SWEBOK Knowledge Area Description for Software Engineering Infrastructure (version 0.5)

David Carrington
Department of Computer Science and Electrical Engineering
The University of Queensland
Brisbane, Qld 4072
Australia
+61 7 3365 3310
davec@csee.uq.edu.au

1 INTRODUCTION

This document provides an initial breakdown of topics within the Software Engineering Infrastructure Knowledge Area as defined by the document "Approved Baseline for a List of Knowledge Areas for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge". This Knowledge Area contains the topics of development methods and software development environments. The Industry Advisory Board in Mont-Tremblant identified "standard designs", "integration" and "reuse" as potential additions to this Knowledge Area. Development of this document has been guided by the document "Knowledge Area Description Specifications for the Stone Man Version of the Guide to the Software Engineering Body of Knowledge (version 0.2)".

2 SOURCES USED

The five standard texts [DT97, Moo98, Pfl98, Pre97, and Som96] have been supplemented by Tucker [Tuc96], who provides nine chapters on software engineering topics. In particular, Chapter 112, "Software Tools and Environments" by Steven Reiss {Rei96] was particularly helpful for this Knowledge Area. Specialised references have been identified for particular topics, e.g., Szyperski [Szy97] for component-based software. *Note: citations will use the letters plus year digits form in the initial versions of this document while the list of references is volatile. Ultimately reference numbers will be used to conform to the IEEE style.*

3 AUTHOR REFLECTIONS

Scope of Knowledge Area

The request by the Industry Advisory Board that the topics "standard designs", "integration" and "reuse" be added to this Knowledge Area has proven difficult to achieve in a coherent fashion. Few links have been identified between these topics and the original topics associated with this Knowledge Area. Appendix 1 of the "Approved Baseline" document suggests that an additional Knowledge Area covering the life cycle operation of integration would be an alternative location for these topics. Initial development of the Software Engineering Infrastructure Knowledge Area supports this proposal. Such a Knowledge Area also strengthens the conformance to the structure identified in the ISO/IEC 12207 standard.

LEAVE BLANK THE LAST 2.5 cm (1") OF THE LEFT COLUMN ON THE FIRST PAGE FOR THE COPYRIGHT NOTICE.

Do SWEBOK Knowledge Area Descriptions need to preserve this space?

Approaches to Knowledge Area Breakdown

The Stone Man Version of the Guide to the Software Engineering Body of Knowledge conforms at least partially with the partitioning of the software life cycle in the ISO/IEC 12207 Standard. Some Knowledge Areas, such as this one, are intended to cover knowledge that applies to multiple phases of the life cycle. One approach to partitioning topics in this Knowledge Area would be to use the software life cycle phases. For example, software methods and tools could be classified according to the phase with which they are associated. This approach was not seen as effective. If software engineering infrastructure could be cleanly partitioned by life cycle phase, it would suggest that this Knowledge Area could be eliminated by allocating each part to the corresponding life cycle Knowledge Area, e.g., infrastructure for software design to the Software Design Knowledge Area. Such an approach would fail to identify the commonality of, and interrelationships between, both methods and tools in different life cycle phases.

There are many links between methods and tools, and one possible structure would seek to exploit these links. However because the relationship is not a simple "one-to-one" mapping, this structure has not been used to organise topics in this Knowledge Area. This does mean that these links are not explicitly identified.

Some topics in this Knowledge Area do not have corresponding reference materials identified in the matrices in Appendix 2. There are two possible conclusions: the topic area is not relevant to this Knowledge Area, or additional reference material needs to be identified. Feedback from reviewers will be helpful to resolve this issue.

4 DEFINITION OF KNOWLEDGE AREA

The Software Engineering Infrastructure Knowledge Area includes both the development methods and the software development environments knowledge areas identified in the Straw Man version of the guide.

Development methods impose structure on the software development activity with the goal of making the activity systematic and ultimately more likely to be successful. Methods usually provide a notation and vocabulary, procedures for performing identifiable tasks and guidelines for checking both the process and the product. Development methods vary widely in scope, from a single life cycle phase to the complete life cycle.

Software development environments are the computer-based tools that are intended to assist the software development process. Tools are often designed to support particular methods, reducing any administrative load associated with applying the method manually. Like methods, they are intended to make development more systematic, and they vary in scope from supporting individual tasks to encompassing the complete life cycle.

The emergence of software components as a viable approach to software development represents a maturing of the discipline to overcome the "not invented here" syndrome. The point is often made that more mature engineering disciplines rely on component technologies [Szy97]. Using components affects both methods and tools but the extent of this effect is currently difficult to quantify.

5 OUTLINE OF KNOWLEDGE AREA

This section contains a "first-cut" breakdown of topics in the Software Engineering Infrastructure Knowledge Area.

The Development Methods section has been divided into three subsections: *heuristic methods* dealing with informal approaches, *formal methods* dealing with mathematically based approaches, and *prototyping methods* dealing with software development approaches based on various forms of prototyping. The three subsections are not disjoint; rather they represent distinct concerns. For example, an object-oriented method may incorporate formal techniques and rely on prototyping for verification and validation.

The top-level partitioning of the Software Tools section is based initially on part of the ISO/IEC 12207 Standard by separating out development and maintenance, supporting activities and management tools. The remaining categories cover integrated tool sets (also known as software engineering environments), and tool assessment techniques. This section is broader than the preceding methods section as it covers the full spectrum of tools.

The term "software document" refers generically to any product of a software life-cycle task. Creation and editing refers to human-controlled development whereas translation refers to machine-controlled development.

The Component Integration section dealing with components and integration has been partitioned into topics dealing with individual components, reference models that describe how components can be combined, and the more general topic of reuse.

I. Development Methods

A. Heuristic methods
   1. structured methods
   2. data-oriented methods
   3. object-oriented methods
   4. real-time methods

B. Formal methods
   1. specification languages: *model-oriented, property-oriented, behaviour-oriented, visual, executable*
   2. refinement (reification)
   3. verification/proving properties: *theorem proving, model checking*

C. Prototyping methods
   1. styles: *throwaway, evolutionary/iterative, executable specification*
   2. prototyping target: *requirements, design, user interface*
   3. evaluation techniques

II. Software Tools

A. Development & maintenance tools
   1. creation & editing of software documents
   2. translation of software documents: *preprocessors, compilers, linkers/loaders, GUI generators, document generators*
   3. analysis of software documents: *syntactic/semantic; data, control flow and dependency analysers*
   4. comprehension tools: *debuggers, static and dynamic visualisation, program slicing*
   5. reverse and re-engineering tools

B. Supporting activities tools
   1. configuration management tools: *system builders, version managers*
   2. review tools
   3. verification & validation tools: a*nimation, prototyping and testing tools*

C. Management tools
   1. measurement tools
   2. planning and estimation tools
   3. communication tools (groupware)

D. Workbenches: integrated CASE tools and Software Engineering Environments
   1. storage repositories (software engineering databases)
   2. integration techniques: *platform, presentation, process, data, control*
   3. process tools
   4. meta-tools

E. Tool assessment techniques

III. Component Integration

A. Component definition
   1. interface specifications
   2. protocol specifications
   3. off-the-shelf components

B. Reference models
   1. patterns
   2. frameworks
   3. standard architectures: c*lient-server, middleware products*
   4. semantic interoperability

C. Reuse
   1. types of reuse: *code, design, requirements*
   2. re-engineering
   3. reuse repositories
   4. cost/benefit analysis

6 RELEVANT STANDARDS

No standards for software development methodologies have been identified although individual methods are sometimes standardised.

For software tools, the relevant standards are:

- Trial-Use Standard Reference Model for Computing System Tool Interconnections, IEEE Std 1175-1992
- IEEE Recommended Practice for the Evaluation and Selection of CASE Tools, IEEE Std 1209-1992 (ISO/IEC 14102)
- IEEE Recommended Practice for the Adoption of CASE Tools, IEEE Std 1348-1995 (ISO/IEC 14471)

For reuse, the relevant standards are:

- IEEE Standard for Information Technology — Software Reuse—Data Model for Reuse Library Interoperability: Basic Interoperability Data Model (BIDM), IEEE Std 1420.1-1995
- Supplement to IEEE Standard for Information Technology —Software Reuse—Data Model for Reuse Library Interoperability: Asset Certification Framework, IEEE Std 1420.1a-1996
- Guide for Information Technology—Software Reuse—Concept of Operations for Interoperating Reuse Libraries, IEEE Std 1430-1996

7 CLASSIFICATION OF KNOWLEDGE AREA

In this section, the topics of this Knowledge Area will be classified according to the categories of engineering design knowledge defined by Vincenti [Vin90, Chapter 7].

To be completed.

8 RELATED DISCIPLINES

Mastery of the Software Engineering Infrastructure Knowledge Area is considered to require knowledge of the following Knowledge Areas of Related Disciplines:

- Computer Science
- Mathematics
- Project Management

To be completed.

The full list of Knowledge Areas of Related Disciplines can be found in the document "A Proposed Baseline for a List of Related Disciplines".

ACKNOWLEDGEMENTS

This document was developed from the jump-start document written by Tuong Vinh Ho. Phil Cook, Andrew Coyle, Ian MacColl and Jim Welsh helped to identify relevant information sources and offered helpful advice. Their contribution is gratefully acknowledged. The two referees for version 0.1, Don Bagert and Jorge Diaz-Herrera, provided useful advice about topics and structure (see Appendix 3).

REFERENCES

1. Alan W. Brown and Kurt C. Wallnau. Engineering of Component-Based Systems. In *Component-Based Software Engineering*, Alan W. Brown (Editor), IEEE Computer Society Press, pages 7-15, 1996.
2. Edmund M. Clarke, Jeanette M. Wing et al. Formal Methods: State of the Art and Future Directions. *ACM Computer Surveys*, 28(4):626-643, 1996.
3. Merlin Dorfman and Richard H. Thayer, Editors. *Software Engineering*. IEEE Computer Society, 1997.
4. James W. Moore. S*oftware Engineering Standards: A User's Road Map*. IEEE Computer Society, 1998.
5. Shari Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall, 1998.
6. Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. 4th edition, McGraw-Hill, 1997.
7. Steven P. Reiss. Software Tools and Environments, Ch. 112, pages 2419-2439. In Tucker [Tuc96], 1996.
8. Ian Sommerville. *Software Engineering*. 5th edition, Addison-Wesley, 1996.
9. Clemens Szyperski. *Component Software: Beyond Object-oriented Programming*. Addison-Wesley, 1997.
10. Allen B. Tucker, Jr., Editor-in-chief. *The Computer Science and Engineering Handbook*. CRC Press, 1996.
11. Walter G. Vincenti. *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. John Hopkins University Press, 1990.

APPENDIX 1: BLOOM'S TAXONOMY ANALYSIS

This appendix will (when complete) identify for each topic in this Knowledge Area the level at which a "graduate plus four years experience" should "master this topic.

To be completed.

APPENDIX 2: REFERENCE MATERIAL MATRICES

The following matrices indicate for each topic sources of information within the selected references (see Section 2). The tutorial volume [DT97] contains a collection of papers organised into chapters. The following papers are referenced (section numbers have been added to reference individual papers more conveniently):

Chapter 5: Software Development Methodologies

5.1 Object-oriented Development, Linda M. Northrup

5.2 Object-oriented Systems Development: Survey of Structured Methods, A.G. Sutcliffe

5.4 A Review of Formal Methods, Robert Vienneau

Chapter 12 Software Technology

12.1 The Re-engineering and Reuse of Software, Patrick A.V. Hall and Lingzi Jin

12.2 Prototyping: Alternate Systems Development Methodology, J.M. Carey

12.3 A Classification of CASE Technology, Alfonso Fuggetta

The text by Pfleeger [Pfl98] is structured according to the phases of a life cycle so that discussion of methods and tools is distributed throughout the book.

Note: formatting these matrices within the two-column format is excessively constraining. Landscape mode may provide the maximum flexibility.

I. Development Methods	BW96	CW96	DT97	Moo98	Pfl98	Pre98	Rei96	Som96	Szy97
A. Heuristic Methods						10-23			
1. structured methods			5.2		4.5,	10-18		15	
2. data-oriented methods			5.2			12.8			
3. object-oriented methods			5.1, 5.2		4.4, 7.5	19-23		6.3, 14	
4. real-time methods						15		16	
B. Formal Methods		✓	5.4			24, 25		9-11	
1. specification languages		✓			4.5	24.4			
2. refinement						25.3			
3. verification/proving properties		✓			5.7, 7.3			24.2	
C. Prototyping Methods			12.2		4.6, 5.6	2.5, 11.4		8	
1. styles			12.2		4.6	11.4			
2. prototyping target			12.2						
3. evaluation techniques									

II. Software Tools	BW96	CW96	DT97	Moo98	Pfl98	Pre98	Rei96	Som96	Szy97
A. Development & maintenance tools			12.3		10.5	29	112.2		
1. creation & editing			12.3				112.2		
2. translation tools			12.3		10.5		112.2		
3. analysis tools		✓	12.3		7.7, 8.7		112.5	24.3	
4. comprehension tools			12.3				112.5		
5. reverse & re-engineering tools			12.3						
B. Supporting Activity tools			12.3				112.3		
1. configuration management			12.3		10.5			33.3, 33.4	
2. review tools			12.3						
3. verification & validation tools		✓	12.3		7.7, 8.7		112.3		
C. Management tools			12.3						
1. measurement tools			12.3						
2. planning & estimation tools			12.3						
3. communication (groupware) tools			12.3						
D. Workbenches and CASE tools			12.3	11	1.8	29	112.3, 112.4	25-27	
1. storage repositories			12.3			29.6			
2. integration techniques			12.3	11	1.8	29.5		27.1	
3. process tools			12.3		2.3, 2.4		112.4		
4. meta-tools			12.3					26.4	
F. Tool assessment techniques				11	8.10				

III. Component Integration	BW96	CW96	DT97	Moo98	Pfl98	Pre98	Rei96	Som96	Szy97
A. Component definition	✓		12.1			26.4			4, 5, 22
1. interface specifications						26.5			5
2. protocol specifications									
3. off-the-shelf components	✓					26.5			
B. Reference models	✓								9, 20, 21
1. patterns									9
2. frameworks									21
3. standard architectures	✓					28		13	20
4. semantic interoperability	✓								9
C. Reuse			12.1	11	11.4	26, 27		20	
1. types of reuse			12.1		11.4	26.2		20	
2. re-engineering			12.1		10.6	27		34	
3. reuse repositories			12.1		11.4	26.5			
4. cost/benefit analysis			12.1		11.4	26.6			

APPENDIX 3: FEEDBACK ON VERSION 0.1

The following comments were received as feedback on version 0.1 of this document and were handled as described below in italics. For subsequent reviews, it may be more appropriate to create a separate document for the feedback comments and their disposition.

Don Bagert:

Here are my comments concerning Infrastructure v0.1:

1. Concerning "Development Methods", section 1 (object-oriented methods) should include a subsection on patterns. Even though patterns are called "Design Patterns", the "Design Methods" subsection is not sufficient to include patterns, which are more of an architectural technique.

   Patterns have been included in the Component Integration section. The restructuring of the Development Methods section reduces the significance of any addition to this section.

2. Once again concerning "Development Methods", section 4 (structured methods") should include a subsection on structured programming. (Note: Section 1 includes a subsection on object-oriented programming; either structured and OO programming should be both in or both out; I suggest that they are both in.)

   The restructuring of the Development Methods section reduces the significance of this suggestion although the words "structured programming" have been added.

3. For section 8 (Related Disciplines), I would include Computer Science, Mathematics, and possibly Project Management.

   This suggestion has been incorporated in this version.

Jorge L. Diaz Herrera:

The comments below correspond to Carrington's document "SWEBOK Knowledge Area Description for Software Engineering Infrastructure (version 0.1).

Development methods

1. I suggest reorganising the main partitioning of Development Methods into three main categories related by a single aspect (type of method) as follows:

   1) Heuristic methods
      -- make Object-oriented and Structure methods subcategories of this partition
      -- add structured programming, integration and testing, and metrics to Structured methods category (just like for OO)
      -- what about data-oriented methods?
      -- what about real-time methods?

   2) Formal methods

   3) Prototyping methods
      -- add "iii) operational Specifications" to a) styles

   This restructuring suggestion has been adopted.

Software Development Tools

2. could this area be renamed simply "Software Tools" (to avoid redundancy below)?

   Incorporated

3. under 1.b, add "iv) GUI generators" and "v) document generators"

   Incorporated

4. under 3.c.iii, add "(#) Dependency analysers" (use or static structure)

   Incorporated

5. delete 3.d "Documentation tools" since this is included in 1.b above; document generation to conform to a standard is more part of "translation of software documents" than to "supporting tools."

   Incorporated

Component Integration

This area seems to require more work. A clearer definition is lacking. E.g., the whole category of reuse needs to be developed. Numerous topics related to reuse are missing, although they may be treated elsewhere in the SWEBOK; these include product-line practices, domain engineering, domain analysis, development-for-reuse (generic designs) vs. development-with-reuse (product configuration). Also component integration approaches (like top-down vs. bottom-up) may fit in this category too.

Additional work is definitely required in this section. The issue of whether this section belongs in this Knowledge Area is still to be resolved.

Here are some specific comments on this area.

6. not sure if "1.b Off-the-shelf components" belong here.

   Left in for now

7. add "1.c protocol specifications" (counterpart to interface specifications"

   Incorporated

8. change "2) Standard Designs" to "2) Reference Models"

   Incorporated

9. Add "2.d Patterns"

   Incorporated

10. In category 3, be consistent "3) reuse vs. b) re-use"

    Fixed

11. not sure we need “3.a types of reuse …”

    Left in for now

12. in 3.b, “libraries” seems too restrictive, I suggest “repositories”

    Incorporated