresses four issues that arise in system design and operation. They includeare: (1) design for a given operating environment, (2) environmental impact, (3) green design, and (4) compliance with environmental regulations.

Please note that not all of the generic below sections have mature content at this time. Anyone wishing to offer content suggestions should contact the SEBoK Editors in the usual ways.

Overview

A system is designed for a particular operating environment. Product systems, in particular, routinely consider conditions of temperature and humidity. Depending on the product, other environmental conditions may need to be considered, including UV exposure, radiation, magnetic forces, vibration, and others. The allowable range of these conditions must be specified in the requirements for the system.

Requirements

The general principles for writing requirements also apply to specifying the operating environment for a system and its elements. Requirements are often written to require compliance with a set of standards.

System Description

Information to be supplied at a later date.

Discipline Management

Many countries require assessment of environmental impact of large projects before regulatory approval is given. The assessment is documented in an environmental impact statement (EIS). In the United States, a complex project can require an EIS that greatly adds to the cost, schedule, and risk of the project.

Scope

In the U.S., the process in Figure 1 is followed. A proposal is prepared prior to a project being funded. The regulator examines the proposal. If it falls into an excluded category, no further action is taken. If not, an environmental assessment is made. If that assessment determines a finding of no significant impact (FONSI), no further action is taken. In all other cases, an environmental impact statement is required.

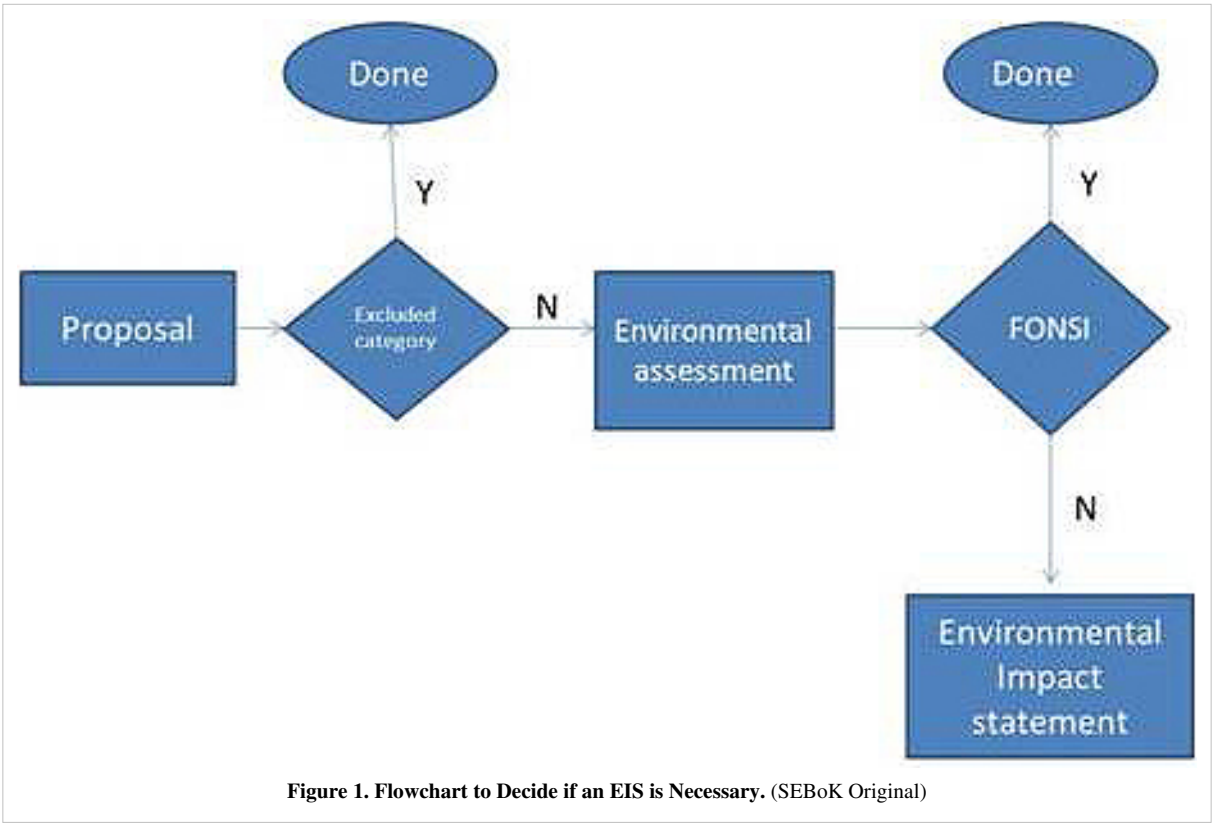


Figure 1. Flowchart to Decide if an EIS is Necessary. (SEBoK Original)

Preparation of an EIS is a resource significant task. Bregman (2000) and Kreske (1996) provide accessible overviews of the process. Lee and Lin (2000) provide a handbook of environmental engineering calculations to aid in the technical submission. Numerous firms offer consulting services.

Legal References

Basic references in the U.S. include the National Environmental Policy Act of 1969 and its implementing regulations (NEPA 1969) and the European Ccommission directive (EC 1985). State and local regulations can be extensive; Burby and Paterson (1993) discuss improving compliance.

Cost and Schedule Implications

Depending on the scale of the project, the preparation of an EIS can take years and cost millions. For example, the EIS for the Honolulu light rail project took four years and cost $156M (Hill 2011). While a project may proceed even if the EIS finds a negative impact, opponents to a project may use the EIS process to delay a project. A common tactic is to claim the EIS was not complete in that it omitted some environmental impacts. Eccleston (2000) provides a guide to planning for EIS.

Energy Efficiency

There is a large amount of literature that has been published about design for energy efficiency. Lovins (2010) offers ten design principles. He also provides case studies (Lovins et al. 2011). Intel (2011) provides guidance for improving the energy efficiency of its computer chips. A great deal of information is also available in regard to the efficient design of structures; DOE (2011) provides a good overview.

Increased energy efficiency can significantly reduce total life cycle cost for a system. For example, the Toyota Prius was found to have the lowest life cycle cost for 60,000 miles, three years despite having a higher initial purchase price (Brown 2011).

Carbon Footprint

Increased attention is being paid to the emission of carbon dioxide. BSI British Standards offers a specification for assessing life cycle greenhouse emissions for goods and services (BSI 2011).

Sustainability

Graedel and Allenby (2009), Maydl (2004), Stasinopoulos (2009), Meryman (2004), and Lockton and Harrison (2008) discuss design for sustainability. Sustainability is often discussed in the context of the UN report on Our Common Future (WCED 1987) and the Rio Declaration (UN 1992).

Discipline Relationships

An enterprise must attend to compliance with the various environmental regulations. Dechant et al. (1994) provide the example of a company in which 17% of every sales dollar goes toward compliance activities. They discuss gaining a competitive advantage through better compliance. Gupta (1995) studies how compliance can improve the operations function. Berry (1998) and Nash (2001) discuss methods for environmental management by the enterprise.

Interactions

Information to be supplied at a later date.

Dependencies

ISO14001 sets the standards for organization to comply with environmental regulations. Kwon and Seo (2002) discuss this in a Korean context, and Whitelaw (2004) presents a handbook on implementing ISO14001.

Discipline Standards

Depending on the product being developed, standards may exist for operating conditions. For example, ISO 9241-6 specifies the office environment for a video display terminal. Military equipment may be required to meet MILSTD 810G standard (DoD 2014) in the US, or DEF STAN 00-35 in the UK (MoD 2006).

The U.S. Federal Aviation Administration publishes a list of EIS best practices (FAA 2002).

The U.S. Environmental Protection Agency (EPA) defines green engineering as: the design, commercialization, and use of processes and products, which are feasible and economical, while minimizing (1) generation of pollution at the source and (2) risk to human health and the environment (EPA 2011). Green engineering embraces the concept that decisions to protect human health and the environment can have the greatest impact and cost effectiveness when applied early to the design and development phase of a process or product.

The EPA (2011) offers the following principles of green engineering:

- Engineer processes and products holistically, use systems analysis, and integrate environmental impact assessment tools.
- Conserve and improve natural ecosystems while protecting human health and well-being.
- Use life-cycle thinking in all engineering activities.
- Ensure that all material and energy inputs and outputs are as inherently safe and benign as possible.
- Minimize depletion of natural resources.
- Strive to prevent waste.
- Develop and apply engineering solutions, while being cognizant of local geography, aspirations, and cultures.
- Create engineering solutions beyond current or dominant technologies; additionally, improve, innovate, and invent (technologies) to achieve sustainability.
- Actively engage communities and stakeholders in development of engineering solutions.

Personnel Considerations

Information to be supplied at a later date.

Metrics

Information to be supplied at a later date.

Models

Information to be supplied at a later date.

Tools

Information to be supplied at a later date.

Practical Considerations

Pitfalls

Information to be provided at a later date.

Proven Practices

Information to be provided at a later date.

Other Considerations

Information to be provided at a later date.

References

Works Cited

Berry, MA. 1998. "Proactive corporate environmental management: Aa new industrial revolution." *The Academy of Management Executive*, 12 (2): 38-50.

Bregman, J.I. 2000. *Environmental Impact Statements*, 2nd ed. Boca Raton, FL, USA: CRC Press.

Brown, C. 2011 "The Green Fleet Price Tag." Business Fleet. Available: http://www.businessfleet.com/Article/Story/2011/07/The-Green-Fleet-Price-Tag.aspx.

BSI. 2011. "Specification for the assessment of the life cycle greenhouse gas emissions of goods and service, PAS 2050:2011." London, UK: British Standards Institution (BSI). Available: http://shop.bsigroup.com/en/forms/PASs/PAS-2050.

Burby, R.J., and R.G. Paterson. 1993. "Improving compliance with state environmental regulations." *Journal of Policy Analysis and Management*, 12(4): 753–772.

Dechant, K., B. Altman, R.M. Downing, and T. Keeney. 1994. "Environmental lLeadership: From cCompliance to cCompetitive aAdvantage." *Academy of Management Executive*, 8(3): 7.

DoD. 2014. *Department of Defense Test Method Standard: Environmental Engineering Considerations and Laboratory Tests*, MIL-STD-810G Change Notice 1. Washington, DC, USA: US Army Test and Evaluation Command, US Department of Defense (DoD). Accessed November 4, 2014. Available: http://www.atec.army.mil/publications/Mil-Std-810G/MIL-STD-810G%20CN1.pdf.

Eccleston, C. 2000. *Environmental Impact Statements: A Comprehensive Guide to Project and Strategic Planning*. New York, NY, USA: Wiley.

EPA. 2011. "Green Engineering. Environmental Protection Agency (EPA)." Washington, D.C., USA: United States Environmental Protection Agency. Available: http://www.epa.gov/oppt/greenengineering/.

EC. 1985. "Council Directive of 27 June 1985 on the assessment of the effects of certain public and private projects on the environment (85/337/EEC)." European Commission (EC). Available: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:1985L0337:20090625:EN:PDF.

FAA. 2002. "Best Practices for Environmental Impact Statement (EIS) Management." Federal Aviation Administration (FAA). Available: http://www.faa.gov/airports/environmental/eis_best_practices/?sect=intro.

Graedel, T.E., and B.R. Allenby. 2009. *Industrial Ecology and Sustainable Engineering*. Upper Saddle River, NJ, USA: Prentice Hall.

Gupta, M.C. 1995. "Environmental management and its impact on the operations function." *International Journal of Operations and Production Management*, 15 (8): 34-51.

Hill, T. 2011. "Honolulu Rail's nNext sStop?" *Honolulu Magazine*. July 2011.

Intel. 2011. "Energy Efficiency." Intel Corporation. Accessed: August 29, 2012. Available: http://www.intel.com/intel/other/ehs/product_ecology/energy.htm.

Kreske, D.L. 1996. *Environmental Iimpact Sstatements: Aa Ppractical Gguide for Aagencies, Ccitizens, and Cconsultants*. New York, NY: Wiley.

Kwon, D.M., and M.S. Seo. 2002. "A study of compliance with environmental regulations of ISO 14001 certified companies in Korea." *Journal of Environmental Management*. 65 (4): 347-353.

Lee, C.C., and S.D. Lin. 2000. *Handbook of Environmental Engineering Calculations*. New York, NY, USA: McGraw Hill Professional.

Lockton, D., and D. Harrison. 2008. "Making the user more efficient: Design for sustainable behaviour." *International Journal of Sustainable Engineering*. 1 (1): 3-8.

Lovins, A. 2010. "Factor Ten Engineering Design Principles," version 1.0. Available: http:// www. rmi. org/ Knowledge-Center/Library/2010-10_10xEPrinciples.

Lovins, A., et al. 2011. "Case Studies." Available: http://move.rmi.org/markets-in-motion/case-studies/.

Maydl, Peter. 2004. "Sustainable eEngineering: State-of-the-aArt and pProspects." *Structural Engineering International*. 14 (3): 176-180.

Meryman, H. 2004. "Sustainable eEngineering uUsing sSpecifications to mMake it hHappen." *Structural Engineering International*. 14 (3).

MoD. 2006. *Standard 00-35, Environmental Handbook for Defence Materiel (Part 3) Environmental Test Methods*. London, England, UK: UK Ministry of Defence (MoD). Available: http:// www. everyspec. com/ DEF-STAN/ download.php?spec=DEFSTAN00-35_I4.029214.pdf.

Nash, J. 2001. *Regulating Ffrom the Iinside: Ccan Eenvironmental Mmanagement Ssystems Aachieve Ppolicy Ggoals?* Washington, DC, USA: Resources for the Future Press.

NEPA. 1969. *42 USC 4321-4347. National Environmental Policy Act (NEPA)*. Accessed January 15, 2012. Available: http://ceq.hss.doe.gov/nepa/regs/nepa/nepaeqia.htm.

Stasinopoulos, P. 2009. *Whole Ssystem Ddesign: Aan Iintegrated Aapproach to Ssustainable Eengineering*. London, UK: Routledge.

UN. 1992. "Rio Declaration on Environment and Development." United Nations (UN). Available: http:// www. unep.org/Documents.Multilingual/Default.asp?documentid=78&articleid=1163.

Whitelaw, K. 2004. *ISO 14001: Environmental Systems Handbook*, 2nd ed. Oxford, UK: Elsevier.

WCED. 1987. "Our Common Future. World Commission on Economic Development (WCED)." Available: http:// www.un-documents.net/wced-ocf.htm.

Primary References

Bregman, J.I. 2000. *Environmental Impact Statements*, 2nd ed. Boca Raton, FL, USA: CRC Press.

Graedel, T.E., and B.R. Allenby. 2009. *Industrial Ecology and Sustainable Engineering*. Upper Saddle River, NJ, USA: Prentice Hall.

Lee, C.C., and S.D. Lin. 2000. *Handbook of Environmental Engineering Calculations*. New York, NY, USA: McGraw Hill Professional.

Whitelaw, K. 2004. *ISO 14001: Environmental Systems Handbook*, 2nd ed. Oxford, UK: Elsevier.

Additional References

None.

< Previous Article | Parent Article | Next Article (Part 7) >

SEBoK v. 2.2, released 15 May 2020

Article Sources and Contributors

Letter from the Editor *Source*: https://www.sebokwiki.org/d/index.php?oldid=59155 *Contributors*: Apyster, Bkcase, Cnielsen, Dholwell, Eleach, Kguillemette, Mhaas, Radcock, Rcloutier, Smenck2, Wikiexpert

BKCASE Governance and Editorial Board *Source*: https://www.sebokwiki.org/d/index.php?oldid=57626 *Contributors*: Apyster, Bkcase, Cnielsen, Dhenry, Kguillemette, Mhaas, Radcock, Rcloutier, Smenck2, WikiWorks, WikiWorks753

Acknowledgements and Release History *Source*: https://www.sebokwiki.org/d/index.php?oldid=59143 *Contributors*: Apyster, Asquires, Bkcase, Cnielsen, Ddori, Dhenry, Dholwell, Eleach, Janthony, Jgercken, Kguillemette, Mhaas, Nicole.hutchison, Radcock, Rcloutier, Smenck2, Smurawski, Wikiexpert

Cite the SEBoK *Source*: https://www.sebokwiki.org/d/index.php?oldid=59010 *Contributors*: Apyster, Bkcase, Cnielsen, Dholwell, Kguillemette, Smenck2

Bkcase Wiki:Copyright *Source*: https://www.sebokwiki.org/d/index.php?oldid=58407 *Contributors*: Apyster, Bkcase, Cnielsen, Dhenry, Kguillemette, Radcock, Smenck2, Wikiexpert

Related Disciplines *Source*: https://www.sebokwiki.org/d/index.php?oldid=58794 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Kguillemette, Mhaas, Radcock, Rmalove, SYSIND11, Smenck2, Thilburn, WikiWorks, Wikiexpert, Zamoses

Systems Engineering and Software Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58955 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Mhaas, Radcock, Rmadachy, Rmalove, Skmackin, Smurawski, Thilburn, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

Software Engineering in the Systems Engineering Life Cycle *Source*: https://www.sebokwiki.org/d/index.php?oldid=58845 *Contributors*: Asquires, Bkcase, Mhaas, Radcock, Thilburn, WikiWorks, WikiWorks753

The Nature of Software *Source*: https://www.sebokwiki.org/d/index.php?oldid=59031 *Contributors*: Apyster, Asquires, Bkcase, Dhenry, Dholwell, Jgercken, Mhaas, Rmalove, Thilburn, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

An Overview of the SWEBOK Guide *Source*: https://www.sebokwiki.org/d/index.php?oldid=58268 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Hdavidz, Jgercken, Kguillemette, Mhaas, Rmalove, Smenck2, Thilburn, WikiWorks753, Wikiexpert, Zamoses

Key Points a Systems Engineer Needs to Know about Software Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58454 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Eleach, Jgercken, Kguillemette, Mhaas, Rmalove, Smenck2, Smurawski, Thilburn, WikiWorks753, Wikiexpert, Zamoses

Software Engineering Features - Models, Methods, Tools, Standards, and Metrics *Source*: https://www.sebokwiki.org/d/index.php?oldid=58969 *Contributors*: Bkcase, Mhaas, Thilburn, WikiWorks

Systems Engineering and Project Management *Source*: https://www.sebokwiki.org/d/index.php?oldid=58802 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Kguillemette, Mhaas, Rturner, Skmackin, Smenck2, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

The Nature of Project Management *Source*: https://www.sebokwiki.org/d/index.php?oldid=59034 *Contributors*: Apyster, Asquires, Bkcase, Dhenry, Dholwell, Hdavidz, Jgercken, Kguillemette, Mhaas, Rmalove, Rturner, Smenck2, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

An Overview of the PMBOK® Guide *Source*: https://www.sebokwiki.org/d/index.php?oldid=58239 *Contributors*: Asquires, Bkcase, Dhenry, Dholwell, Kguillemette, Mhaas, Rmalove, Smenck2, Wikiexpert

Relationships between Systems Engineering and Project Management *Source*: https://www.sebokwiki.org/d/index.php?oldid=58799 *Contributors*: Apyster, Asquires, Bkcase, Dhenry, Dholwell, Eleach, Jgercken, Kguillemette, Mhaas, Rturner, Smenck2, Smurawski, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

The Influence of Project Structure and Governance on Systems Engineering and Project Management Relationships *Source*: https://www.sebokwiki.org/d/index.php?oldid=59023 *Contributors*: Asquires, Bkcase, Dhenry, Dholwell, Eleach, Kguillemette, Mhaas, Radcock, Smenck2, Smurawski, WikiWorks, WikiWorks753, Wikiexpert

Procurement and Acquisition *Source*: https://www.sebokwiki.org/d/index.php?oldid=58528 *Contributors*: Apyster, Asquires, Bkcase, Cnielsen, Dfairley, Dhenry, Dholwell, Jgercken, Kguillemette, Mhaas, Rmalove, Skmackin, Smenck2, Smurawski, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

Systems Engineering and Industrial Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58846 *Contributors*: Apyster, Bkcase, Dhenry, Dholwell, Eleach, Hsillitto, Kguillemette, Mhaas, SYSIND11, Smenck2, WikiWorks, WikiWorks753, Wikiexpert

Systems Engineering and Specialty Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58963 *Contributors*: Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Mhaas, Nicole.hutchison, Phisterp, Radcock, Rmalove, Rturner, S.Jackson, Skmackin, Smenck2, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

Reliability, Availability, and Maintainability *Source*: https://www.sebokwiki.org/d/index.php?oldid=59217 *Contributors*: Apyster, Asquires, Bkcase, Dhenry, Dholwell, Jgercken, Mhaas, Phisterp, Rmalove, Rturner, Skmackin, Smenck2, Smurawski, WikiWorks, Wikiexpert, Zamoses

Human Systems Integration *Source*: https://www.sebokwiki.org/d/index.php?oldid=58545 *Contributors*: Apyster, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Mhaas, Phisterp, Radcock, Rmalove, Sbooth, Skmackin, Smenck2, Smurawski, WikiWorks753, Wikiexpert, Zamoses

Safety Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58788 *Contributors*: Apyster, Asquires, Bkcase, Cnielsen, Dfairley, Dhenry, Dholwell, Jgercken, Mhaas, Phisterp, Radcock, Rmalove, Skmackin, Smenck2, Smurawski, WikiWorks, Wikiexpert, Zamoses

Security Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58902 *Contributors*: Apyster, Araher, Asquires, Bkcase, Cnielsen, Dfairley, Dhenry, Dholwell, Jgercken, Mhaas, Phisterp, Radcock, Skmackin, Smenck2, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

Electromagnetic Interference/Electromagnetic Compatibility *Source*: https://www.sebokwiki.org/d/index.php?oldid=58125 *Contributors*: Apyster, Araher, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Jsnoderly, Mhaas, Nicole.hutchison, Phisterp, Radcock, Rturner, Skmackin, Smenck2, Smurawski, Wikiexpert, Zamoses

System Resilience *Source*: https://www.sebokwiki.org/d/index.php?oldid=58925 *Contributors*: Apyster, Araher, Asquires, Bkcase, Dhenry, Dholwell, Jbrtis, Jgercken, Kguillemette, Mhaas, Nicole.hutchison, Phisterp, Radcock, Rturner, S.Jackson, Sjackson, Skmackin, Smenck2, Smurawski, WikiWorks, WikiWorks753, Wikiexpert, Zamoses

Manufacturability and Producibility *Source*: https://www.sebokwiki.org/d/index.php?oldid=58536 *Contributors*: Apyster, Araher, Asquires, Bkcase, Dfairley, Dhenry, Dholwell, Jgercken, Kforsberg, Mhaas, Nicole.hutchison, Phisterp, Radcock, Rturner, Skmackin, Wikiexpert, Zamoses

Affordability *Source*: https://www.sebokwiki.org/d/index.php?oldid=58320 *Contributors*: Araher, Bkcase, Dhenry, Dholwell, Kguillemette, Mhaas, Phisterp, Radcock, Rmadachy, Smenck2, Wikiexpert

Environmental Engineering *Source*: https://www.sebokwiki.org/d/index.php?oldid=58227 *Contributors*: Bkcase, Cnielsen, Dhenry, Dholwell, Eleach, Kguillemette, Mhaas, Phisterp, Radcock, Smenck2, Smurawski, WikiWorks, Wikiexpert

Image Sources, Licenses and Contributors

File:Rob_cloutier_bio_photo.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:Rob_cloutier_bio_photo.jpg *License*: unknown *Contributors*: -

File:RobSignature2.jpeg *Source*: https://www.sebokwiki.org/d/index.php?title=File:RobSignature2.jpeg *License*: unknown *Contributors*: Bkcase

File:Hutchison profilephoto.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:Hutchison_profilephoto.png *License*: unknown *Contributors*: Bkcase

File:SEBoK Navigation Related.PNG *Source*: https://www.sebokwiki.org/d/index.php?title=File:SEBoK_Navigation_Related.PNG *License*: unknown *Contributors*: Bkcase

File:Fig_1_A_generic_life_cycle_KF.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:Fig_1_A_generic_life_cycle_KF.png *License*: unknown *Contributors*: Bkcase

File:Aligned-Process-Models.PNG *Source*: https://www.sebokwiki.org/d/index.php?title=File:Aligned-Process-Models.PNG *License*: unknown *Contributors*: Bkcase

File:PM-SE1.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:PM-SE1.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:PM-SE2.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:PM-SE2.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:P6_Fig1_The_Organizational_Continuum_KN.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:P6_Fig1_The_Organizational_Continuum_KN.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:ACQProcessModel_NoWhiteS.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:ACQProcessModel_NoWhiteS.png *License*: unknown *Contributors*: Dhenry, Smenck2, Smurawski

File:RelatingACQtoRFP_NoWhiteS.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:RelatingACQtoRFP_NoWhiteS.png *License*: unknown *Contributors*: Smenck2, Smurawski

File:Fig._1_Integration_Process_for_Specialty_Engineering.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:Fig._1_Integration_Process_for_Specialty_Engineering.png *License*: unknown *Contributors*: Smenck2, Smurawski

File:Fault_tree.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:Fault_tree.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:Simple_RBD.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:Simple_RBD.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:Figure 1.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:Figure_1.png *License*: unknown *Contributors*: Smenck2, Smurawski

File:Figure 2.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:Figure_2.jpg *License*: unknown *Contributors*: Smenck2, Smurawski

File:Disruption_Diagram.PNG *Source*: https://www.sebokwiki.org/d/index.php?title=File:Disruption_Diagram.PNG *License*: unknown *Contributors*: Bkcase, Smurawski

File:ResilienceScenarioComponents.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:ResilienceScenarioComponents.png *License*: unknown *Contributors*: Bkcase

File:StateTransitionModel.png *Source*: https://www.sebokwiki.org/d/index.php?title=File:StateTransitionModel.png *License*: unknown *Contributors*: Nicole.hutchison

File:Environmental_Engineering_HighRes.jpg *Source*: https://www.sebokwiki.org/d/index.php?title=File:Environmental_Engineering_HighRes.jpg *License*: unknown *Contributors*: Smenck2, Smurawski