

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE  
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO  
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR  
THE DEGREE OF DOCTOR OF PHILOSOPHY  
Ph. D.

BY  
Kenza MERIDJI

ANALYSIS OF SOFTWARE ENGINEERING PRINCIPLES FROM  
AN ENGINEERING PERSPECTIVE

MONTREAL, JANV 11, 2010

© Kenza Meridji, 2010

THIS THESIS HAS BEEN EVALUATED  
BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain Abran, PhD, Thesis Supervisor  
Département de génie logiciel et des TI École de Technologie Supérieure

Mr. Pierre Bourque, PhD, President of the Board of Examiners  
Département de génie logiciel et des TI École de Technologie Supérieure

Mr. Eric Lefevre, PhD, Examiner  
Département de génie logiciel et des TI École de Technologie Supérieure

Mr. Juan Garbajosa Sopena, PhD, External Examiner  
Universidad Politécnica de Madrid (UPM), Spain

THIS THESIS WAS PRESENTED AND DEFENDED  
BEFORE A BOARD OF EXAMINERS AND THE PUBLIC

DECEMBER 22, 2009

A L'ECOLE DE TECHNOLOGIE SUPERIEURE

## ACKNOWLEDGMENTS

First and foremost I would like to express my gratitude to Professor Alain Abran, my thesis supervisor at École de technologie supérieure for his continuous support, time, advice and patience throughout this thesis. Without his patient guidance, this work would never have been carried out.

I wish also to thank my examiners Dr. Pierre Bourque, Eric Lefevre and Juan Garbajosa Sopena. Also, I would like to thank Dr. Normand Seguin for his input to the 34 candidate principles to my thesis.

I would like to thank everyone in the Software Engineering Research Laboratory (GÉLOG), the Department of Software Engineering and IT. Finally, I wish to express my thanks to my family for their support and understanding throughout this long process.

# ANALYSE DES PRINCIPES DU GENIE LOGICIEL D'UNE PERSPECTIVE D'INGENIERIE

Kenza MERIDJI

## RÉSUMÉ

L'ingénierie du logiciel a récemment émergé comme un nouveau domaine d'ingénierie et continue d'évoluer. Le génie logiciel est une discipline dont l'objectif est la production de logiciels de haute qualité, mais il manque de maturité par rapport aux autres domaines de l'ingénierie traditionnelle. Les domaines traditionnels de l'ingénierie ont leurs propres principes basés sur la physique, la chimie ou les mathématiques. Puisque le domaine du génie logiciel n'est pas fondé sur les lois de la nature, il est plus difficile de comprendre l'ensemble de ses principes.

Cette recherche sur l'ensemble des principes fondamentaux candidats contribuera à une meilleure compréhension et, éventuellement, à l'enseignement des principes du génie logiciel. En outre, elle aidera à améliorer le contenu du Guide SWEBOK du point de vue du génie.

Ce travail de recherche a permis d'étudier la question du génie logiciel comme une discipline du génie en utilisant les catégories de connaissances en génie de Vincenti, d'identifier des principes fondamentaux à partir d'un ensemble de candidats, et enfin d'examiner l'absence de description explicite et systématique de ces principes, et leur application, dans le Guide SWEBOK.

Les deux principaux objectifs de cette étude sont l'identification des principes fondamentaux de l'ingénierie du génie logiciel à partir des 34 principes candidats et la description des directives opérationnelles pour ces principes en utilisant comme base le contenu du Guide SWEBOK.

Pour atteindre ces objectifs, la méthodologie suivante de recherches a été utilisée. Les principales phases de cette méthodologie de recherche sont: l'analyse, d'un point de vue d'ingénierie, de la question du génie logiciel et de l'ensemble des 34 principes fondamentaux candidats, l'identification des principes de génie logiciel dans le contenu du Guide SWEBOK - ISO TR 19759, la description des lignes directrices opérationnelles sur la base du contenu du Guide SWEBOK et aligné avec la norme IEEE 1362-1998 Concept of Operations (CONOPS) Document.

Le résultat de cette thèse est l'identification d'un ensemble de neuf principes fondamentaux du génie logiciel et la description de directives opérationnelles pour ces principes.

**Mots-clés:** Principes de génie logiciel, principes fondamentaux, Vincenti, perspective de l'ingénierie.

# ANALYSIS OF SOFTWARE ENGINEERING PRINCIPLES FROM AN ENGINEERING PERSPECTIVE

Kenza MERIDJI

## ABSTRACT

Software engineering has recently emerged as a new engineering field in a continuing evolution. Software engineering is a discipline whose aim is the production of high quality software, but lacks maturity compared to other traditional engineering fields. Traditional engineering fields have their own principles originating from physics, chemistry and mathematics. However, since the software engineering discipline is not based on natural laws, establishing a set of principles is more challenging.

This research on the set of candidate fundamental principles will contribute to a better understanding and possibly, to the teaching of the principles of software engineering and it will help improve the content of the software engineering body of knowledge (SWEBOK) Guide from an engineering perspective.

This research work investigated the issue of software engineering as an engineering discipline using Vincenti categories of engineering knowledge; identified engineering fundamental principles from a set of candidates; and finally investigated the lack of explicit and systematic descriptions of these principles and their application, as in the SWEBOK Guide.

The two main research objectives are the identification of the fundamental principles of software engineering from the 34 candidates principles; and the description of operational guidelines for these principles, based on the content of the SWEBOK Guide.

To achieve these objectives, the following research methodology was used. The main phases of this research methodology are: the analysis, from an engineering perspective, of software engineering and the set of 34 fundamental principles candidates; the identification of the software engineering principles in the content of the SWEBOK Guide – ISO TR 19759; the description of the operational guidelines on the basis of the content of the SWEBOK Guide and aligned with the IEEE standard 1362-1998 Concept of Operations (ConOps) Document.

The main outcome of this research study is the identification of a set of nine software engineering fundamental principles and the description of operational guidelines.

**Keywords:** Software engineering principles, candidate fundamental principles CFP, Vincenti, engineering perspective.

## TABLE OF CONTENTS

	Page
INTRODUCTION .....	1
CHAPTER 1 SOFTWARE ENGINEERING PRINCIPLES IN THE LITERATURE.....	6
1.1 Introduction.....	6
1.2 Candidates principles of software engineering.....	6
1.3 The identification of engineering knowledge types and characteristics .....	17
1.4 Other related works.....	19
1.5 Summary .....	22
CHAPTER 2 RESEARCH OBJECTIVES AND METHODOLOGY .....	24
2.1 Introduction.....	24
2.2 Research goal .....	25
2.3 Research objectives.....	25
2.4 Research motivation.....	25
2.5 Users of research.....	26
2.6 Research inputs .....	26
2.7 Overview of the research methodology .....	26
2.8 Detailed research methodology.....	28
CHAPTER 3 ANALYSIS OF SOFTWARE ENGINEERING FROM AN ENGINEERING PERSPECTIVE.....	32
3.1 Introduction.....	32
3.2 Vincenti's engineering viewpoint.....	33
3.2.1 Overview and context .....	33
3.2.2 Vincenti's categorization criteria & goals .....	34
3.3 Vincenti's classification of engineering knowledge types.....	40
3.3.1 Relationship between the various categories .....	40
3.4 Vincenti's classification of engineering knowledge-type models .....	41
3.4.1 Fundamental design concepts .....	41
3.4.2 Criteria and specifications.....	45
3.4.3 Theoretical tools.....	46
3.4.4 Quantitative data .....	47
3.4.1 Design instrumentalities.....	49
3.5 The Design process .....	51
3.5.1 The engineering design process in Vincenti .....	51
3.5.2 The Engineering process in the SWEBOK.....	54
3.5.3 Design notion in the SWEBOK Guide .....	54
3.5.4 Design KA: mapping between Vincenti and the SWEBOK Guide.....	54
3.6 Mapping results for the Vincenti classification of engineering knowledge .....	58

3.6.1	Software requirements: Vincenti's view point .....	65
3.6.2	Software design: Vincenti view .....	65
3.6.3	Software construction: Vincenti view.....	66
3.7	Analysis using the Vincenti classification of engineering knowledge .....	66
3.8	Summary .....	67
CHAPTER 4	SOFTWARE ENGINEERING PRINCIPLES: DO THEY MEET ENGINEERING CRITERIA? .....	69
4.1	Introduction.....	69
4.2	Analysis methodology .....	70
4.2.1	Step 1: Identification of two sets of verification criteria. ....	71
4.2.2	Step 2: Verification execution .....	72
4.2.3	Step 3: Analysis and selection .....	72
4.2.4	Step 4: Design and execution of an external verification .....	72
4.3	Identification of engineering criteria: step 1 .....	73
4.3.1	Vincenti.....	73
4.3.2	IEEE and ACM joint curriculum .....	74
4.4	Verification against the two sets of criteria: step 2.....	74
4.5	Analysis and consolidation using both sets of criteria: step 3 .....	76
4.5.1	Analysis across each set of engineering criteria .....	76
4.5.2	Identification of a hierarchy.....	80
4.6	External verification: step 4.....	81
4.6.1	Design .....	81
4.6.2	Execution .....	82
4.7	Summary .....	83
CHAPTER 5	IDENTIFICATION OF SOFTWARE ENGINEERING PRINCIPLES WITHIN THE CONTENT OF THE SWEBOK GUIDE .....	85
5.1	Introduction.....	85
5.2	Mapping the FP to the SWEBOK KAs.....	85
5.3	The FP in software requirements knowledge area.....	86
5.4	The FP in software quality knowledge area.....	89
5.5	Results in other KAs .....	91
5.5.1	Software design knowledge area .....	91
5.5.2	Software construction knowledge area .....	93
5.5.3	Software testing knowledge area .....	94
5.5.4	Software maintenance knowledge area.....	95
5.5.5	Software configuration management knowledge area.....	96
5.5.6	Software engineering management knowledge area.....	98
5.5.7	Software engineering process knowledge area .....	100
5.6	Analysis of the mapping results.....	101
5.7	Summary .....	103

CHAPTER 6	DESCRIPTION OF AN OPERATIONAL PERSPECTIVE OF THE SOFTWARE ENGINEERING PRINCIPLES ON THE BASIS OF THECONTENT OF THE SWEBOK GUIDE .....	105
6.1	Introduction.....	105
6.2	Proposed operational guidelines for the SWEBOK (Annex D).....	106
6.3	Software requirements – description of an operational perspective .....	107
6.4	Software design– description of an operational perspective.....	114
6.5	Software construction– description of an operational perspective .....	120
6.6	Software testing– description of an operational perspective .....	127
6.7	Software maintenance– description of an operational perspective.....	138
6.8	Software configuration management: operational perspective of the software engineering FP .....	144
6.9	Software engineering process– description of an operational perspective .....	148
6.10	Software quality– description of an operational perspective .....	154
6.11	Summary .....	161
CHAPTER 7	DEVELOPMENT OF A CONSOLIDATED SWEBOK VIEW FOR MEASUREMENT FP.....	162
7.1	Introduction.....	162
7.2	Coverage of the measurement principle in the KAs of the SWEBOK guide .....	162
7.3	Consolidated view of measurement FP.....	164
7.4	Consolidated view model of the measurement FP.....	165
7.5	Measurement process:.....	165
	7.5.1 Establish and sustain measurement commitment activity .....	165
	7.5.2 Plan the measurement process .....	166
	7.5.3 Perform the measurement process .....	166
	7.5.4 Evaluate the measurement process .....	167
7.6	Summary .....	167
CHAPTER 8	ANALYSIS OF A SWEBOK KA FROM AN ENGINEERING PERSPECTIVE WITH RESPECT TO THE ENGINEERING FUNDAMENTAL PRINCIPLES.....	169
8.1	Introduction.....	169
8.2	Identification of engineering concepts in the “Software requirements” KA with respect to the FP and Vincenti. ....	170
	8.2.1 Mapping 1: Vincenti categories of engineering knowledge and software requirements KA.....	170
	8.2.2 Mapping 2: The list of FP to each of the SWEBOK KAs .....	170
	8.2.3 Mapping 3: Vincenti’s categories of engineering knowledge the “Software requirements” with respect the FP .....	173



8.3	Vincenti's categories and FP in the requirements KA.....	174
8.4	Mapping results from Vincenti's viewpoint .....	175
8.5	Mapping results from the FP viewpoint in the requirements KA.....	176
8.6	Summary .....	179
CHAPTER 9	DEVELOPING AN EVALUATION METHOD TO VERIFY THE OPERATIONAL GUIDELINES IN THE SWEBOK GUIDE .....	181
9.1	Introduction.....	181
9.2	Evaluation method for the operational guidelines .....	181
9.2.1	Phase 1: Design of an operational model of operational guidelines.....	182
9.2.2	Phase 2: Conduct the evaluation procedure .....	184
9.2.3	Phase 3: Evaluation results .....	185
9.3	Summary .....	188
CONCLUSION.....		190
ANNEX I	CLASSIFICATION OF THE KNOWLEDGE CONTAINED IN THE SWEBOK GUIDE FOR THE 3 KAs. ....	197
ANNEX II	PRESENTATION OF THE KNOWLEDGE CONTAINED IN THE 3 KA OF THE SWEBOK GUIDE. ....	212
ANNEX III	MAPPING RESULTS BETWEEN THE FP AND THE ENGINEERING CRITERIA .....	216
ANNEX IV.	DETAILED RESULTS FOR THE IDENTIFICATION OF THE SOFTWARE ENGINEERING PRINCIPLES WITHIN SWEBOK GUIDE CONTENT – ISO TR 19759.....	220
ANNEX V	PROPOSED DRAFT: OPERATIONAL GUIDELINES OF THE NINE FUNDAMENTAL PRINCIPLES OF SOFTWARE ENGINEERING BASED ON THE SWEBOK GUIDE CONTENT - ISO TR 19759.....	223
ANNEX VI	MAPPING SWEBOK KA WITH VINCENTI SIX CATEGORIES AND THE NINE FUNDAMENTAL PRINCIPLES (9 FP). ....	260
ANNEX VII	WORKSHOP INTERNATIONAL CONFERENCE ON ENGINEERING EDUCATION – ICEE 2007- COIMBRA (PORTUGAL) .....	268
BIBLIOGRAPHY.....		271

## LIST OF TABLES

	Page
Table 1.1	List of candidate fundamental principles.....10
Table 1.2	Key characteristics of publications of software engineering principles ....12
Table 1.3	Classification of references (in alphabetic order) .....13
Table 1.4	Inventory of the candidate FP .....15
Table 1.5	Engineering criteria from the IEEE & ACM joint software engineering ..18
Table 3.1	Vincenti’s vocabulary relating to engineering terms and concepts .....34
Table 3.2	Vincenti: Engineering knowledge categories and concepts.....36
Table 3.3	Vincenti: Engineering knowledge categories and goals .....39
Table 3.4	Design according to Vincenti vs. design in the software engineering life cycle .....55
Table 3.5	Mapping of the design process in engineering life cycle.....57
Table 3.6	Software requirements in the SWEBOK guide .....60
Table 3.7	Software design in the SWEBOK guide .....61
Table 3.8	Software construction in the SWEBOK guide.....63
Table 4.1	Engineering criteria identified in Vincenti .....73
Table 4.2	Identification of IEEE & ACM engineering criteria.....74
Table 4.3	Candidate FP that directly meets criteria from either sets of criteria .....77
Table 4.4	List of software engineering FP .....79
Table 4.5	Hierarchy of candidate FP .....80
Table 5.1	Summary mapping of the engineering FP in the “Software requirements” KA.....87
Table 5.2	Detailed mapping of the engineering FP in the “Software requirements” KA.....87
Table 5.3	Summary mapping of the engineering FP in the “Software requirements” KA.....90
Table 5.4	Detailed mapping of the engineering FP in the “Software quality” KA....90
Table 5.5	Presence of engineering FP in the “Software design” KA .....91

Table 5.6	Detailed presence of engineering FP in the “Software design” KA .....92
Table 5.7	Presence of engineering FP in the “Software construction” KA .....93
Table 5.8	Detailed presence of engineering FP in the “Software construction” KA.93
Table 5.9	Presence of engineering FP in the “Software testing” KA .....94
Table 5.10	Detailed presence of engineering FP in the “Software testing” KA .....94
Table 5.11	Presence of engineering FP in the “Software maintenance” KA.....95
Table 5.12	Detailed presence of engineering FP in the “Software maintenance” KA 96
Table 5.13	Presence of engineering FP in the “Software configuration management” KA.....96
Table 5.14	Detailed presence of engineering FP in “Software Configuration Management” .....97
Table 5.15	Presence of engineering FP in the “Software engineering management” KA.....98
Table 5.16	Detailed presence of engineering FP in: “Software engineering management” .....99
Table 5.17	Presence of engineering FP in the “Software engineering process” KA .100
Table 5.18	Detailed presence of engineering FP in: Software engineering process..100
Table 7.1	The measurement FP in the SWEBOK guide KA. ....163
Table 7.2	Consolidated View of the measurement FP .....164
Table 8.1	Mapping Vincenti’s categories to the FP in the “Software requirements” KA.....175
Table 9.1	Evaluation results of the measurement FP in the SWEBOK KA .....187

## LISTE OF FIGURES

		Page
Figure 1.1	Relationships between principles, standards and practices.....	8
Figure 1.2	Project steps of the study on fundamental principles.....	9
Figure 1.3	Knowledge areas (KA) of the SWEBOK guide .....	20
Figure 2.1	Research methodology – overview of phases .....	29
Figure 3.1	Vincenti’s classification of engineering knowledge .....	40
Figure 3.2	Relationships between theoretical tools & quantitative data .....	41
Figure 3.3	Elements of a fundamental design concept.....	42
Figure 3.4	Project input .....	42
Figure 3.5	Designers initial knowledge.....	43
Figure 3.6	Design pyramid .....	43
Figure 3.7	Relationships between normal configurations, operational principles .....	44
Figure 3.8	Designer’s goals.....	45
Figure 3.9	Problem definition level output .....	46
Figure 3.10	Theoretical tools model.....	47
Figure 3.11	Quantitative data model .....	48
Figure 3.12	Practical considerations model .....	50
Figure 3.13	Design instrumentalities model.....	51
Figure 3.14	Modeling of the levels of design hierarchy, as described in Vincenti .....	53
Figure 3.15	Design in Vincenti vs. design in the software engineering life cycle.....	57
Figure 4.1	The Four-steps verification process .....	70
Figure 4.2	Identification of Vincenti engineering criteria (Vincenti W. G. 1990).....	71
Figure 4.3	Identification of the IEEE-ACM engineering criteria .....	71
Figure 4.4	Step 2: Verification process against engineering criteria .....	72
Figure 5.1	Frequency of engineering fundamental principles by knowledge area ...	102
Figure 5.2	Frequency of engineering fundamental principles for SWEBOK KAs...	103
Figure 6.1	SWEBOK Guide: “Software requirements” knowledge area.....	107
Figure 6.2	Measurement process: Operational view –requirements KA .....	108

Figure 6.3	General view for applying “Build with and for reuse” in requirements ..109
Figure 6.4	Requirements elicitation- operational view in requirement KA .....111
Figure 6.5	SWEBOK guide: “Software design” knowledge area .....114
Figure 6.6	SWEBOK guide: “Software construction” knowledge area.....120
Figure 6.7	Measurement in the construction KA .....121
Figure 6.8	Construction activities- operational view in construction KA.....122
Figure 6.9	Constructing for verification- operational view in construction KA .....125
Figure 6.10	Standards- operational view in construction KA .....125
Figure 6.11	SWEBOK guide: “Software testing” knowledge area .....127
Figure 6.12	Evaluation of the program under test.....129
Figure 6.13	Phases for the testing activities- Operational view in testing KA .....134
Figure 6.14	Phases for the testing levels - operational view in testing KA .....135
Figure 6.15	Testing techniques- operational view in testing KA.....136
Figure 6.16	SWEBOK Guide: “Software maintenance” knowledge area .....138
Figure 6.17	Maintenance activities- operational view in maintenance KA .....142
Figure 6.18	SWEBOK guide: “Software configuration “management knowledge ....144
Figure 6.19	Software configuration status accounting- operational view in the configuration management KA.....146
Figure 6.20	SWEBOK guide: “Software engineering process” knowledge area.....148
Figure 6.21	Related product measurements- operational view in the process KA .....150
Figure 6.22	Software process management cycle- operational view in the process ...152
Figure 6.23	Process assessment models and operational view in the process KA.....153
Figure 6.24	SWEBOK guide: “Software quality” knowledge area .....154
Figure 6.25	Software management processes .....156
Figure 6.26	Software quality assurance .....157
Figure 6.27	Management reviews- operational view in the quality KA .....158
Figure 6.28	Technical reviews - operational view in the quality KA .....159
Figure 6.29	Inspections - operational view in the quality KA .....160
Figure 7.1	Model of a consolidated SWEBOK view of the measurement FP .....166
Figure 8.1	Mapping of the Vincent’s engineering knowledge to the SWEBOK .....171

Figure 8.2	Mapping the set of the FP to the SWEBOK guide .....	172
Figure 8.3	Mapping of the categories of engineering knowledge to the set of FP in “Software requirements” KA .....	173
Figure 8.4	Vincenti’s six categories & the FP frequencies in requirements KA. ....	177
Figure 8.5	Frequency of fundamental principles for “Software requirements” .....	179
Figure 9.1	The three phases of the evaluation procedure of operational guidelines .	182
Figure 9.2	Operational model of operational guidelines .....	183
Figure 9.3	Generic evaluation procedure .....	184
Figure 9.4	Evaluation procedure of operational guidelines .....	185

## LIST OF ABBREVIATIONS

SWEBOK	Software Engineering Body of Knowledge
KA	Knowledge area
FP	Fundamental principles
CFP	Candidate fundamental principles
Engineering FP	Engineering fundamental principles
R&D	Research and development
SCR	Software change request
SE	Software engineering
IT	Information technology

## INTRODUCTION

Software engineering is defined as “The application of a systematic, disciplined, quantitative approach to the development, operation and maintenance of software, the application of engineering to software” (IEEE-Std 610.12 1990). Software engineering is somewhat unusual as an engineering discipline because software does not exist in nature, unlike in other traditional engineering fields where engineers observe natural laws and try to understand them. Software engineers must observe software projects which are intellectual products and not the products of nature. Figuring out the list of fundamental principles for software engineering represents therefore a challenge.

“There are millions of software professionals worldwide, and software is a ubiquitous presence in our society” (ISO-TR-19759, 2004). However, the recognition of software engineering as an engineering discipline is still being challenged.

“Achieving consensus by the profession on a core body of knowledge is a key milestone in all disciplines, and has been identified by the IEEE Computer Society as crucial for the evolution of software engineering towards professional status” (ISO-TR-19759, 2004).

Software engineering, as a discipline, is certainly not yet as mature as other engineering disciplines. As a new engineering discipline, in comparison to other engineering disciplines, software engineering does not have yet a wide consensus on its engineering foundations.

This research aims to contribute to the maturation of the foundations of the software engineering through: the analysis of the Software Engineering Body of Knowledge (SWEBOK) (ISO-TR-19759 2004) from an engineering perspective; the analysis, from an engineering perspective, of the set of the 34 candidate fundamental principles for software engineering; and the implementation of operational guidelines of software engineering fundamental principles, on the basis of the SWEBOK Guide.



## **Problem statement**

Software engineering is not as mature as other engineering discipline, lacking well recognized fundamental principles that contribute to the foundation of an engineering discipline.

The research work in software engineering has focused on developing methods, techniques and tools. Less work has been done on defining fundamental principles of software engineering and much less R&D has been done to verify candidate fundamental principles. Most of the authors who have proposed candidates for fundamental principles have proposed individual opinions about these principles, and most have not carried out research to support their proposals of candidate fundamental principles.

The content of each knowledge area (KA) in the SWEBOK Guide was developed by domain experts and extensively reviewed by an international community of peers. This Delphi-type approach, while very extensive and paralleled by national reviews at the ISO level, did not specifically address the engineering perspective, nor did it provide a structured technique to ensure the completeness and full coverage of fundamental engineering topics. Therefore, no evidence was provided that had adequately tackled the identification and documentation of the knowledge expected to be present in an engineering discipline.

An example of a structured research on the identification of software engineering principles has been undertaken by Seguin who identified 34 candidate fundamental principles (Séguin N. 2006). However, this set of candidate principles has not been analyzed from an engineering perspective.

This thesis aims to analyse software engineering from an engineering perspective, analyse the 34 candidates principles from an engineering perspective and provide an explicit and systematic description of these engineering principles, and of their application, for example in the SWEBOK Guide.

## **Thesis organization**

This thesis contains nine chapters and seven Annexes. The current introduction outlines the problem statement and the organization of the thesis.

Chapter 1 presents an overview of the literature review done on the fundamental principles of software engineering.

Chapter 2 presents the research project definition with its research goals, objectives and users of the research results. Chapter 2 also presents the detailed methodology designed to tackle the research objectives, including the research phases and the research inputs.

Chapter 3 presents the analysis of software engineering from an engineering perspective. This chapter presents the analysis of Vincenti's categories of engineering knowledge, a comparison between traditional design vs. software engineering design and the application of Vincenti categories of engineering knowledge to some of the SWEBOK knowledge areas

Chapter 4 presents the process for identifying software engineering principles. This chapter describes the identification of engineering criteria from Vincenti and IEEE & ACM, the application of these criteria to the 34 candidate fundamental principles, the corresponding analysis and selection of fundamental principles and, a design and execution of an external verification.

Chapter 5 presents the coverage of the software engineering principles within the content of the SWEBOK Guide (ISO-TR-19759 2004)

Chapter 6 presents a description, from an operational perspective, of the software engineering principles on the basis of the content of the SWEBOK Guide (ISO-TR-19759 2004): in this chapter, the description of these operational perspective are aligned with the

IEEE 1362:1998 Guide for information technology- Concepts of Operations (ConOps) (IEEE STD 1362-1998) .

Chapter 7 presents a consolidated view of the engineering fundamental principle on software measurement.

Chapter 8 presents the design of an evaluation procedure for the operational guidelines for evaluation purposes and also presents the evaluation of the operational guidelines for the engineering fundamental principle on software measurement, as documented in the SWEBOK Guide.

Chapter 9 presents the analysis of the SWEBOK “Software requirements” KA from an engineering perspective with respect to the engineering fundamental principles. Throughout this chapter, the Vincenti’s categories of engineering knowledge of the “Software requirements” knowledge area are mapped to the engineering fundamental principles

The Conclusion chapter summarizes the results of this thesis, its contributions and limitations, the expected impacts and suggestions future work.

Finally, this thesis contains seven Annexes.

Annex I presents the results of the mapping between the knowledge contained in the SWEBOK Guide and the Vincenti’s categories of engineering knowledge for the “Software requirements”, “Software design” and “Software construction” knowledge areas.

Annex II presents the new breakdown of the knowledge contained in the SWEBOK Guide for the “Software requirements”, “Software design” and “Software construction” knowledge areas by categories of Vincenti’s engineering knowledge.

Annex III presents the mapping results between the engineering fundamental principles and Vincenti and the IEEE & ACM engineering criteria.

Annex IV presents the detailed results related to the mapping of the set of the nine engineering principles into the related knowledge areas of the SWEBOK Guide.

Annex V presents the detailed operational guidelines of the principles of software engineering aligned with the IEEE Std 1362-1998 IEEE Guide for Information Technology System Definition. Concepts of Operations (ConOps) Document.

Annex VI presents the mapping between the Vincenti's categories of engineering knowledge and the lists of engineering principles for the "Software requirements", "Software design" and "Software construction" knowledge areas.

Annex VII presents the program of the workshop the "Engineering foundations of software engineering".

## CHAPTER 1

### SOFTWARE ENGINEERING PRINCIPLES IN THE LITERATURE

#### 1.1 Introduction

Software engineering is a new emerging engineering discipline in comparison to traditional engineering disciplines. Many authors have published on principles of software engineering, Normand Seguin (Séguin N. 2006) has conducted a literature survey on the principles of software engineering.

As an engineering discipline, software engineering should be analyzed from an engineering viewpoint and should have a recognized set of principles: however, software engineering does not have yet a set of recognized engineering principles and there is not yet agreement on a well documented and established foundation from an engineering perspective.

This chapter presents the literature survey and is organized as follows: section 1.2 describes related work undertaken on principles of software engineering that is, work published over the last decade on the principles that have been proposed for software engineering. Section 1.3 introduces the identification of engineering knowledge types and characteristics. Section 1.4 presents other related works and finally section 1.5 presents a summary.

#### 1.2 Candidates principles of software engineering

##### Davis 1995

(Davis A.M. 1995) published a book on “201 Principles of Software Development” which contains the first published collection of principles of software development. Davis proposed a definition of the term “principle” and organized his 201 principles into categories; these categories are composed of the different phases of software development in addition to management, product assurance and evolution. Davis provided a definition for each of the

201 principles. He identified his 15 most important principles of software engineering in an article published in 1994 “Fifteen principles of software engineering” (Davis A.M. 1994). Davis did not provide any criteria for their identification nor did he provide a methodology.

### **Jabir, Moore 1998**

Jabir is a surname given to a group of experts who participated in a 1996 to a workshop at the IEEE Forum on Software Engineering Standards Issues where eight candidate principles were identified.

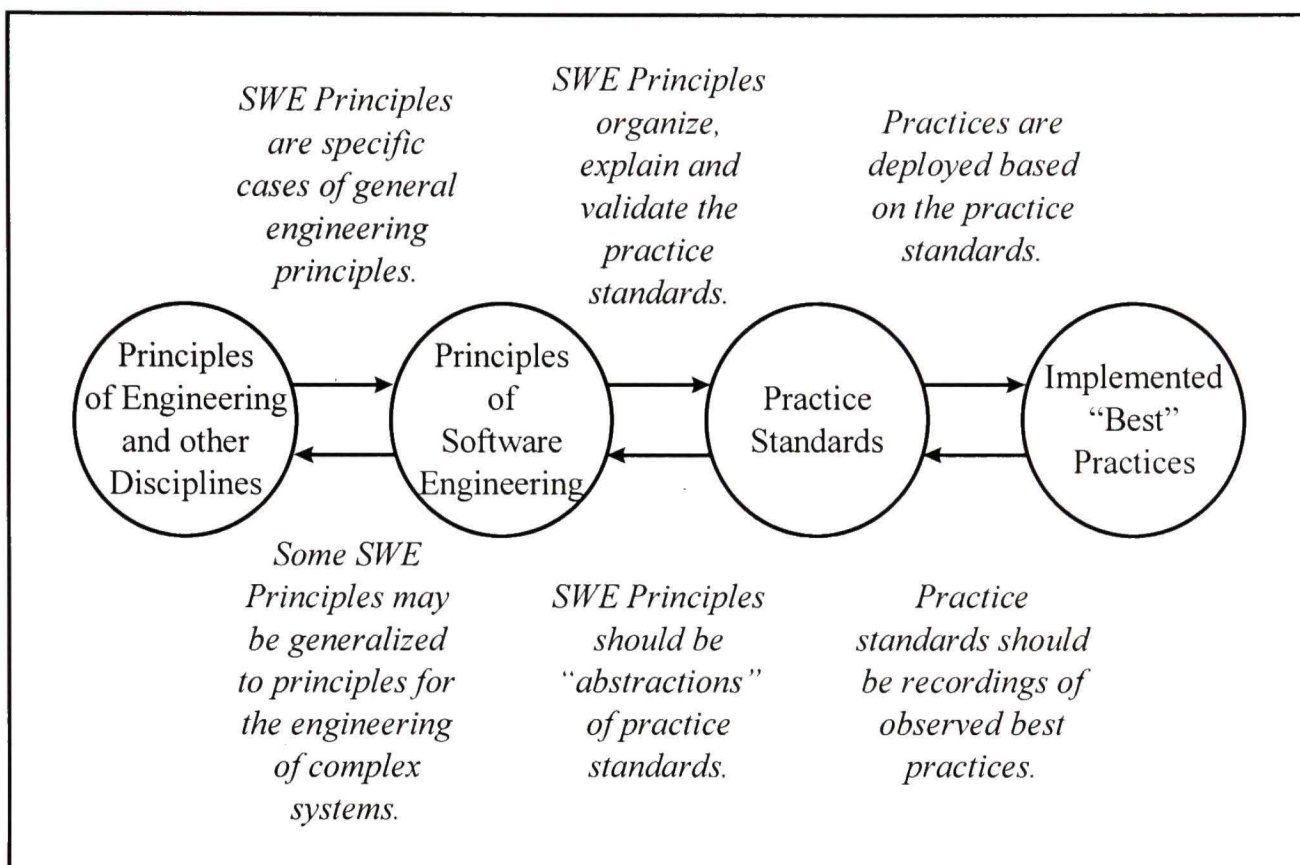
Jabir *et al.* published in 1998 “A search for fundamental principles of software engineering” (Jabir *et al.* 1998). Jabir *et al.* explored the nature of software engineering as well as the relationships between principles, standards and practices. Furthermore, Jabir *et al.* discussed the characteristics and criteria for identifying fundamental principles and applied these principles to the eight identified principles.

### **Dupuis et al. 1999, Bourque et al. 2002**

In their paper “Fundamental principles of software engineering – a journey” published in 2002 (Bourque P. et al. 2002) identified a set of fundamental principles through a well documented research methodology. Defining the relationships between principles, standards and implemented best practices as illustrated in Figure 1.1.

Furthermore, the following seven criteria were identified to select a candidate fundamental principle:

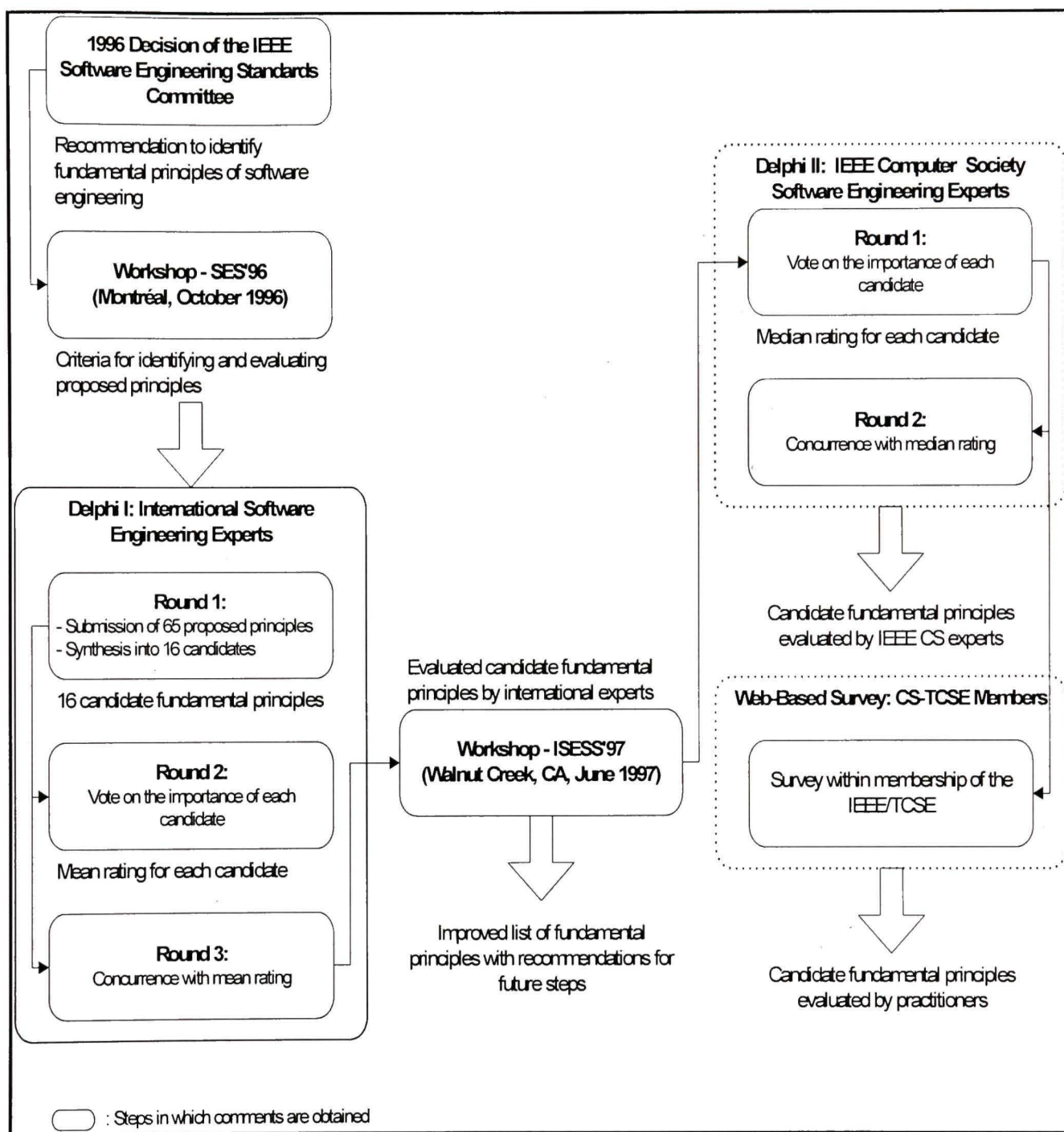
- Fundamental principles are less specific than methodologies and techniques;
- Fundamental principles are more enduring than methodologies and techniques;
- Fundamental principles are typically discovered or abstracted from practice and should have some correspondence with best practices;



**Figure 1.1 Relationships between principles, standards and practices**  
(Bourque P. et al. 2002)

- Software engineering fundamental principles should not contradict more general fundamental principles, but there may be tradeoffs in the application of principles;
- A fundamental principle should not conceal a tradeoff;
- A fundamental principle should be precise enough to be capable of support or contradiction;
- A fundamental principle should relate to one or more underlying concepts.

The research methodology to identify their first set of candidate fundamental principles of software engineering included two workshops, two Delphi studies composed respectively of three and two rounds and a web based survey which was conducted over the internet among a group of software engineering experts from the IEEE Technical Committee on Software Engineering IEEE-TCSE – Figure 2.1



**Figure 1.2 Project steps of the study on fundamental principles in**  
(Bourque P. et al. 2002)

These studies resulted in a list of 15 candidate fundamental principles of software engineering see Table 1.1.



Table 1.1 List of candidate fundamental principles  
(Bourque P. et al. 2002)

Identification	Candidate fundamental principles
(A)	Apply and use quantitative measurements in decision-making
(B)	Build with and for reuse
(C)	Control complexity with multiple perspectives and multiple levels of abstraction
(D)	Define software artifacts rigorously
(E)	Establish a software process that provides flexibility
(F)	Implement a disciplined approach and improve it continuously
(G)	Invest in the understanding of the problem
(H)	Manage quality throughout the life cycle as formally as possible
(I)	Minimize software component interaction
(J)	Produce software in a stepwise fashion
(K)	Set quality objectives for each deliverable
(L)	Since change is inherent to software, plan for it and manage it
(M)	Since tradeoffs are inherent to software engineering, make them explicit and document them
(N)	To improve design, study previous solutions to similar problems
(O)	Uncertainty is unavoidable in software engineering. Identify and manage it

**Baskerville, Ramesh, Levine, Pries-Heje, Slaughter 2003**

In the article “Is internet-speed software development different?” Baskerville et al. (2003) developed practices and principles for Internet speed applications. The purpose of this study was to clarify how and why internet speed application practices were different from traditional engineering practices. The methodology used was based on a two- phased study. In this study, one note that the traditional software development principles defined by (Bourque P. et al. 2002) do not overlap with the internet speed development principles, and that internet speed development practices are compatible with agile principles. However, there was no definition for the terms practices, principles and metaprinciples. Also, there was no difference documented between principles and metaprinciples (Baskerville R. et al. 2003).

**Ghezzi et al. (2003)**

Ghezzi *et al.* published in 2003 a software engineering textbook “Fundamentals of Software Engineering”. In their book, the authors described seven software engineering principles. The authors also provided definitions and examples for each of their seven principles (rigor and formality, separation of concerns, modularity, abstraction, anticipation of change, generality and incrementality). The authors did not follow any methodology for the identification and definition of these principles. The authors used their set of principles with examples to attain the quality objectives for design, specification and the management of software engineering (Ghezzi C. et al. 2003).

**Abran, Séguin, Bourque, Dupuis (2004)**

(Abran A. et al. 2004) published a literature survey “The search for software engineering principles: An overview of results”, where they reviewed the related research publications from individual authors as well as collaborative work done on software engineering principles.

Abran *et al.* came up with the key characteristics of the 1983-2002 publications on software engineering principles. These key characteristics of the publications surveyed includes the terms used by the authors, whether or not definitions were provided and the criteria for identifying a principle, the number of proposed principles, the statement style and the source of the proposed list of principles (expert opinions, literature, observation and historical data) see Table 1.2

Table 1.2 Key characteristics of publications on software engineering principles  
(in alphabetical order of authors)  
(Abran A. et al. 2004)

Reference	Terms	Definition	Criteria	Number	Statement style	Source
Boehm (1983)	Principle	None	Yes (2)	7	Rules	Historical data analysis
Booch & Bryan (1994)	Principle	None	No	7	Concept	Literature
Bourque al. (2002)	Principle	Yes	Yes (8)	15	Rules	Expert opinions
Buschman et al. et al. (1996)	Principle/ Technique	None	No	10	Concept	Literature
Davis (1995)	Principle	Yes	No	201	Rules	Literature
Ghezzi <i>et al.</i> (2003)	Principle	Yes	No	7	Mix	Literature, opinion
Lehman (1980)	Laws	None	No	5	Concept	Observation, analysis
Maibaum (2000)	Principle	None	No	3	Concept	Opinion
Meyer (2001)	Principle	Yes	No	13	Mix	Opinion
Mills (1980)	Principle	None	No	4	Concept	Opinion
Royce (1970)	Steps	None	No	5	Rules	Opinion
Wasserman (1996)	Concept	None	No	8	Concept	Opinion, literature
Wieggers (1996)	Principle	None	No	14	Rules	Observation, opinion

In addition, for each reference Abran *et al.* presented publication type, the basis for discussion, an overview of the research methodology, supporting number of references and the scope, as illustrated in Table 1.3.

Table 1.3 Classification of references (in alphabetic order)  
(Abran A. et al. 2004)

Reference	Publication type	Discussion	Research methodology	Supporting number of references	Scope
Boehm (1983)	Paper	Empirical	Implicit	49	Life cycle
Booch & Bryan (1994)	Book	Theoretical	Implicit	12	Construction
Bourque et al. (2002)	Paper	Empirical	Explicit	11	Life cycle
Buschman et al. (1996)	Book	Theoretical	-	10	Architecture
Davis (1995)	Book	Theoretical	Implicit/analytic	124	Life cycle
Ghezzi al. (2003)	Book	Theoretical	Implicit	24	Life cycle
Lehman (1980)	Paper	Empirical	Implicit/observation	13	Maintenance
Maibaum (2000)	Paper	Theoretical	-	11	General
Meyer (2001)	Paper	Theoretical	-	10	Curriculum
Mills (1980)	Paper	Theoretical	-	16	General
Royce (1970)	Paper	Theoretical	-	0	Life cycle
Wasserman (1996)	Paper	Theoretical	-	19	General
Wieggers (1996)	Book	Theoretical	Experimentation	-	Software engineering culture

A key finding of this study is that the research work published on the search fundamental principles to software engineering had not been based on a research methodology, to the exception of Bourque *et al.*, but rather on personal observations and opinions.

## Normand Seguin 2006

Subsequently, (Abran A. et al. 2004), (Séguin N. 2006), (Séguin N. 2007) inventoried, from the literature on software engineering principles, 308 principles proposed in the work of individual authors - for instance: (Boehm B.W. 1983), (Davis A.M. 1995), (Wieggers K.E. 1996) - or in collaborative effort: (Bourque P. et al. 2002), (Buschmann F. et al. 1996), (Ghezzi C. et al. 2003), (Dupuis R. et al. 1999), (Bourque P. and Dupuis R. 1997), (Dupuis R. et al. 1997). To the exception of (Ghezzi C. et al. 2003), these authors have proposed only nominative principles, without including either formal definitions or procedures for implementing these principles.

To verify whether or not each of these 308 proposed principles was indeed a candidate fundamental principle (CFP), a two-step verification process was used in (Séguin N. 2006), (Séguin N. 2007):

### Step 1: Identification of verification criteria

A. Five individual criteria were identified in (Jabir et al 1998):

- A principle is a proposal formulated in a prescriptive way;
- A principle should not be directly associated with, or arise from, a technology, a method, or a technique, or itself be an activity of software engineering;
- The principle should not dictate a compromise (or a proportioning) between two actions or concepts;
- A principle of software engineering should include concepts connected to the engineering discipline;
- It must be possible to test the formulation of a principle in practice, or to check its consequences.

B. Two additional criteria across the full set of proposals were also identified in (Jabir et al 1998):

- The principles should be independent, not deduced (Boehm B.W. 1983)
- A principle should not contradict another known principle (Bourque P. et al. 2002).

Step 2: Analytical verification of each of the proposed 308 principles surveyed against these criteria. In (Séguin N., 2006) it is reported that only 34 out of the 308 proposals met the full set of criteria to be recognized as CFP. Table 1.4 presents the list of these CFP from (Séguin N., 2006), in alphabetical order. However this set of 34 CFP still has not been analyzed from an engineering perspective.

Table 1.4 Inventory of the candidate FP  
(Séguin N. 2006)

No.	Candidate fundamental principles – in alphabetical order
1	Align incentives for developer and customer
2	Apply and use quantitative measurements in decision making
3	Build software so that it needs a short user manual
4	Build with and for reuse
5	Define software artifacts rigorously
6	Design for maintenance
7	Determine requirements now
8	Don't overstrain your hardware
9	Don't try to retrofit quality
10	Don't write your own test plans
11	Establish a software process that provides flexibility
12	Give product to customers early
13	Grow systems incrementally
14	Implement a disciplined approach and improve it continuously
15	Invest in the understanding of the problem
16	Involve the customer
17	Keep design under intellectual control
18	Maintain clear accountability for results
19	Produce software in a stepwise fashion

Table 1.4 Inventory of the candidate FP (continued)  
(Séguin N. 2006)

No.	Candidate fundamental principles – in alphabetical order
20	Fix requirement specification errors now
21	Quality is the top priority; long-term productivity is a natural consequence of high quality
22	Rotate (high performer) people through product assurance
23	Since change is inherent to software, plan for it and manage it
24	Since tradeoffs are inherent to software engineering, make them explicit and document them
25	Strive to have a peer, rather than a customer, find a defect
26	Tailor cost estimation methods
27	To improve design, study previous solutions to similar problems
28	Use better and fewer people
29	Use documentation standards
30	Write programs for people first
31	Know software engineering's techniques before using development tools
32	Select tests based on the likelihood that they will find faults
33	Choose a programming language to assure maintainability
34	In the face of unstructured code, rethink the module and redesign it from scratch

### Yingxu Wang 2007

In his book “Software engineering foundations”, Wang stated that “Software engineering is an immature and fast growing discipline which depends on multidisciplinary foundations such as philosophy, computation, mathematics, informatics, systems engineering management, cognitive informatics, linguistics and engineering economics”.

The author in his book tried to identify and explore the various knowledge and disciplines that form the foundations of software engineering. The objective of his study was to define a

framework that integrates a set of software engineering principles. Wang inventoried the principles of software engineering from different authors and among these, the work done by Dupuis et al. 1999 (Dupuis R. et al. 1999). Wang inventoried a list of 55 principles of software engineering from different authors. His methodology consisted next in the elimination of the overlaps between these 55 principles: as a result, 31 principles remained.

Next, the author proposed what he referred to as a unified framework of software engineering principles based on the mapping between each of the 31 proposed principles of software engineering.

The fifty five principles inventoried from the literature and from which 31 principles were derived were considered by Wang as engineering principles (Wang Y. 2007). But a key question still remains: are these really engineering principles?

### **1.3 The identification of engineering knowledge types and characteristics**

#### **Vincenti 1990**

In his book “What engineers know and how they know it”, Vincenti described different types of engineering knowledge based on his study of the epistemology of engineering. Vincenti analyzed five case studies in aeronautical engineering over a period of fifty years and proposed six categories of engineering knowledge. Vincenti stated that this classification can be transposed to other engineering domains (Vincenti W. G. 1990). His engineering knowledge types are:

- Fundamental design concepts: contains “the operational principle of the device”;
- Criteria and specifications: allow the engineer to “translate general, quantitative goals couched in concrete technical terms”;
- Theoretical tools: to support engineers work. These include mathematical methods and theories involved for the design calculation;



- Quantitative data: used by engineers. This data is obtained based on empirical observation or calculated with mathematical models;
- Practical considerations: are activities without formal codification;
- Design instrumentalities: contain “the procedure, ways of thinking and judgmental skills by which it is done”.

### IEEE & ACM joint curriculum 2004

The IEEE Computer Society (IEEE-CS) and the Association for Computing Machinery (ACM) published in 2004 recommendations, for the software engineering curriculum (IEEE and ACM, 2004).

The IEEE and ACM SE curriculum includes (in its chapter 2) a list of seven engineering characteristics. According to the IEEE and ACM, these seven characteristics are common to all engineering disciplines. Table 1. describes these characteristics that can also apply to the software engineering discipline.

Table 1.5 Engineering criteria  
(IEEE and ACM, 2004)

ID.	Engineering criteria
1	Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision point that is appropriate for the current task in the current context. Appropriateness can be judged by tradeoff analysis, which balances costs against benefits.
2	Engineers measure things, and, when appropriate, work quantitatively; they calibrate and validate their measurements; and they use approximations based on experience and empirical data.
3	Engineers emphasize the use of a disciplined process when creating a design and can operate effectively as part of a team in doing so.
4	Engineers can have multiple roles: research, development, design, production, testing, construction, operations, management, and others, such as sales, consulting, and teaching.

Table 1.5 Engineering criteria (continued)  
(IEEE and ACM, 2004)

ID.	Engineering criteria
5	Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering.
6	Engineers, via their professional societies, advance by the development and validation of principles, standards, and best practices.
7	Engineers reuse designs and design artefacts.

#### 1.4 Other related works

##### **SWEBOK guide 2004**

The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), written under the auspices of the IEEE Computer Society's Professional Practices Committee, was initiated in 1998 to develop an international consensus in pursuing the following objectives:

- To characterize the content of the software engineering discipline;
- To promote a consistent view of software engineering worldwide;
- To provide access to the software engineering body of knowledge;
- To clarify the place, and set the boundary, of software engineering with respect to other disciplines;
- To provide a foundation for curriculum development and individual certification material.

In 2004, the IEEE Computer Society and ISO published a guide the software engineering body of knowledge – the SWEBOK Guide (ISO-TR-19759 2004).

The SWEBOK Guide is subdivided into ten knowledge areas. Each knowledge area is composed of topics and subtopics. Figure 1.3 illustrates the ten knowledge areas.

<b>KA01 - Requirements</b>	<b>KA02 - Design</b>	<b>KA03 - Construction</b>	<b>KA04 - Testing</b>	<b>KA05 - Maintenance</b>
<b>KA06 - Software Configuration Management</b>				
<b>KA07 - Software Engineering Management</b>				
<b>KA08 - Software Engineering Process</b>				
<b>KA09 - Software Engineering Tools and Methods</b>				
<b>KA10 - Software Quality</b>				

**Figure 1.3 Knowledge areas (KA) of the SWEBOK guide  
(ISO-TR-19759 2004)**

The content of each knowledge area (KA) in the SWEBOK Guide was developed by domain experts and extensively reviewed by an international community of peers.

The content of the SWEBOK Guide has many objectives. However, these objectives do not include the identification of software engineering fundamental principles nor of their operational guidelines.

### **Robert, Abran, Bourque 2002**

In 2002, Robert *et al.* published “A technical review of the software construction knowledge area in the SWEBOK Guide”. Robert *et al.* used (Vincenti W. G. 1990) classification of engineering knowledge to identify the types of engineering knowledge contained in the “Software construction” knowledge area of the trial version of the SWEBOK Guide. The goal of this analysis was the identification of the weaknesses in the “Software construction” knowledge area.

In conclusion, a new breakdown was proposed by Robert *et al.* for the “Software construction” KA. The use of Vincenti’s classification helped in clarifying the missing parts

of engineering knowledge in the “Software construction” of the 2001 trial version of the SWEBOK Guide (Robert F. et al. 2002).

### **Guay 2004**

A comparative analysis was undertaken by (Guay et al. 2004) in “Comparative analysis between the SWEBOK Guide and the fundamental principles of software engineering” to evaluate the Software Engineering Body of Knowledge (SWEBOK) Guide with respect to its coverage of the fifteen candidate fundamental principles identified by (Bourque et al.2002) in “Fundamental principles of software engineering – a journey”.

The methodology consisted of three steps. The first step was the analysis of the composition of the CFP that is, whether each CFP forms one part or can be decomposed into two different parts.

The second step was the identification in the trial version of the textual descriptions that corresponded to each of the fifteen CFP, within the content of the ten knowledge areas and their level of description.

Finally, the third step built a visualization of the breakdown for the correspondence of the different CFP within the SWEBOK Guide (Guay B. 2004).

### **IEEE STD 1362-1998 Concept of Operations (ConOps) Document**

This IEEE guide illustrates the format and contents of the concept of operations (ConOps) document: “A ConOps is a user-oriented document that describes system characteristics of the to-be-delivered system from the end user viewpoint” (IEEE STD 1362-1998).

It is used for “software-intensive systems: software-only or software/hardware/people systems” (IEEE STD 1362-1998). This standard contains the set of elements that should be present in all Concept Operational document described as follow:

The operational concepts “indicate the operational features that are to be provided without specifying the design detail” (IEEE STD 1362-1998).

The operational scenario “is a step-by-step description of how the proposed system should operate and interact with its users and its external interfaces under a given set of circumstances” (IEEE STD 1362-1998).

The operational capabilities are the capabilities of the system provided by scenario.

The operational impact defines the “impacts of the proposed system on the users, the developers, and the support and maintenance organizations” (IEEE STD 1362-1998).

The operational improvements provides “summary of the benefit to be provided by the proposed system” (IEEE STD 1362-1998).

## **1.5 Summary**

The principles of software engineering surveyed in this chapter summarized the pioneer pursuits of software engineering principles in the last 19 years. Many researchers have published on principles of software engineering from 1994 to 2007; Normand Seguin’s study involved a range of authors covering a period of 33 years from 1970 to 2003. As a result, 308 principles were identified (among them the work done by (Bourque *et al.* 2002) from which 34 were identified as candidate principles (Séguin N. 2006). The selection process was rigorous as each of the principles was analyzed through two steps with a number of verification criteria.

Software engineering is defined by IEEE as an engineering discipline; however, it's lacks an established foundation. Needing to be analyzed from an engineering perspective. Software engineering lacks the description of the recognized principles. The candidate principles were surveyed and analyzed by many authors; however, none of them tackled the issue of being engineering principles or not. For these reasons, one should investigate the following research issues:

- The lack of analysis of software engineering from an engineering discipline;
- Are these candidate principles indeed engineering principles or not ?
- The lack of the explicit and systematic description and the application of these software engineering principles in the SWEBOK Guide

## CHAPTER 2

### RESEARCH OBJECTIVES AND METHODOLOGY

#### 2.1 Introduction

The research methodology designed for this research work is qualitative. For the research problem selected for this thesis, the problem addressed is not well understood: this can be explained by a low level of maturity of the domain of study. And also, due to the originality of the domain under investigation, there has been to date very little research work in the area of describing principles of software engineering.

The research issues investigated in this thesis are defined as follow:

- Is software engineering an engineering discipline?
- Are these principles indeed engineering principles or not?
- The lack of explicit and systematic description of these engineering principles, and of their application, for example in the SWEBOK Guide (ISO-TR-19759 2004).

To help structure the research topic, to approach the research problem and to carry out a rigorous scientific investigation, an adaptation of the Basili's framework for experimental research will be followed. The Basili's framework have proven efficient in software engineering research (Basili V. et al. 1986).

This chapter describes the research project definition including: the research goal, the research objectives, the users of research, as well as the research methodology and the research inputs.

## **2.2 Research goal**

Software engineering lacks maturity compared to other engineering disciplines. The research goal of this thesis is to contribute to the software engineering discipline from an engineering perspective, through the identification of software engineering fundamental principles (engineering FP) and the description of operational guidelines for these engineering FP.

## **2.3 Research objectives**

The following two research objectives have been selected:

- Identification of the engineering fundamental principles of software engineering from the 34 candidates identified by (Séguin N. 2006);
- Description of operational guidelines for the engineering fundamental principles based on the content of the SWEBOK Guide.

To achieve these research objectives from an engineering perspective, the following approach was selected: analysis of Vincenti engineering knowledge and of the IEEE & ACM engineering characteristics (IEEE and ACM, 2004), as well as the analysis of the content of the SWEBOK Guide.

## **2.4 Research motivation**

This research study on the set of candidate fundamental principles will contribute to a better understanding and possibly, to the teaching of the principles of software engineering and will help improve the content of the body of knowledge SWEBOK Guide (ISO-TR-19759 2004) from an engineering perspective.



## **2.5 Users of research**

The results of this research will be used mainly by the software engineering research community and specifically by the people working on the foundation of software engineering. It may also provide teachers with teaching material for software engineering courses. In addition, there is interest in the IEEE Computer Society for the two objectives selected in order to better understand the scope of its standards, and of their foundations, as indicated by the chair of the Computer Society's Professional Practices Committee, Mr. James W. Moore.

Ultimately, the result of this research will also provide help to all software engineers wanting to develop software from an engineering approach.

## **2.6 Research inputs**

The key inputs to this research are:

- The Vincenti's classification of engineering knowledge based on five aeronautical case studies (Vincenti W. G. 1990);
- The 34 candidate fundamental principles identified in (Séguin N. 2006);
- SWEBOK Guide (2004): The generally accepted body of knowledge in software engineering - the SWEBOK Guide - (ISO-TR-19759 2004);
- IEEE & ACM Software Engineering Curriculum (IEEE and ACM, 2004).

## **2.7 Overview of the research methodology**

This section presents an overview of the research methodology designed to pursue the research objectives. This research methodology consists of eight phases as seen in figure 2.1.

**Phase 1: Literature survey**

Phase 1 of the research methodology consists of surveying the literature on topics linked to the software engineering principles, engineering knowledge and the SWEBOK Guide from an engineering perspective.

**Phase 2: Analysis of software engineering from an engineering perspective**

Phase 2 of the research methodology consists of analyzing the concept of engineering “design” and comparing it with the “design” concept in the SWEBOK Guide using the Vincenti categories of engineering knowledge.

**Phase 3: Software engineering principles: Do they meet engineering criteria?**

Phase 3 of the research methodology identifies the engineering criteria and analysis of the set of 34 CFP defined by Seguin from an engineering perspective.

**Phase 4: Identification of the software engineering principles in the content of the SWEBOK Guide**

The research methodology for phase 4 identifies the software engineering principles within the knowledge areas of the SWEBOK Guide.

**Phase 5: Identification of the Operational Guidelines in the SWEBOK Guide**

The research methodology for phase 5 proposes operational guidelines for the engineering fundamental principles on the basis of the content of the SWEBOK Guide - (ISO-TR-19759 2004).

**Phase 6: Development of a consolidated SWEBOK view for the Measurement fundamental principle**

The research methodology for phase 6 proposes a consolidated view for the measurement fundamental principle.

### **Phase 7: Analysis of 3 SWEBOK KA from an engineering perspective with respect to the engineering fundamental principles**

The research methodology for phase 7 maps the Vincenti categories of engineering knowledge to the “Software requirements”, “Software design”, “Software construction” knowledge areas with regards to fundamental principles.

### **Phase 8: Evaluation of the operational guidelines in the SWEBOK Guide**

The research methodology for phase 8 consists of the design and execution of an evaluation procedure for the operational guidelines within the SWEBOK Guide for evaluation purposes.

## **2.8 Detailed research methodology**

### **Phase 1: Literature survey**

It is noted from the literature survey in chapter 1 that software engineering still lacks the analysis of software engineering principles from an engineering perspective.

### **Phase 2: Analysis of software engineering from an engineering perspective**

This phase of the methodology consists of following three steps:

- **Step 2.1 Analysis of Vincenti’s categories of engineering knowledge**  
This step identifies and analyzes the six Vincenti’s categories of engineering knowledge (Vincenti W. G. 1990) to facilitate the understanding of these categories: fundamental design concepts, criteria and specifications, theoretical tools, quantitative data, practical considerations, and design instrumentalities.
- **Step 2.2 Analysis and comparison between traditional design vs. design in software engineering**  
This step consists of the mapping of the design concept in (Vincenti W. G. 1990) with the design concept in the SWEBOK Guide (ISO-TR-19759 2004) Figure 2.1.

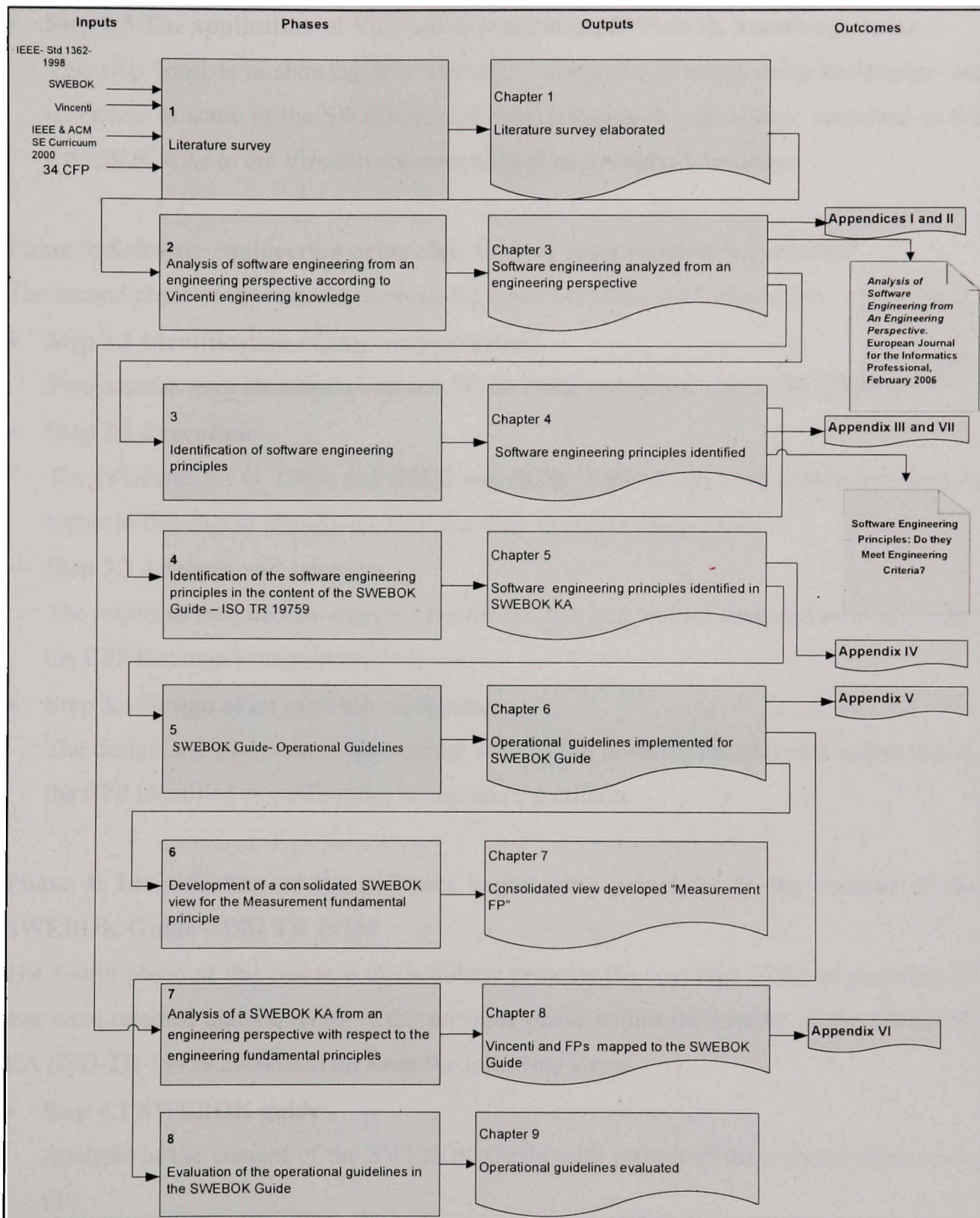


Figure 2.1: Research methodology – overview of phases

- **Step 2.3 The application of Vincenti to some of the SWEBOK knowledge areas.**

This step consists in showing how Vincenti's categories of engineering knowledge are addressed in some of the SWEBOK Guide by mapping the knowledge contained in the SWEBOK KAs to the Vincenti six categories of engineering knowledge.

**Phase 3: Software engineering principles: Do they meet engineering criteria?**

The second phase of the research methodology consists of the following steps:

- **Step 3.1 Identification of engineering criteria**

Two sources were identified (Vincenti W. G. 1990) and (IEEE and ACM, 2004).

- **Step 3.2 Execution**

The (Vincenti W. G. 1990) and (IEEE and ACM, 2004) engineering criteria are used as inputs to this step to identify the CFP that map to engineering criteria;

- **Step 3.3 Analysis and selection**

The results of the previous mapping are used in this step and are analyzed to finally select the CFP that map to engineering criteria;

- **Step 3.4 Design of an external verification**

The design and execution of an external verification to verify the previous output that is, the CFP identified as conforming to engineering criteria.

**Phase 4: Identification of the software engineering principles in the content of the SWEBOK Guide – ISO TR 19759**

The fourth phase of this research methodology presents the coverage of the engineering FP that were retained and validated in the previous phase within the content of the SWEBOK KA (ISO-TR-19759 2004) and includes the following steps:

- **Step 4.1 SWEBOK guide**

Analysis of the content of the SWEBOK Guide with respect of the selected engineering FP;

- **Step 4.2 Mapping FP**

Mapping of each of the selected engineering FP to the content of the SWEBOK Guide KA.

**Phase 5: SWEBOK guide - operational guidelines**

This phase proposes operational guidelines for the SWEBOK Guide (ISO-TR-19759 2004). This phase includes analysis of the content of the chapters of the SWEBOK Guide KA; and description of the operational guidelines structured with (IEEE STD 1362-1998).

**Phase 6: Development of a consolidated SWEBOK view for the measurement fundamental principle**

This phase consists of developing a consolidated view for the measurement fundamental principles and designs a model for a consolidated view.

**Phase 7: Analysis of a SWEBOK KA from an engineering perspective with respect to the engineering fundamental principles**

This phase consists in mapping between Vincenti's categories of engineering knowledge (Vincenti W. G. 1990), the lists of the engineering FP and SWEBOK KA (ISO-TR\_19759 2004). Vincenti's categories of engineering knowledge are used as an analytical tool to map each of the engineering principles that are present in the "Software requirements", "Software design" and "Software construction" KA with respect to engineering fundamental principle.

**Phase 8: Evaluation of the operational guidelines in the SWEBOK guide**

This phase consists of designing and executing a procedure to evaluate the operational guidelines and is composed of design of an operational model and conduct the evaluation procedure.

## CHAPTER 3

### ANALYSIS OF SOFTWARE ENGINEERING FROM AN ENGINEERING PERSPECTIVE

#### 3.1 Introduction

The SWEBOK Guide, also adopted as technical report TR19759 by the (ISO-TR-19759 2004), has been selected to explore the following question: Is software engineering truly an engineering discipline?

This chapter presents phase 2. An approach is proposed to investigate in a structured way the content of the SWEBOK Guide to verify what engineering knowledge is included in this Guide, and what could be missing. This approach is based on Vincenti's classification of engineering knowledge (Vincenti W. G. 1990). However, since this classification of engineering knowledge had not, at the time of this investigation, been used to analyze other engineering disciplines, it was felt necessary to carry out some structuring and modeling of the criteria embedded within Vincenti's work to render its use practical in the analysis of the SWEBOK Guide (ISO-TR-19759 2004).

In particular, the engineering design concepts had to be probed further, since at first glance there seemed to be a disconnect between the SWEBOK Guide concept of design and Vincenti's description of engineering design. Finally, Vincenti's categories of engineering knowledge (Vincenti W. G. 1990) are used to analyze a selection of three chapter's of the SWEBOK Guide: "Software requirements", "Software design" and "Software construction" KA.

This chapter is organized as follows: Section 3.2 introduces Vincenti's engineering viewpoint. Section 3.3 presents Vincenti's classification of engineering knowledge types. Section 3.4 presents Vincenti's classification of engineering knowledge types models.

Section 3.5 introduces the design process. Section 3.6 presents the mapping results. Section 3.7 presents the analysis of the mapping results and in section 3.8 a summary is presented. Annex I describes the mappings between the corresponding concepts for the classification of engineering knowledge types and the related “Software requirements”, “Software design” and in “Software construction” KA.

Annex II presents the new breakdown of the “Software requirements”, “Software design” and in “Software construction” KAs based on the categories of engineering knowledge.

## **3.2 Vincenti’s engineering viewpoint**

### **3.2.1 Overview and context**

As noted in chapter 1, (Vincenti W. G. 1990) in his book “What engineers know and how they know it”, classified categories of engineering knowledge. Furthermore, Vincenti stated that this classification is not specific to the aeronautical engineering domain, but can be transferred to other engineering domains. However, he did not provide documented evidence of this applicability and generalization to other engineering disciplines. In addition, no author could be identified as having attempted to do so either. In this chapter, one proposes some pioneering work on the use of Vincenti’s categorization of engineering knowledge as constituting criteria for investigating software engineering from an engineering perspective.

The Vincenti categorization of knowledge (Vincenti W. G. 1990) was first used in a graduate seminar in 2002 at the École de technologie supérieure, Université du Québec, as an analytical tool to tackle this issue by analyzing each of the SWEBOK KAs (ISO-TR-19759 2004) separately. The initial purpose of this approach was to gain insights into their content and structure which, by definition, were expected to be of an engineering knowledge type. While it was easy for graduate students at the Master’s degree and doctoral levels to use Vincenti’s criteria to analyze the SWEBOK design KA and to propose a mapping to the Vincenti’s categorization, this proved to be much more challenging for all the other KAs, to



the point where some of these students questioned the relevance of the applicability of Vincenti's categorization to these other SWEBOK KAs and as a corollary to that, that these other KAs did not necessarily constitute knowledge of an engineering type. The specific vocabulary defined by Vincenti is presented in Table 3.1.

### 3.2.2 Vincenti's categorization criteria

Vincenti provides a categorization of engineering design knowledge and the activities that generate it. However, the divisions are not entirely exclusive; some items of knowledge can contain the knowledge of more than one category. From Vincenti's definitions of each engineering knowledge-type category (ISO-TR-19759 2004), a number of categories were identified and have been listed in Table 3.2. A short description of each category is provided in Table 3.3.

Table 3.1 Vincenti's vocabulary relating to engineering terms and concepts  
(Vincenti W. G. 1990)

<b>Engineering vocabulary</b>	<b>Definitions</b>
<b>Design</b>	"Denotes both the content of a set of plans (e.g. in the design for a new airplane) and the process by which those plans are produced".
<b>Normal configuration</b>	"The general shape and arrangement commonly agreed upon to best embody the operational principle".
<b>Normal technology</b>	According to Edward's constant that "what technological communities usually do" comprises "the improvement of the accepted tradition or its application under new or more stringent conditions."

Table 3.1 Vincenti's vocabulary relating to engineering terms and concepts  
(Vincenti W. G. 1990) (continued)

Engineering vocabulary	Definitions
<b>Normal design</b>	<p>“The design involved in normal technology”.</p> <p>“The engineer working with such a design knows at the outset how the device in question works and what its customary features are, and that, if properly designed along such lines, it has a good likelihood of accomplishing the desired task”.</p> <p>“Normal design is evolutionary rather than revolutionary”.</p>
<b>Operational principle</b>	<p>Defines the essential fundamental concept of a device,.</p> <p>“How its characteristic parts... fulfill their special function in combination to [sic] an overall operation which archives the purpose.”</p>
<b>Production</b>	<p>“Denotes the process by which these plans are translated into the concrete artifice”.</p>
<b>Operation</b>	<p>“Deals with the employment of the artifice in meeting the recognized need”.</p>
<b>Radical design</b>	<p>“How the device should be arranged, or even how it works, is largely unknown. The designer has never seen such a device before and cannot presume that it will succeed”.</p>
<b>Engineering knowledge</b>	<p>“The knowledge used by engineers”.</p> <p>“Engineering knowledge has to do not only with design, but also with production and operation”.</p>
<b>Descriptive knowledge</b>	<p>“The knowledge of how things are”.</p>
<b>Prescriptive knowledge</b>	<p>“The knowledge of how things should be to attain a desired end”.</p>

Table 3.1 Vincenti's vocabulary relating to engineering terms and concepts  
(Vincenti W. G. 1990) (continued)

Engineering vocabulary	Definitions
<b>Device</b>	"Devices are single, relatively compact entities, such as airplanes, electric generators, turret lathes, and so forth".
<b>Systems</b>	"Systems are assemblies of devices brought together for a collective purpose. Examples are airlines, electric power systems and automobile factories".
<b>Technologies</b>	"Denote systems and devices taken together".
<b>Concepts</b>	"May exist explicitly only in the designer's mind. They are unstated givens for the project, having been absorbed by osmosis, so to speak, by the engineer in the course of his development, perhaps even before entering formal engineering training. They had to be learned deliberately by the engineering community at some time, however, and form an essential part of design knowledge".

Table 3.2 Vincenti engineering knowledge categories and corresponding concepts  
(Vincenti W. G. 1990)

Engineering knowledge category	Corresponding concepts
<b>Fundamental design concepts</b>	<ul style="list-style-type: none"> <li>• About the design</li> <li>• Designers must know the operational principle of the device.</li> <li>• How the device works</li> <li>• Normal configuration</li> <li>• Normal design</li> </ul>

Table 3.2 Vincenti engineering knowledge categories and corresponding concepts  
(Vincenti W. G. 1990) (continued)

<b>Engineering knowledge category</b>	<b>Corresponding concepts</b>
<b>Criteria and specifications</b>	<ul style="list-style-type: none"> <li>• Specific requirements of an operational principle</li> <li>• General qualitative goals</li> <li>• Specific quantitative goals laid out in concrete technical terms</li> <li>• The design problem must be “well defined”.</li> <li>• Unknown or partially understood criteria</li> <li>• This task takes place at the project definition level in the design hierarchy.</li> <li>• Definition of technical specifications</li> </ul>
<b>Theoretical tools</b>	<ul style="list-style-type: none"> <li>• Mathematical methods and theories for making design calculations</li> <li>• Intellectual concepts for thinking about the design.</li> <li>• Precise and codifiable</li> <li>• They come mostly from deliberate research.</li> <li>• They are not sufficient by themselves.</li> </ul>
<b>Quantitative data</b>	<ul style="list-style-type: none"> <li>• Specify manufacturing processes for production</li> <li>• Display the detail for the device</li> <li>• Data essential for design</li> <li>• Obtained empirically</li> <li>• Calculated theoretically</li> <li>• Represented in tables or graphs</li> <li>• Precise and codifiable</li> <li>• They come mostly from deliberate research.</li> <li>• They are not sufficient by themselves.</li> </ul>

**Table 3.2** Vincenti engineering knowledge categories and corresponding concepts (Vincenti W. G. 1990) (continued)

Engineering knowledge category	Corresponding concepts
<p style="text-align: center;"><b>Practical considerations</b></p>	<ul style="list-style-type: none"> <li>• Theoretical tools and quantitative data are not sufficient. Designers also need practical considerations derived from experience.</li> <li>• Practical considerations are learned on the job, and not in school or from books.</li> <li>• Practical considerations are rarely documented.</li> <li>• Practical considerations are also derived from production and operation.</li> <li>• This knowledge is difficult to define.</li> <li>• This knowledge defies codification</li> <li>• A prototype must often be built to check the designer's work.</li> <li>• The practical consideration learned from operation is judgment.</li> <li>• Rules of thumb.</li> <li>• The practices from which these rules derive include not only design, but production and operation as well.</li> </ul>
<p style="text-align: center;"><b>Design instrumentalities</b></p>	<ul style="list-style-type: none"> <li>• Knowing how</li> <li>• Procedural knowledge</li> <li>• Ways of thinking</li> <li>• Skills based on judgment</li> <li>• Knowledge on how to carry out tasks</li> <li>• Must be part of any anatomy of engineering knowledge</li> </ul>

Table 3.3 Vincenti: engineering knowledge categories and description  
(Vincenti W. G. 1990)

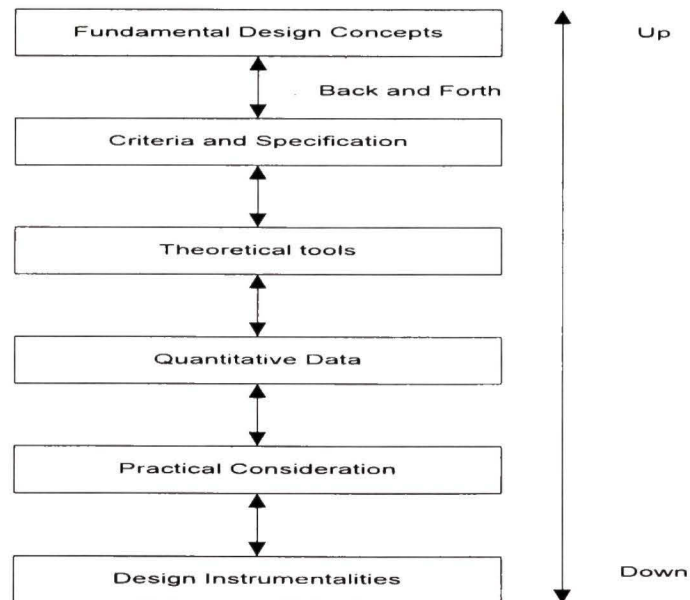
Engineering knowledge category	Description
<b>Fundamental design concepts</b>	Designers embarking on any normal design bring with them fundamental concepts about the device in question.
<b>Criteria and specification</b>	To design a device embodying a given operational principle and normal configuration, the designer must have, at some point, specific requirements in terms of hardware.
<b>Theoretical tools</b>	To carry out their design function, engineers use a wide range of theoretical tools. These include intellectual concepts as well as mathematical methods.
<b>Quantitative data</b>	Even with fundamental concepts and technical specifications at hand, mathematical tools are of little use without data for the physical properties or other quantities required in the formulas. Other kinds of data may also be needed to lay out details of the device or to specify manufacturing processes for production.
<b>Practical considerations</b>	To complement the theoretical tools and quantitative data, which are not sufficient. Designers also need less sharply defined considerations derived from experience.
<b>Design instrumentalities</b>	Besides the analytical tools, quantitative data and practical considerations required for their tasks, designers need to know how to carry out those tasks. How to employ procedures productively constitutes an essential part of design knowledge.

### 3.3 Vincenti's classification of engineering knowledge types

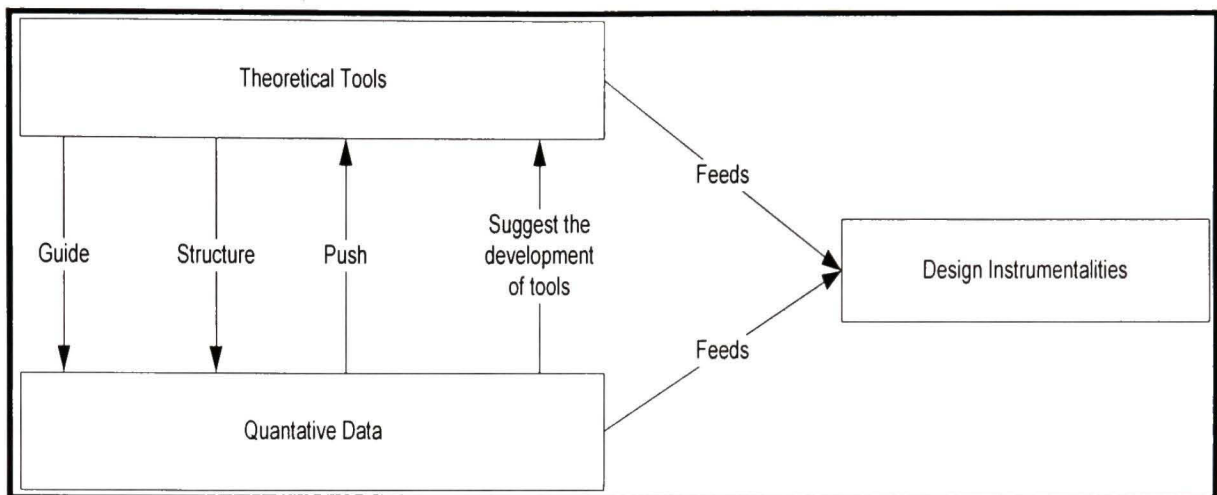
#### 3.3.1 Relationship between the various categories

Since the categories are not mutually exclusive, it is important to understand the relationships between them. An initial modeling of Vincenti's categories of engineering knowledge (Vincenti W. G. 1990) is presented in Figure 3.1. This figure illustrates that, in seeking a design solution, designers move up and down within categories, as well as back and forth from one category to another.

It can also be noted that three categories (theoretical tools, quantitative data and design instrumentalities) are related in the following manner: theoretical tools guide and structure the data, while quantitative data suggest and push the development of tools for their presentation and application – see Figure 3.2. Furthermore, both theoretical tools and quantitative data serve as inputs for design instrumentalities, while appropriate theoretical tools and quantitative data are needed for technical specifications.



**Figure 3.1 Vincenti's classification of engineering knowledge**



**Figure 3.2 Relationships between theoretical tools and quantitative data**

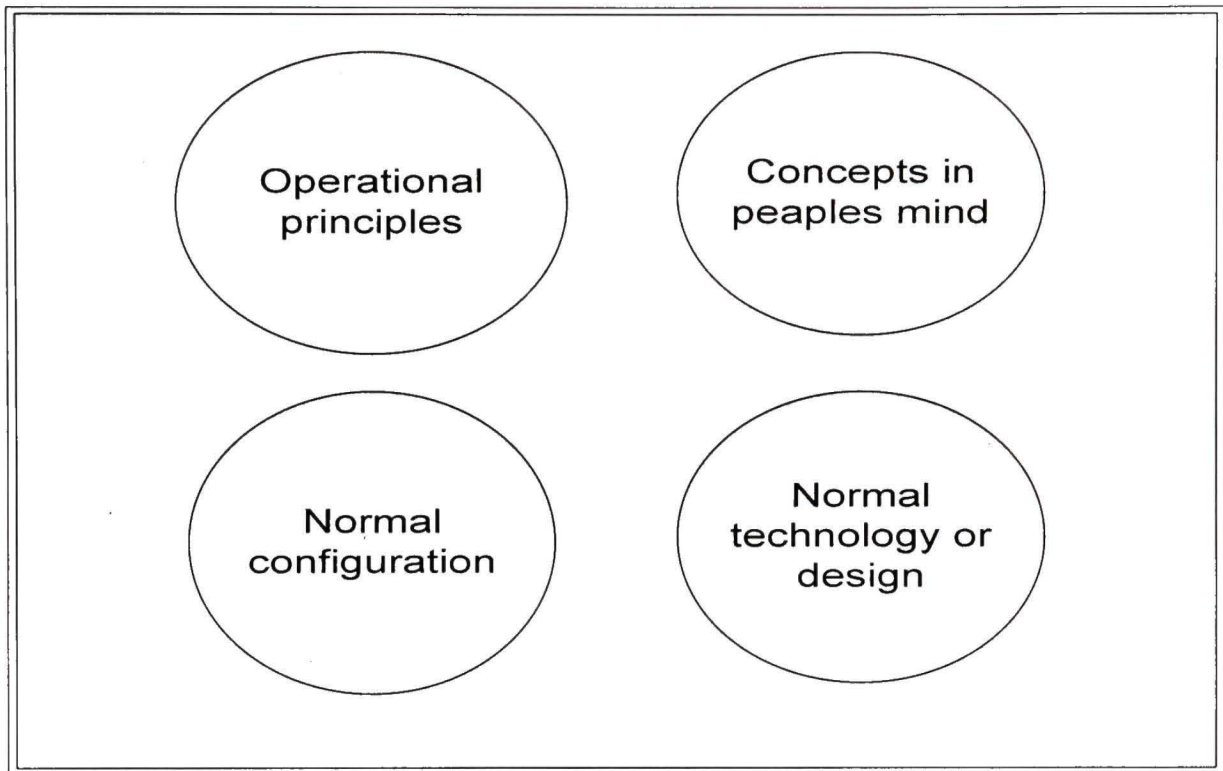
### 3.4 Vincenti's classification of engineering knowledge-type models

This section presents a detailed description of Vincenti's six categories of engineering knowledge (Vincenti W. G. 1990) and the related models for each. Vincenti stated that these categorizations of engineering knowledge are not exclusive, since some concepts of knowledge can be found in more than one category.

#### 3.4.1 Fundamental design concepts

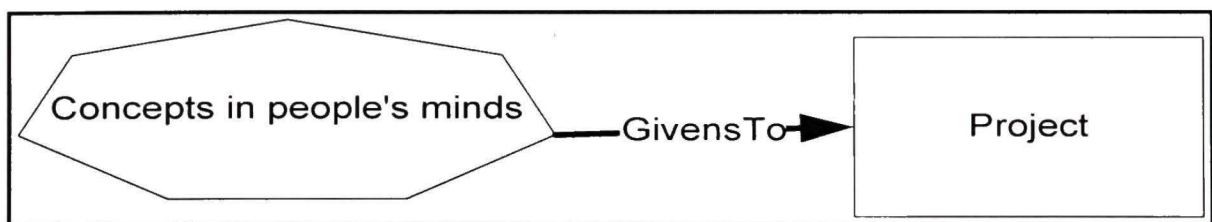
The goal of "fundamental design concepts", according to (Vincenti W. G. 1990), is as follows: "designers setting out on any normal design bring with them fundamental concepts about the device in question" which means the definition of fundamental concepts related to the device by the designer. Fundamental design concepts are composed of four elements in Figure 3.3: operational principles, normal configuration, normal technology and concepts in people's minds. At first, these concepts exist only in the designer's mind - Figure 3.4.





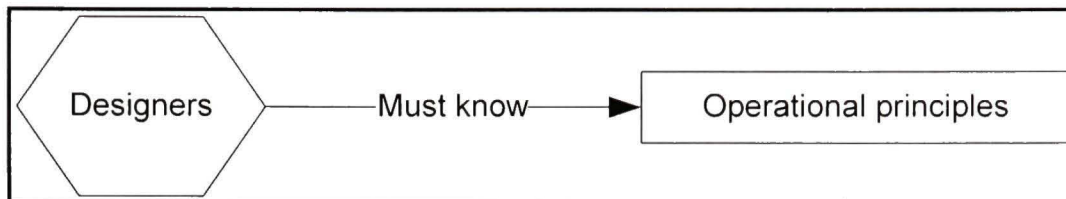
**Figure 3.3 Elements of a fundamental design concept**

**Concepts in people's minds** are inputs to the project – Figure 3.4

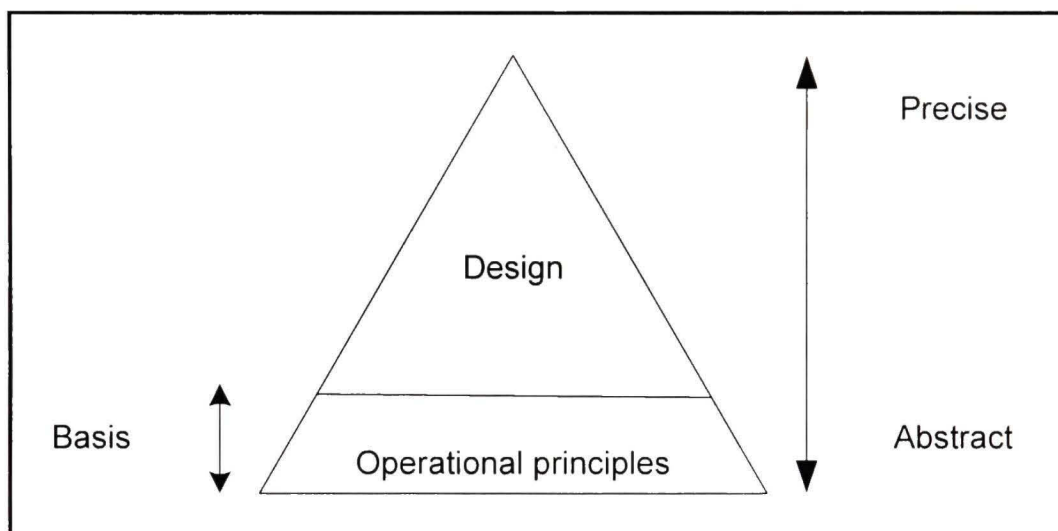


**Figure 3.4 Project inputs**

**Operational principles** define the essential fundamental concept of a device: “How its characteristic parts... fulfill their special functions in combination to [sic] an overall operation...” (Vincenti W. G. 1990). The operational principles must be known by the designers first – Figure 3.5 – and constitute the basic components for the design, whereas operational principles are abstract, and the design moves from abstract concepts to precise concepts – Figure 3.6.



**Figure 3.5 Designers initial knowledge**



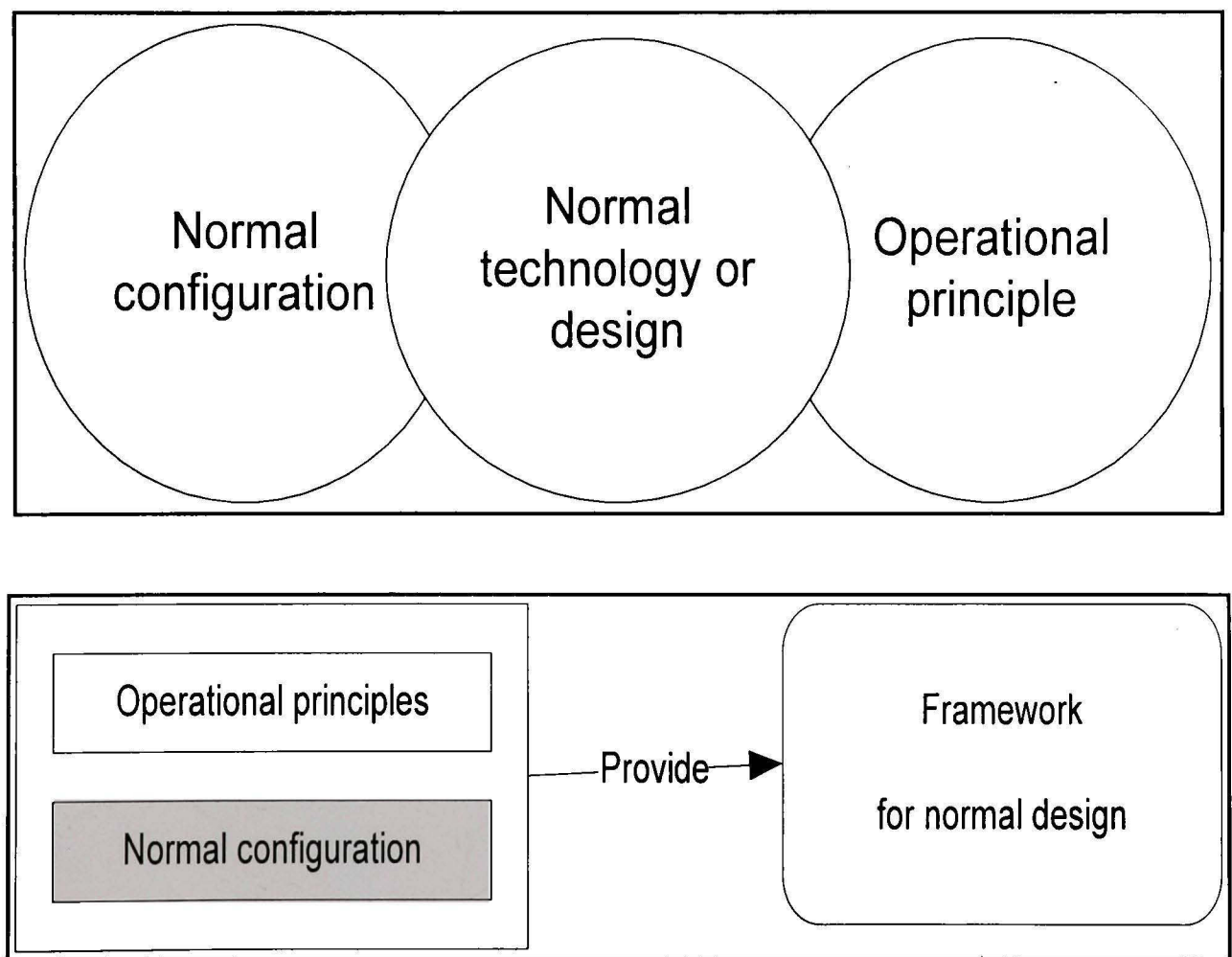
**Figure 3.6 Design pyramid**

**Normal configuration** is “the general shape and arrangement that are commonly agreed to best embody the operational principle” (Vincenti W. G. 1990) .

**Normal technology** is “the improvement of the accepted tradition or its application under new or more stringent conditions” (Vincenti W. G. 1990). Design, in Vincenti, “denotes both the content of a set of plans (as in the design for a new airplane) and the process by which those plans are produced” (Vincenti W. G. 1990). There are two types of design: normal design and radical design. The latter is a kind of design that is unknown to the designer, and where the designer is not familiar with the device itself. The designer does not know how the device should be arranged, or even how it works. The former is a traditional design, where the designer knows how the device works. The designer also knows the traditional features of the device. This type of design is also the design involved in normal technology, which was

mentioned earlier. In conclusion, “normal design is evolutionary rather than revolutionary” (Vincenti W. G. 1990). Finally, a normal configuration and operational principles together provide a framework for normal design – Figure 3.7.

In Vincenti, a normal technology, or design, is part of a normal configuration and of related operational principles.



**Figure 3.7 Relationships between normal configurations, operational Principles and normal design**

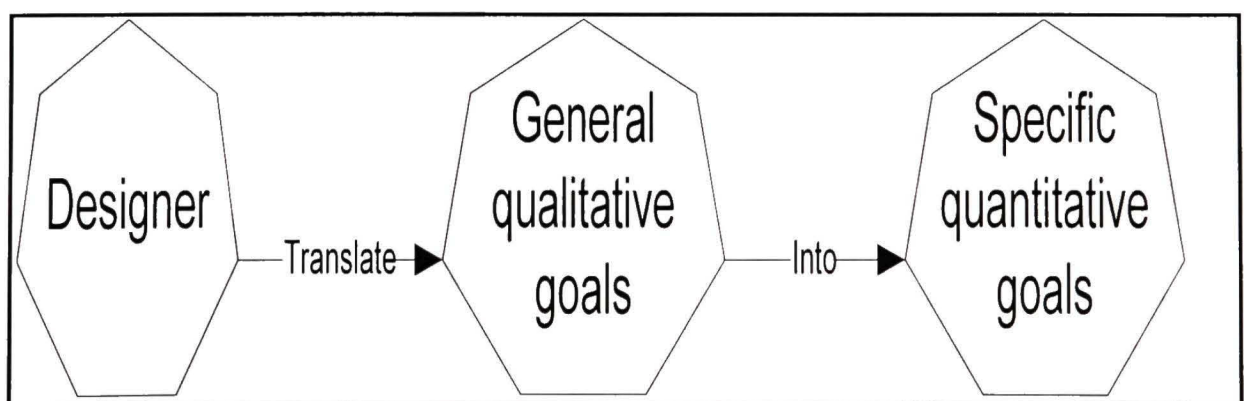
### 3.4.2 Criteria and specifications

The goal for “criteria and specifications” can be expressed as follows: “To design a device embodying a given operational principle and normal configuration, the designer must have, at some point, specific requirements in terms of hardware” (Vincenti W. G. 1990). The designer designs a device meeting specific requirements which include a given operational principle as well as a normal configuration. At first, the design problem must be well defined. Then, the designer translates general qualitative goals into specific quantitative goals -

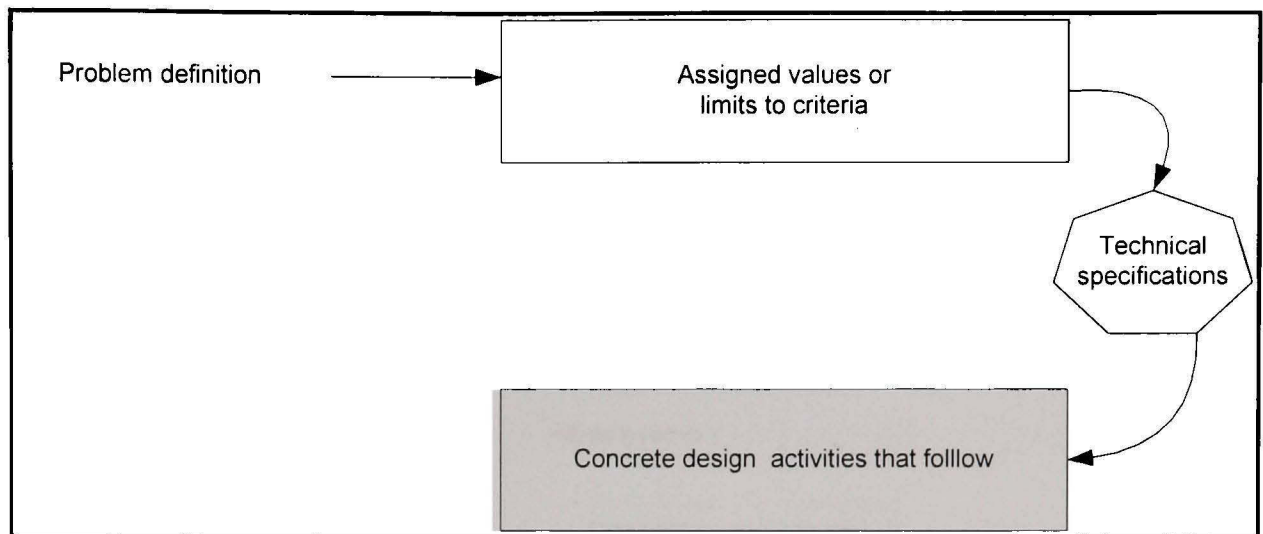
Figure 3.8: the designer assigns values or limits to the characteristics of the device which are crucial for engineering design allowing the designer to provide the details and dimensions of the device that will be given to the builder. Furthermore, the output at the problem definition level is used, in turn, as input to the remaining design activities that follow –

Figure 3.9. These specifications are more important where safety is involved, as in the case of aeronautical devices. The criteria on which the specifications are based become part of the accumulating body of knowledge about how things are done in engineering.

Finally, “criteria and specifications” exist as a category of knowledge only in engineering and not in science. In science, the aim is to understand: scientists do not need to have highly specified or concrete objectives. In engineering, by contrast, to design a device, criteria and specified goals are crucial.



**Figure 3.8 Designer’s goals**



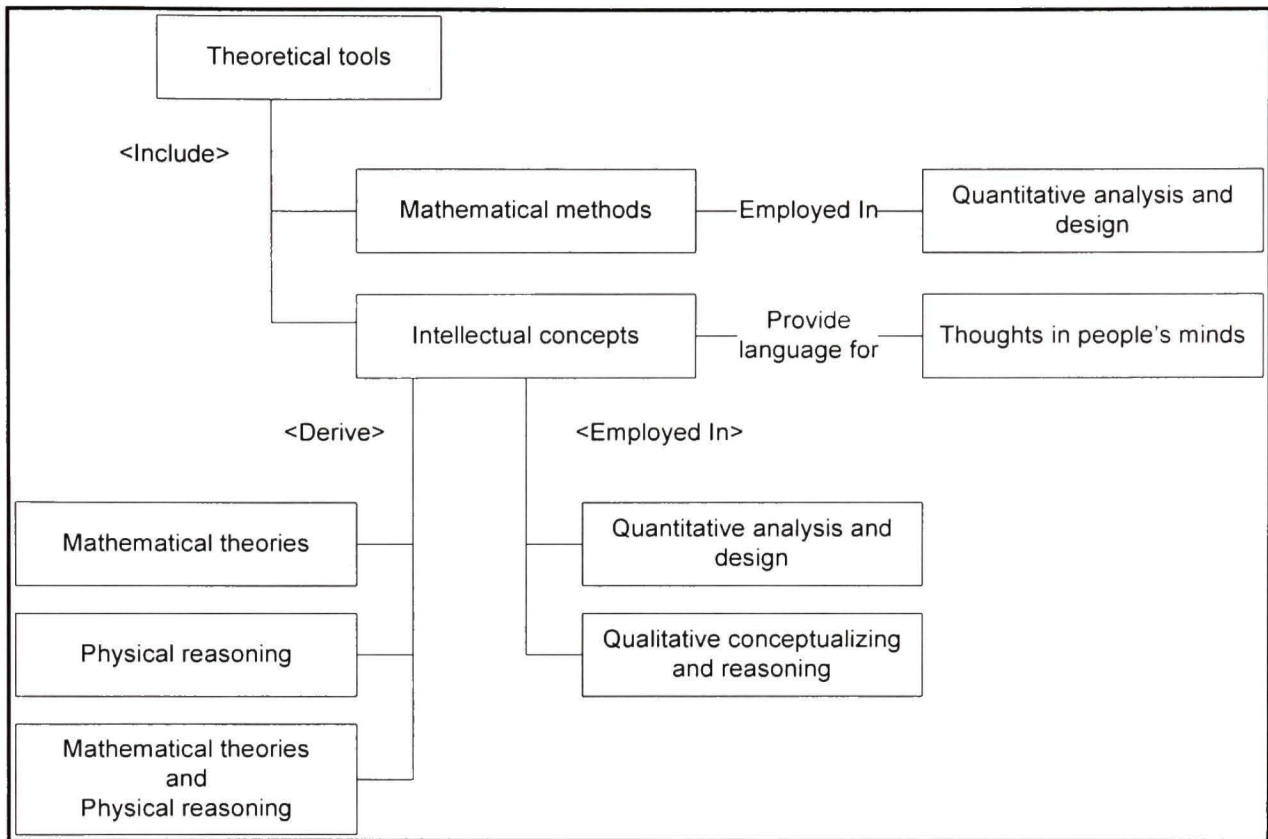
**Figure 3.9 Problem definition level output**

### 3.4.3 Theoretical tools

Theoretical tools are used by engineers to carry out their design. The goal of the “theoretical tools” category is expressed by Vincenti as follows: “To carry out their design function, engineers use a wide range of theoretical tools. These include intellectual concepts as well as mathematical methods” (Vincenti W. G. 1990). Figure 3.10 illustrates intellectual concepts (such as design concepts, mathematical methods and theories) for making design calculations. Both design concepts and methods are part of science.

In the first class of theoretical tools are mathematical methods and theories composed of formulas, either simple or complex, which are useful for quantitative analysis and design. This scientific knowledge must be reformulated to make it applicable to engineering.

The second class of theoretical tools are intellectual concepts which represent the language expressing those thoughts in people’s minds. These concepts are employed first in the qualitative conceptualization and reasoning that engineers have to perform before they carry out the quantitative analysis and design calculations.



**Figure 3.10 Theoretical tools model**

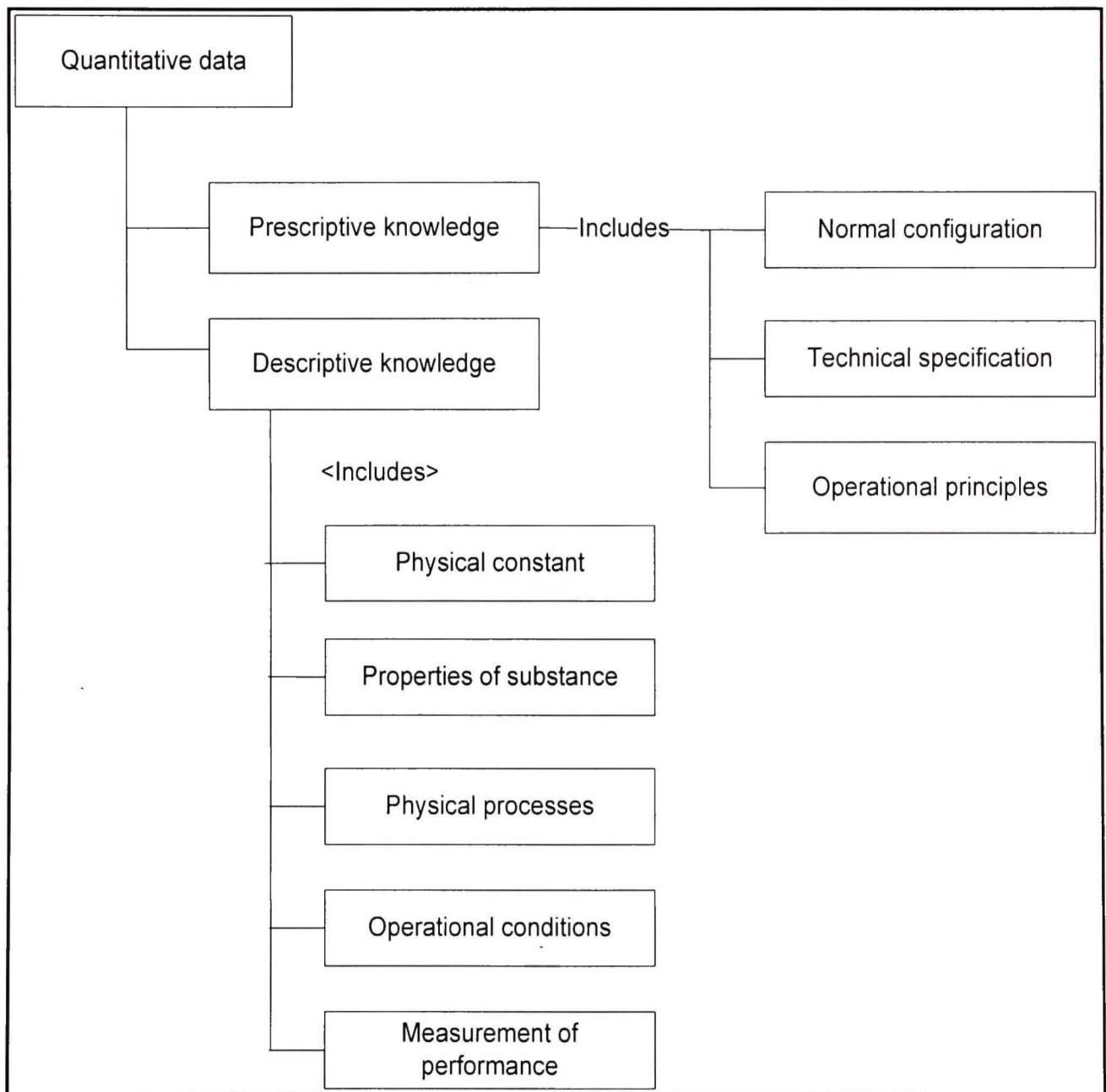
#### 3.4.4 Quantitative data

The goal of “quantitative data” is to lay down “the physical properties or other quantities required in the formulas. Other kinds of data may also be needed to lay out details of the device or to specify manufacturing processes for production” (Vincenti W. G. 1990). Besides fundamental concepts and technical specifications, the designers also need quantitative data to lay out the details of the device. These data can be obtained empirically, or in some cases they can be obtained theoretically and can be represented in tables or graphs.

These data are divided into two types of knowledge: prescriptive and descriptive. Descriptive knowledge is “knowledge of how things are” (Vincenti W. G. 1990) and includes physical constants, properties of substances and physical processes. In some situations, descriptive data refers to operational conditions in the physical world. Descriptive data can also include

measurement of performance. Prescriptive knowledge is “knowledge of how things should be to attain a desired end” (Vincenti W. G. 1990). An example might be: “In order to accomplish this or organize this, arrange things this way”.

Operational principles, normal configuration and technical specifications are prescriptive knowledge because they prescribe how a device should satisfy its objective – Figure 3.11.



**Figure 3.11 Quantitative data model**

### **3.4.5 Practical considerations**

According to Vincenti, the goal of “practical considerations” is “to complement the role of theoretical tools and quantitative data which are not sufficient. Designers also need for their work less sharply defined considerations derived from experience” (Vincenti W. G. 1990). This kind of knowledge is prescriptive in the way that it shows the designers how to proceed with the design to achieve their goal.

Vincenti refers to practical considerations as constituting non codifiable knowledge derived from experience, unlike theoretical tools and quantitative data which are very precise and codifiable because theoretical tools and quantitative data are derived from intentional research.

This category of engineering knowledge is needed by designers as a complement to theoretical tools and quantitative data. These practical considerations are learned on the job, rather than at school or from books. They are not to be formalized or programmed.

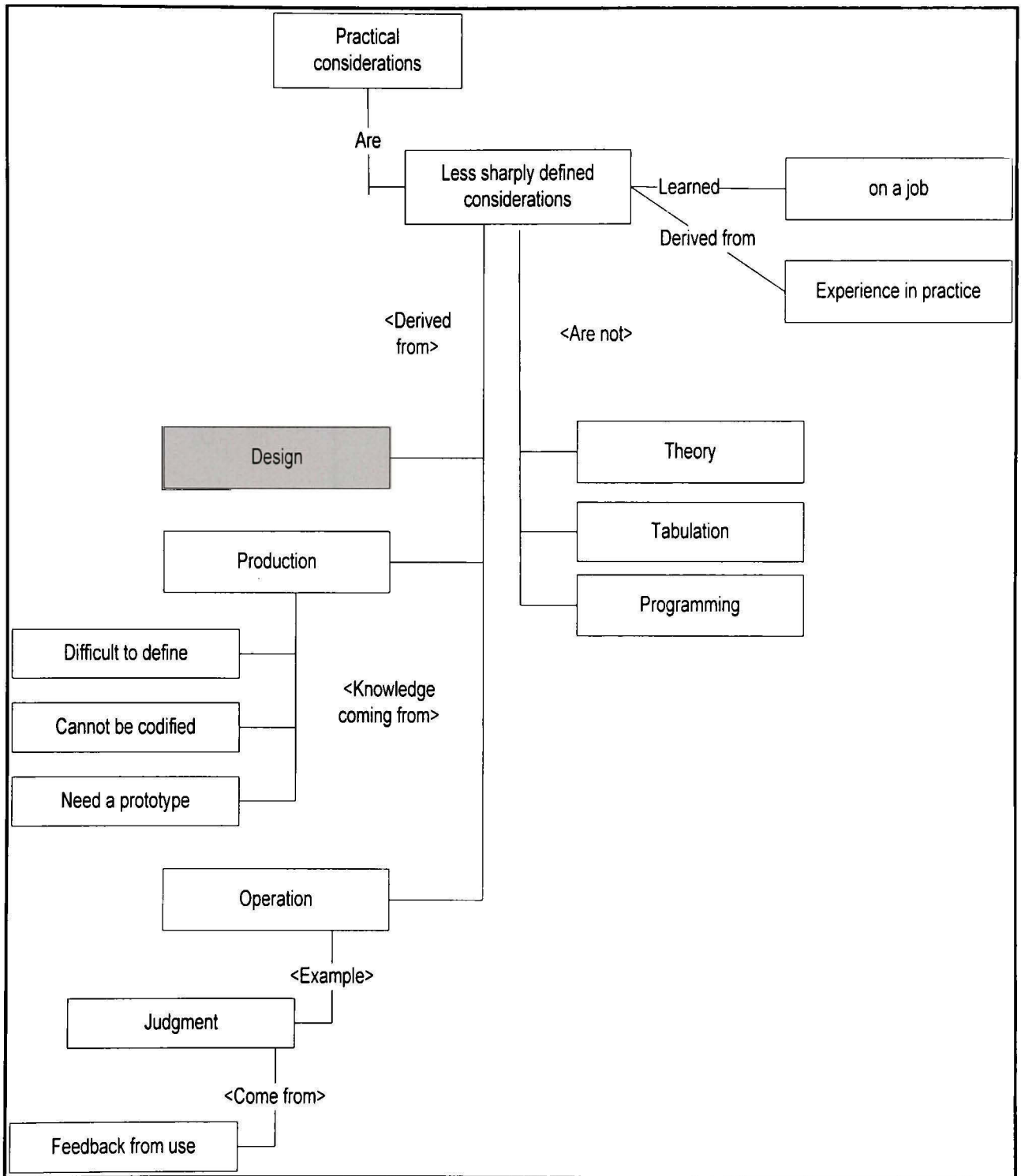
Practical considerations are derived from design, as well as from production and operation. The practical considerations derived from productions are not easy to define and cannot be codified, and a prototype is highly recommended to check the designer’s work. An example of a practical consideration from operation is the judgment that comes from the feedback resulting from use – Figure 3.12.

### **3.4.6 Design instrumentalities**

The goal of “design instrumentalities” in the engineering design process required for the engineer’s tasks is “to know how to carry out those tasks. How to employ procedure productively constitutes an essential part of design knowledge” (Vincenti W. G. 1990).

Having the analytical tools, quantitative data and practical considerations at hand, designers also need procedural knowledge to carry out their tasks, as well as to know how to employ these procedures.

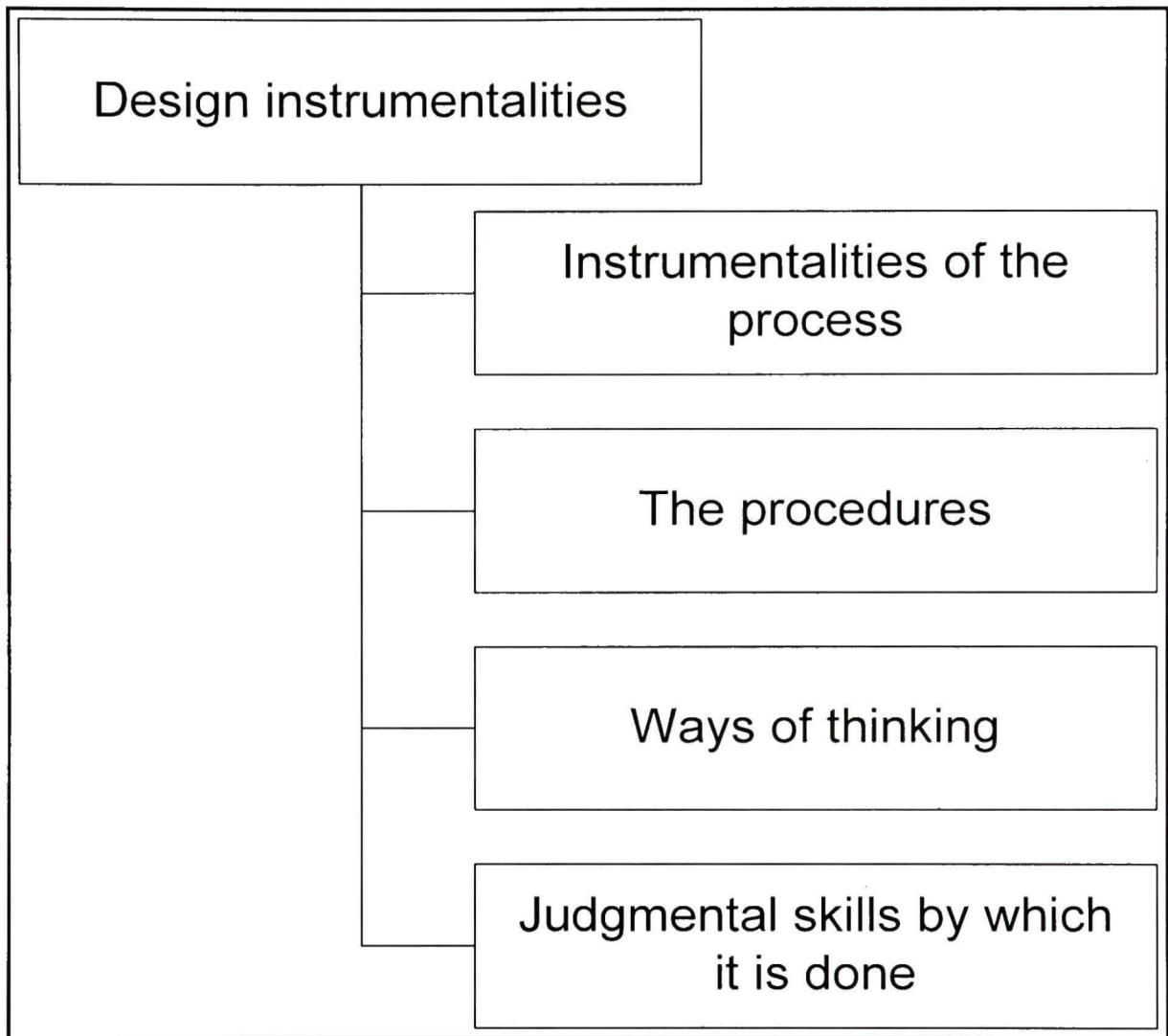




**Figure 3.12 Practical considerations model**

Design instrumentalities contain instrumentalities of the process, the procedures, judgment and ways of thinking. The latter are less tangible than procedures and more tangible than

judgment; an example of ways of thinking is “thinking by analogy” (Vincenti W. G. 1990). Judgment is needed to seek out design solutions and make design decisions – Figure 3.13.



**Figure 3.13: Design instrumentalities model**

### **3.5 The Design process**

#### **3.5.1 The engineering design process in Vincenti**

According to Vincenti, the engineering “design” concept “denotes both the content of a set of plans (as in the design for a new airplane) and the process by which those plans are

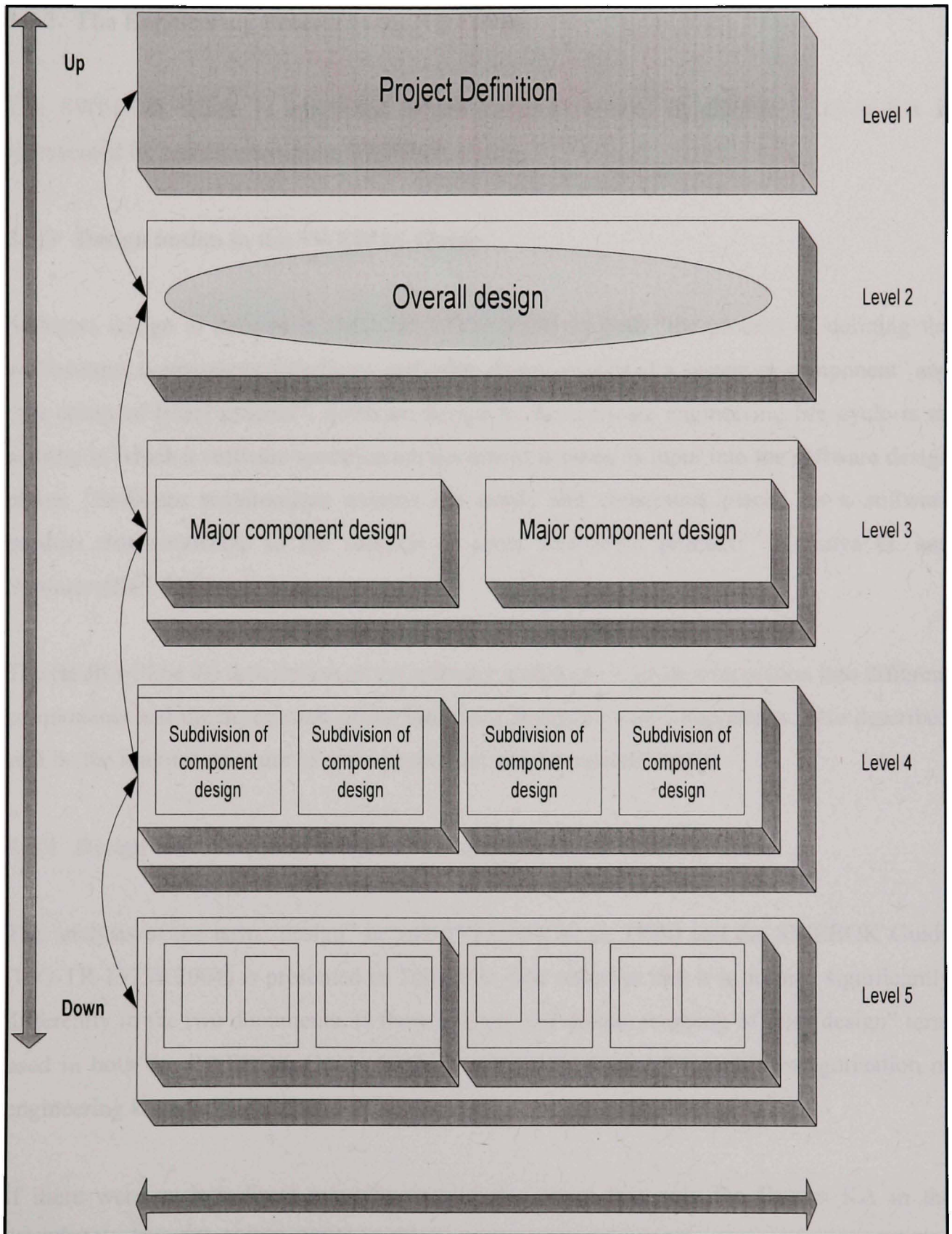
produced” (Vincenti W. G. 1990). In Vincenti’s view, design is an iterative and complex process which consists of plans for the production of a single entity, such as an airplane (device), how these plans are produced, and, finally, the release of these plans for production. Vincenti mentions that there are two types of design in engineering, normal and radical. In the former, the designer knows how the device works, how it should be arranged and what its features are. In the latter, the device is new to the engineer who is encountering it for the first time. Therefore, the engineer does not know how it works or how it should be organized.

Vincenti also mentions that design is a multilevel and hierarchical process. The designer starts by taking the problem as input. The design hierarchies start from the project definition level, located at the upper level of the hierarchy where problems are abstracted and unstructured. At the overall design level, the layout and the proportions of the device are set to meet the project definition. At level 3, the project is divided into its major components.

At level 4, each component is subdivided. At level 5, the subcomponents from level 4 are further divided into specific problems.

At the lower levels, problems are well defined and structured. The design process is iterative, both up and down and horizontally throughout the hierarchy. Vincenti’s view of the levels of design is modeled in

Figure 3.14. At each level of the hierarchy, a design can be either normal or radical.



**Figure 3.14: Modeling of the levels of the design hierarchy, as described in Vincenti**

### **3.5.2 The Engineering Process in the SWEBOK**

The SWEBOK Guide is composed of ten KAs as referred in chapter 1. Each KA is represented by one chapter in the SWEBOK Guide.

### **3.5.3 Design notion in the SWEBOK Guide**

Software design is defined in (ISO-TR-19759 2004) as both “the process of defining the architecture, components, interfaces, and other characteristics of a system or component” and “the result of [that] process”. Software design in the software engineering life cycle is an activity in which a software specification document is taken as input into the software design phase. “Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problem” (Kotonya G. and Sommerville I. 2000).

The result will be the description of the software architecture, its decomposition into different components and the description of the interfaces between those components. Also described will be the internal structure of each component and the related details.

### **3.5.4 Design KA: mapping between Vincenti and the SWEBOK Guide**

The analysis of the term “design” in both (Vincenti W. G. 1990) and the SWEBOK Guide (ISO-TR-19759 2004) is presented in Table 3.4. One observes that it is defined significantly differently in the two documents. Is there a direct and unique mapping of this “design” term used in both the SWEBOK Guide (ISO-TR-19759 2004) and Vincenti’s categorization of engineering knowledge (Vincenti W. G. 1990)?

If there were such a direct mapping, would this mean that only the Design KA in the SWEBOK (ISO-TR-19759 2004) could be mapped to Vincenti’s engineering knowledge (Vincenti W. G. 1990)? Or, alternatively, is the notion of design defined by (Vincenti W. G.

1990) different from the design concept in software engineering as defined in the SWEBOK Guide (ISO-TR-19759 2004)? And, if so, what is its scope within the SWEBOK Guide?

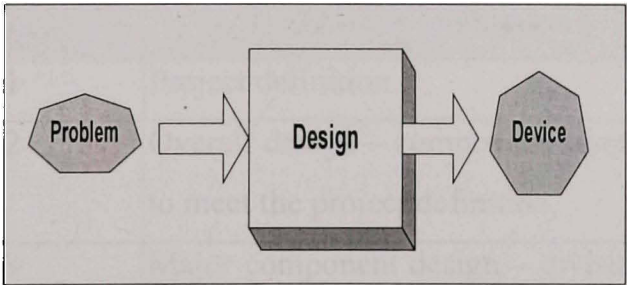
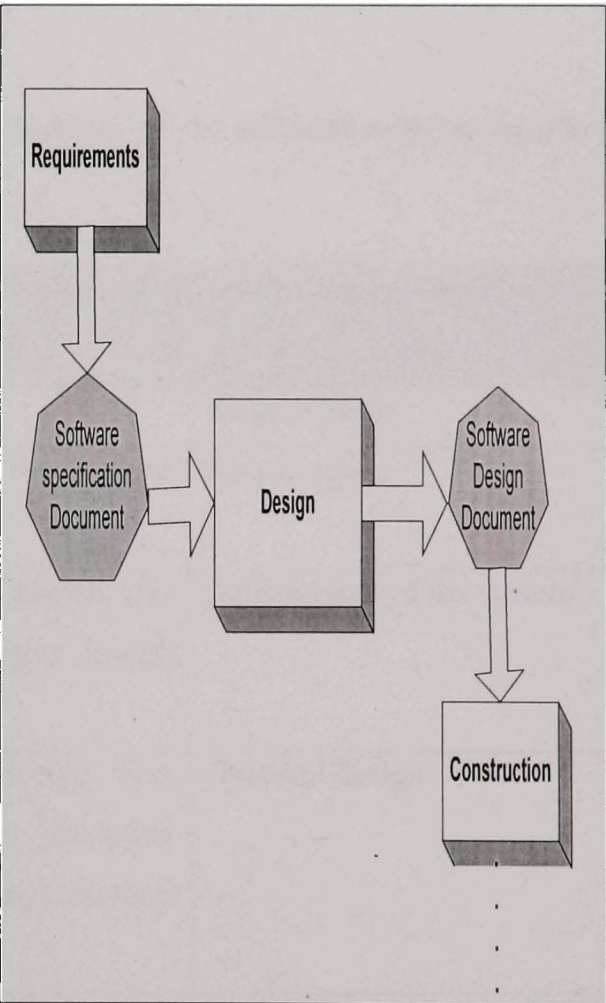
The definitions and descriptions of this term in both Vincenti and the SWEBOK Guide are presented in Table 3.4. One can note that it is defined significantly differently in the two documents; that is, design in engineering according to Vincenti is not limited to design as described in the SWEBOK Guide. In Vincenti, it goes far beyond the scope of the SWEBOK. Being composed of the whole of the software engineering life cycle, as illustrated in Figure 3.15, whereas all the activities of software life cycle, like the requirements phase, the design phase, the construction phase and the testing phase map to a single phase in the engineering cycle, that is, design. These activities do not necessarily take place in the same order. Testing in engineering starts right at the beginning, at the problem definition level, and goes on until the final release of the plans for production, while in the software engineering life cycle, as defined generically in the SWEBOK Guide, testing starts after the construction phase.

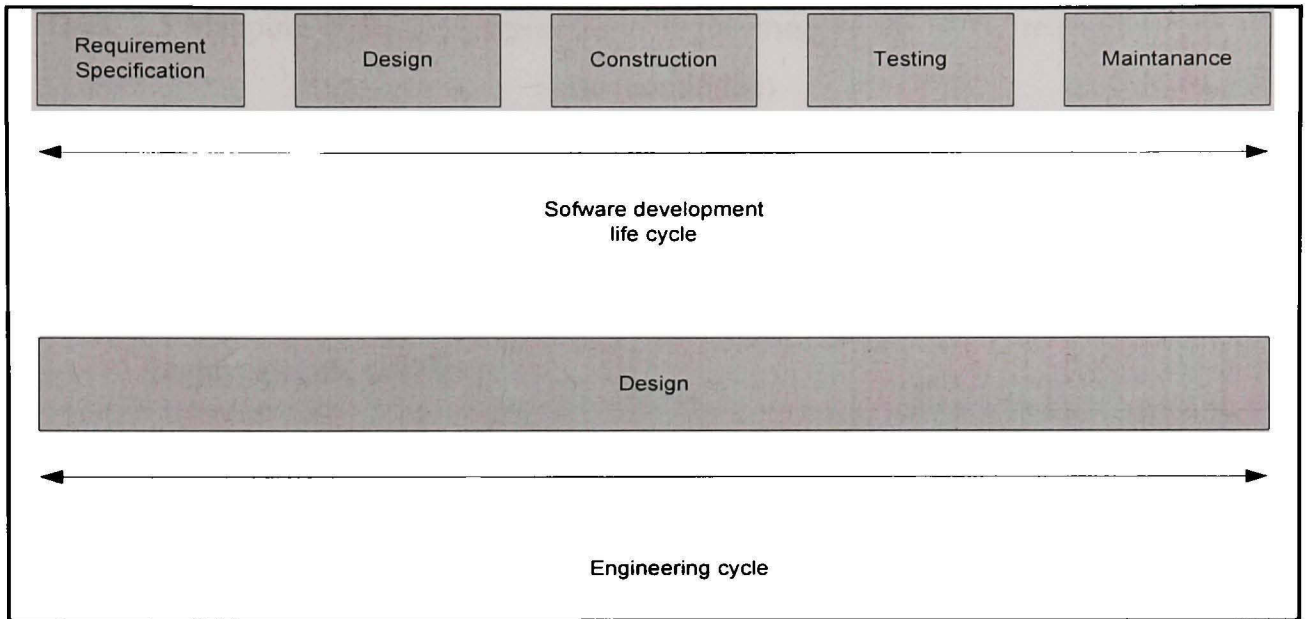
The detailed mapping between the different design levels in engineering and in the software engineering life cycle is presented in Table 3.5

Table 3.4 Design according to Vincenti vs. design in the software engineering life cycle

Design definition for engineering according to Vincenti	Design definition for software engineering
<p>Design, as defined by Vincenti:            “The content of a set of plans (as in the design for a new airplane)” and “the process by which those plans are produced.”</p>	<p>Design is defined in (ISO-TR-19759 2004) as both:            “The process of defining the architecture, components, interfaces, and other characteristics of a system or component” and “the result of [that] process.”</p>

Table 3.4 Design according to Vincenti vs. design in the software engineering life cycle  
(continued)

Design definition for engineering according to Vincenti	Design definition for software engineering
<p>Design, in the engineering life cycle is a process which starts by taking as input the problem, following a set of hierarchical levels. This process moves from problem definition to the production of a device as output.</p>	<p>Software design in the software engineering life cycle is an activity in which software requirements is taken as input to the software design phase for analysis. The result will be a precise description of the internal structure of the program.</p>
 <pre> graph LR     Problem{{Problem}} --&gt; Design[Design]     Design --&gt; Device{{Device}}     </pre>	 <pre> graph TD     Requirements[Requirements] --&gt; SSD{{Software specification Document}}     SSD --&gt; Design[Design]     Design --&gt; SDD{{Software Design Document}}     SDD --&gt; Construction[Construction]     </pre>



**Figure 3.15 Design according to Vincenti vs. design in the software engineering life cycle**

**Table 3.5 Mapping of the design process in engineering vs. the software engineering life cycle**

Levels	Description of the design process in engineering	Software engineering life cycle
1	Project definition	Requirements
2	Overall design – component layout of the airplane to meet the project definition.	Specification
3	Major component design – division of project into wing design, fuselage design, landing gear design, electrical system design, etc.	Architecture of the system
4	Subdivision of areas of component design from level 3 according to the engineering discipline required (e.g. aerodynamic wing design, structural wing design, mechanical wing design)	Detailed design



Table 3.5 Mapping of the design process in engineering vs. the software engineering life cycle (continued)

Levels	Description of the design process in engineering	Software engineering life cycle
5	Further division of the level 4 categories into highly specific problems	Construction

### 3.6 Mapping results for the Vincenti classification of engineering knowledge

This section presents the related results for the analysis of the “Software requirements” KA, the “Software design” and the “Software construction” knowledge areas of the SWEBOK Guide from an engineering perspective.

To analyze the breakdown related to these knowledge areas, the Vincenti classification of engineering knowledge is used to gain further insights on the level of maturity of this KA from an engineering viewpoint.

This analysis is based on the models of engineering knowledge described earlier. These models give one a descriptive analysis of the various key elements contained in each of the corresponding software engineering knowledge areas.

This analysis allows one to make an appropriate mapping among the different categories of the engineering knowledge and the knowledge areas related to the SWEBOK Guide. As a result, this analysis looks into these KA from an engineering perspective.

The different mapping between the categories of engineering knowledge and the 3 SWEBOK KAs are presented in Annex I.

As a result of this mapping a new breakdown is proposed for the “Software requirements”, “Software design” and in “Software construction” KAs based on the Vincenti categories of engineering knowledge types - see Annex II.

Next is presented an analysis of the engineering content within the first three KAs of the SWEBOK Guide using Vincenti categories of engineering knowledge, that is: “Software requirements” in table 3.6, “Software design” in table 3.7 and “Software construction” in table 3.8.

Table 3.6 Software Requirement in the SWEBOK Guide

Software Requirements KA from SWEBOK using Vincenti Categories of engineering knowledge						
Software Requirements KA	Fundamental Design concepts	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Considerations	Design Instrumentalitie
Software Requirements Fundamentals	- Definition of software Requirements - Product and process requirements	- System & software requirements -Functional & Non-functional - Emergent properties				
Requirements process	Process models	Process quality & improvement				- Process support & management - Process actors
Requirements elicitation					Requirements sources	Elicitation techniques
Requirements analysis	Architectural design and requirements allocation	Requirements classification	Conceptual modeling		Requirement negotiation	

Table 3.6 Software Requirement in the SWEBOK Guide (continued)

Software Requirements KA from SWEBOK using Vincenti Categories of engineering knowledge						
Software Requirements KA	Fundamental Design concepts	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Considerations	Design Instrumentalitie
Requirements specification		<ul style="list-style-type: none"> <li>- System definition document</li> <li>- System Requirements specification</li> <li>- System Requirements specification</li> </ul>				
Requirements validation			Model validation			<ul style="list-style-type: none"> <li>- Requirement reviews</li> <li>- Acceptance test</li> <li>- Prototyping</li> </ul>
Practical considerations		Requirement Attributes			<ul style="list-style-type: none"> <li>- Iterative nature of the requirement process</li> <li>-Requirements tracing</li> <li>-Measuring requirement</li> </ul>	Change management

Table 3.7 Software Design in the SWEBOK Guide

Software Design KA		Software Design KA from SWEBOK using Vincenti Categories of engineering knowledge						
KA	Fundamental Design	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Consideratio	Design Instrumental		
<b>Software design fundamentals</b>	<ul style="list-style-type: none"> <li>-General design concepts</li> <li>-The context of software design</li> <li>-The software design process</li> <li>-Enabling techniques</li> </ul>							
<b>Key issues in software design</b>	<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling &amp; fault tolerance</li> <li>-Interaction presentation</li> </ul>	<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling and fault tolerance</li> <li>-Interaction presentation</li> <li>-Data persistence</li> </ul>	<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling and fault tolerance</li> <li>-Interaction presentation</li> <li>-Data persistence</li> </ul>					
<b>Software structure and architecture</b>	<ul style="list-style-type: none"> <li>-Architectural structure and viewpoint</li> <li>-Families of programs and frameworks</li> <li>-Architectural style (Macro architectural patterns)</li> </ul>				-Design Patterns (Micro architectural patterns)	-Architectural style (Macro architectural patterns)		

Table 3.7 Software Design in the SWEBOK Guide (continued)

Software Design KA from SWEBOK using Vincenti Categories of engineering knowledge						
Software Design KA	Fundamental Design	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Consideratio	Design Instrumentalit
Software design quality analysis and evaluation		-Quality attributes -Measures	-Measures -Quality analysis and evaluations techniques		-Quality analysis and evaluations techniques	-Quality analysis and evaluations techniques
Software design notations			-structural descriptions (Static view) - Behavior description (dynamic view)			-structural descriptions (Static view) - Behavior description (dynamic view)
Software design strategies and methods		-General strategies -Function-oriented structured design -Object oriented design -Data structure centered design -Component			-General strategies -Function oriented structured -Object oriented design -Data structure centered design -Component based design	-General strategies -Function-oriented structured design -Object oriented design -Data structure centered design -Component based design

Table 3.8 Software Construction KA from SWEBOK using Vincenti Categories of engineering knowledge

Software Construction KA from SWEBOK using Vincenti Categories of engineering knowledge																			
Software Construction KA	Software Construction KA from SWEBOK using Vincenti Categories of engineering knowledge																		
	<table border="1"> <tr> <th>Fundamental Design</th> <th>Criteria and specification</th> <th>Theoretical Tools</th> <th>Quantitative Data</th> <th>Practical Consideratio</th> <th>Design Instrumentalit</th> </tr> </table>	Fundamental Design	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Consideratio	Design Instrumentalit												
Fundamental Design	Criteria and specification	Theoretical Tools	Quantitative Data	Practical Consideratio	Design Instrumentalit														
<p><b>Software construction fundamentals</b></p>	<table border="1"> <tr> <td>-Minimizing complexity</td> <td>-Standards in construction</td> <td>-Standards in construction</td> <td></td> <td>-Constructing for verification</td> <td>-Constructing for verification</td> </tr> <tr> <td>-Standards in construction</td> <td>-Anticipating change</td> <td></td> <td></td> <td></td> <td>-Standards in construction</td> </tr> </table>	-Minimizing complexity	-Standards in construction	-Standards in construction		-Constructing for verification	-Constructing for verification	-Standards in construction	-Anticipating change				-Standards in construction						
-Minimizing complexity	-Standards in construction	-Standards in construction		-Constructing for verification	-Constructing for verification														
-Standards in construction	-Anticipating change				-Standards in construction														
<p><b>Managing construction</b></p>	<table border="1"> <tr> <td>-Construction planning</td> <td></td> <td>-Construction measurement</td> <td></td> <td></td> <td>-Construction planning</td> </tr> <tr> <td>-Construction models</td> <td></td> <td>-Construction models</td> <td></td> <td></td> <td>-Construction models</td> </tr> <tr> <td></td> <td></td> <td>-Construction planning</td> <td></td> <td></td> <td></td> </tr> </table>	-Construction planning		-Construction measurement			-Construction planning	-Construction models		-Construction models			-Construction models			-Construction planning			
-Construction planning		-Construction measurement			-Construction planning														
-Construction models		-Construction models			-Construction models														
		-Construction planning																	
<p><b>Practical consideration</b></p>	<table border="1"> <tr> <td>-Construction design</td> <td></td> <td>-Construction design</td> <td></td> <td>-Reuse</td> <td>-Construction languages</td> </tr> <tr> <td>-Integration</td> <td></td> <td></td> <td></td> <td>-Construction quality</td> <td>-Coding</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>- Construction testing</td> <td>- Construction quality</td> </tr> </table>	-Construction design		-Construction design		-Reuse	-Construction languages	-Integration				-Construction quality	-Coding					- Construction testing	- Construction quality
-Construction design		-Construction design		-Reuse	-Construction languages														
-Integration				-Construction quality	-Coding														
				- Construction testing	- Construction quality														

### 3.6.1 Software requirements: Vincenti's view point

Table 3.6 presents the mapping between the “Software requirements” KA and the Vincenti categories of engineering knowledge types. From table 3.6 one observes that:

- Some of the “Software requirements fundamentals”, “Process” and “Analysis” topics map with “Fundamental design concepts” category of engineering knowledge.
- Some of the “Software requirements fundamentals”, “Process”, “Analysis”, “Specification” and “Practical considerations” topics map with “Criteria and specification” category of engineering knowledge.
- Some of “Requirements analysis” and “Validation” topics map with “Theoretical tools” category of engineering knowledge.
- Some of “Requirement elicitation”, “Analysis”, and “Practical considerations” topics map with “Practical considerations” category of engineering knowledge.
- Some of “Requirements process”, “Validation” and “Practical considerations” topics map with “Design instrumentalities” category of engineering knowledge.

### 3.6.2 Software design: Vincenti view

Table 3.7 presents the mapping between the “Software design” KA and the Vincenti categories of engineering knowledge types:

- Some of the “Software design fundamentals”, “Key issues in software design” and “Software structure and architecture” topics map with Vincenti “Fundamental design concepts” category of engineering knowledge.
- Some parts of “Key issues in software design” topics map with “Criteria and specification” and “Theoretical tools” categories of engineering knowledge.
- Some parts of “Software structure and architecture” topics map with “Practical considerations” and “Design instrumentalities”.
- “Software design quality analysis and evaluation” topics map with all Vincenti categories of engineering knowledge except for the “Fundamental design concepts” and “Quantitative data”.



- “Software design notations” map with “Theoretical tools” and “Design instrumentalities” categories of engineering knowledge.
- “Software design strategies and methods” map with “Criteria and specification”, “Practical considerations” and “Design instrumentalities”.

### **3.6.3 Software construction: Vincenti view**

Table 3.8 presents the mapping between the “Software construction” KA and the Vincenti categories of engineering knowledge types:

- Some of the “Software construction fundamentals” map with all categories of engineering knowledge except for “Quantitative data”.
- Some of “Managing construction” subarea map with “Fundamental design concepts”, “Theoretical tools” and “Design instrumentalities” Vincenti categories of engineering knowledge.
- Some of “Practical considerations” topics in “Software construction” KA map with “Fundamental design concepts”, “Theoretical tools”, “Practical considerations” and “Design instrumentalities” Vincenti category of engineering knowledge.

## **3.7 Analysis using the Vincenti classification of engineering knowledge**

This section presents the analysis of the related results presented earlier for the three knowledge areas. This analysis can provide useful insights from an engineering perspective into these KAs and helps categorize the knowledge contained in the “Software requirements”, “Software design” and “Software construction” KAs of the SWEBOK Guide, for instance, covering all the categories of engineering knowledge from an engineering viewpoint, and does not mean that this results analysis are complete and inclusive as follows:

- Most of the “Software requirements” engineering knowledge is concentrated in “Fundamental design concepts” and “Criteria and specifications” categories of engineering knowledge. The other parts of the “Software requirements” knowledge are in

“Practical considerations” and “Design instrumentalities” categories of engineering knowledge.

- The “Software requirements” knowledge is still lacking engineering knowledge from the “Theoretical tools” and “Quantitative data” categories.
- The “Software design” KA topics map with all Vincenti categories except “Quantitative data” where some data collected from design measurements are lacking.
- The “Software construction” KA in the SWEBOK has a full mapping with “Fundamental design concepts”, “Theoretical tools”, and “Design instrumentalities”.

### **3.8 Summary**

The SWEBOK Guide (ISO-TR-19759 2004) documents an international consensus on ten software engineering KAs within what is referred to as an engineering discipline. Software engineering, as a discipline, is certainly not yet as mature as other engineering disciplines, and some authors have even challenged the notion that software engineering is indeed engineering. The work presented in this chapter has involved investigating this engineering perspective, first by analyzing the Vincenti classification of engineering knowledge, and second by comparing the design concept in Vincenti vs. the design concept in the SWEBOK Guide.

The result of this analysis shows that the design issue in Vincenti is not limited to the design issue in the SWEBOK Guide. Design in engineering according to Vincenti is not limited to design as described in the SWEBOK Guide, going beyond, in that it is composed of the whole of the software engineering life cycle.

Finally, the SWEBOK “Software requirements”, “Software design” and “Software construction” KA were used as examples and analyzed using Vincenti categories from an engineering perspective.

This analysis has shown that almost all the categories of engineering knowledge described by Vincenti are present in these KAs of the SWEBOK addressing the full coverage of almost all related engineering-type knowledge.

## CHAPTER 4

### SOFTWARE ENGINEERING PRINCIPLES: DO THEY MEET ENGINEERING CRITERIA?

#### 4.1 Introduction

As mentioned in a literature review covering the previous 19 years on exploring the candidate fundamental principles of software engineering, 308 distinct proposals of fundamental principles were identified (Séguin N. 2006). These 308 distinct proposals were then analyzed against a set of criteria related to the specific concept of “principle”, following which only 34 were recognized as bona fide CFP (Séguin N. 2006). This study did not, however, include within its research scope, an analysis of these candidates from an engineering perspective.

This chapter presents phase 3: an analysis of these 34 CFP from an engineering perspective. One of the challenges of this research phase, of course, was to figure out what criteria should be verified from an engineering perspective since, in the traditional engineering literature, such criteria are not explicitly described. This chapter documents the approach selected to identify these verification criteria, as well as what was found when these criteria were applied to the set of 34 CFP.

This chapter is organized as follows: Section 4.2 presents the analysis methodology selected. Section 4.3 identifies the engineering criteria. Section 4.4 presents the verification against the two sets of criteria. Section 4.5 presents an analysis and consolidation using both sets of criteria. Section 4.6 presents the external verification. Finally, section 4.7 presents a discussion and future research directions.

## 4.2 Analysis methodology

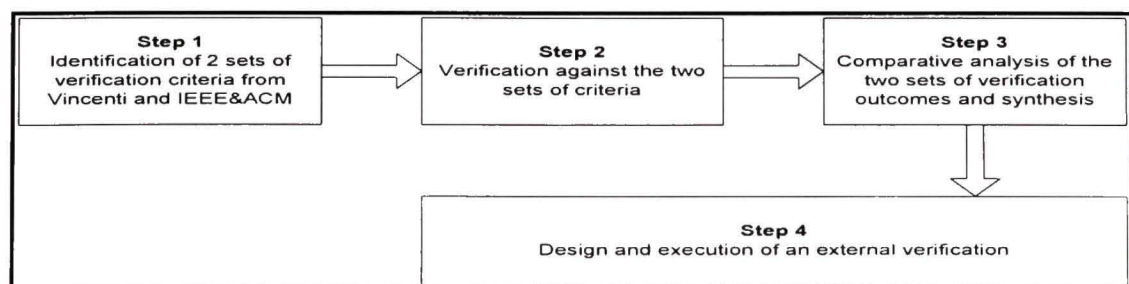
The scope of the criteria used in (Séguin N. 2006) was limited to the concept of “principles” and did not include the specificities of engineering concepts themselves.

The list of 34 CFP (Séguin N. 2006) constitutes the input to the analysis process required to verify whether or not they are indeed bona fide engineering principles. More specifically, the research issue addressed in this chapter is: which of these 34 CFP are indeed software engineering fundamental principles (hereinafter referred to as the FP)?

To conduct such a verification, engineering criteria must be available, but no related work could be identified. The first challenge was then to determine verification criteria from an engineering perspective. To tackle this issue, it was necessary first to study the epistemology of engineering. For that purpose, two sources, Vincenti, the author of the book, “What engineers know and how they know it” (Vincenti W. G. 1990) and the joint IEEE-ACM software engineering curriculum (IEEE and ACM, 2004) were selected:

- (Vincenti W. G. 1990) has identified a number of engineering knowledge types as key to the engineering disciplines and from which engineering criteria can be derived.
- The (IEEE and ACM, 2004) have documented a set of topics within their joint software engineering curriculum and from which engineering criteria can be derived.

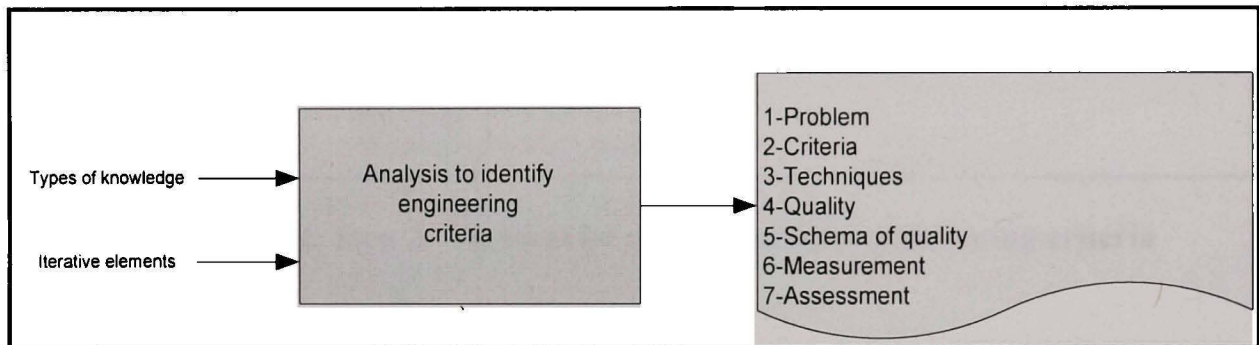
The approach designed for identifying relevant criteria and applying them to the set of 34 CFP (Séguin N. 2006) consists of four steps – see Figure 4.1:



**Figure 4.1 The Four-steps verification process**

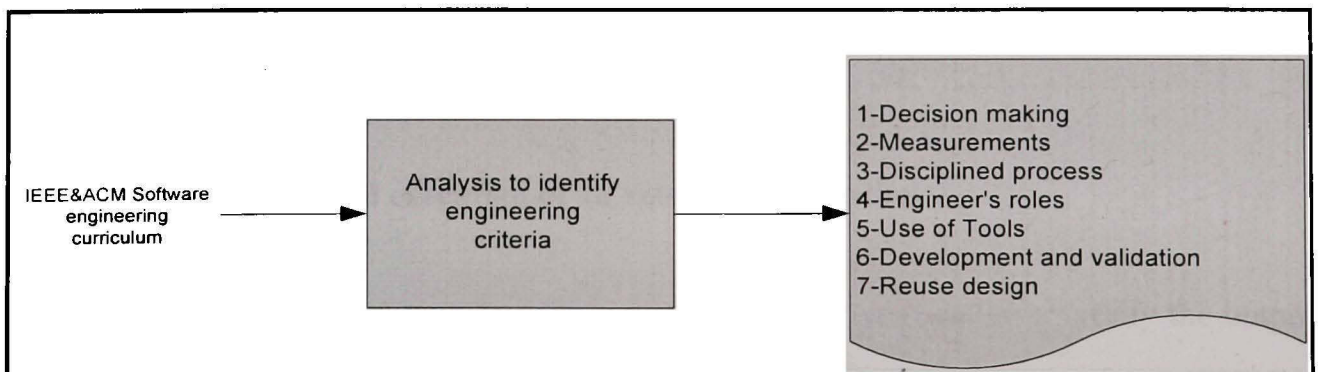
#### 4.2.1 Step 1: Identification of two sets of verification criteria.

This step consists of the identification of criteria which would be relevant to any engineering discipline. Such criteria could have been taken either 'as is', when identified and defined expressively, or derived, when documented only in an implicit manner. The inputs to this step are the two sources of information identified from the literature work and the outputs are: the two sets of criteria derived from (Vincenti W. G. 1990) and from the (IEEE and ACM, 2004) joint software engineering curriculum. The criteria identification phase based on (Vincenti W. G. 1990) is summarized in Figure 4.2 which shows its inputs and its outputs.



**Figure 4.2 Identification of Vincenti engineering criteria (Vincenti W. G. 1990)**

The criteria identification phase based on the (IEEE and ACM, 2004) criteria is summarized in Figure 4.3 which shows its inputs and its outputs.

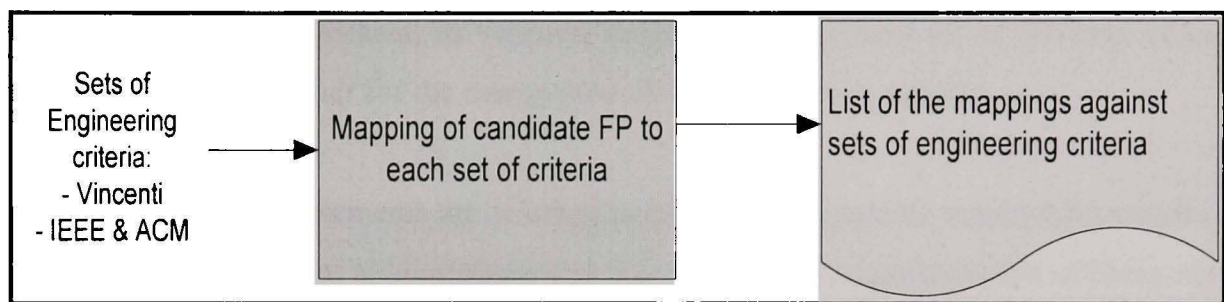


**Figure 4.3 Identification of the IEEE-ACM engineering criteria (IEEE and ACM, 2004)**

#### 4.2.2 Step 2: Verification execution

The 34 candidate FP are taken as inputs in the second step and analyzed next with respect to the two sets of engineering criteria identified in step 1.

The output will be the FP that have at least one direct mapping and those that have only an indirect mapping to either (Vincenti W. G. 1990) or to the (IEEE and ACM, 2004) engineering criteria. This second step is illustrated in Figure 4.4.



**Figure 4.4: Step 2: Verification process against engineering criteria**

#### 4.2.3 Step 3: Analysis and selection

Step 3 carried out the analysis across each set of engineering criteria.

This step identifies those candidate FP that meet engineering criteria from both sets of criteria, and those that do not. For instance, the candidate FP that meet only the Vincenti criteria (Vincenti W. G. 1990) and the candidate FP that only meet the IEEE-ACM criteria (IEEE and ACM, 2004) are analyzed to check whether or not they can be identified from the FP that are recognized as engineering FP.

#### 4.2.4 Step 4: Design and execution of an external verification

Step 4 carried out the design and execution of an external verification to verify the output provided by step 3, that is, the selected set of engineering FP.

### 4.3 Identification of engineering criteria: step 1

#### 4.3.1 Vincenti

(Vincenti W. G. 1990) has distinguished seven elements for engineering which he referred to as “interactive elements” and which he selected prior to categories of engineering knowledge types. These elements show the “epistemological structure of the engineering learning process” (Vincenti W. G. 1990) based on the analysis of the five aeronautical case studies. These seven elements represent, in Vincenti’s opinion, a necessary set of different elements that interact with each other for the completion of an engineering activity.

These seven interactive elements are referred to here as the Vincenti engineering criteria and are listed in Table 4.1. The abbreviations that were selected to represent each of these criteria are listed in the right-hand column of Table 4.1.

Table 4.1 Engineering criteria identified in Vincenti

ID.	Vincenti engineering criteria	Abbreviation
1	Recognition of a problem	<b>Problem</b>
2	Identification of concepts and criteria	<b>Criteria</b>
3	Development of instruments and techniques	<b>Techniques</b>
4	Growth and refinement of opinions regarding desirable qualities	<b>Quality</b>
5	Combination of partial results from 2, 3 and 4 into practical schema for research	<b>Testing</b>
6	Measurement of characteristics	<b>Measurement</b>
7	Assessment of results and data	<b>Assessment</b>



### 4.3.2 IEEE and ACM joint curriculum

The IEEE Computer Society (IEEE-CS) and the Association for Computing Machinery (ACM) (IEEE and ACM, 2004) describe the characteristics of an engineering discipline (see Table 4.2). These characteristics are adopted here as engineering verification criteria. The abbreviations selected to represent each of these criteria are listed in the right-hand column of Table 4.2.

### 4.4 Verification against the two sets of criteria: step 2

The set of the 34 candidate FP is next mapped to the two sets of engineering criteria: each candidate FP is taken as input and analyzed using each of Vincenti's (Vincenti W. G. 1990) seven criteria and, again, each of the seven (IEEE and ACM, 2004) software engineering criteria.

Table 4.2 Identification of IEEE & ACM engineering criteria

ID.	Engineering criteria identified	Abbreviation
1	Engineers proceed by making a series of decisions, carefully evaluating options, and choosing an approach at each decision point that is appropriate for the current task in the current context. Appropriateness can be judged by tradeoff analysis, which balances costs against benefits.	<b>Decision making</b>
2	Engineers measure things, and, when appropriate, work quantitatively. They calibrate and validate their measurements and they use approximations based on experience and empirical data.	<b>Measurements</b>
3	Engineers emphasize the use of a disciplined process when creating a design and can operate effectively as part of a team in doing so.	<b>Disciplined process</b>

Table 4.2 Identification of IEEE &amp; ACM engineering criteria (continued)

ID.	Engineering criteria identified	Abbreviation
4	Engineers can have multiple roles: research, development, design, production, testing, construction, operations, management and others, such as sales, consulting, and teaching.	<b>Engineer's roles</b>
5	Engineers use tools to apply processes systematically. Therefore, the choice and use of appropriate tools is key to engineering.	<b>Use of Tools</b>
6	Engineers, via their professional societies, advance by the development and validation of principles, standards, and best practices.	<b>Development and validation</b>
7	Engineers reuse designs and design artifacts.	<b>Reuse design</b>

The detailed output of the mapping to Vincenti's engineering criteria is presented in Annex III-1, where the letter D represents a direct mapping, and the letter I, an indirect mapping. For instance, in Annex III-1:

- Candidate FP #2, on measurement maps directly to Vincenti criteria #6 (Measurement) and it maps indirectly to Vincenti's criteria #4 (Quality);
- Candidate FP #31 (Know software engineering's techniques before using development tools) has only an indirect mapping to criteria #3 (Techniques) and to #7 (Assessment);
- Finally, there are candidates FP with no mapping to any engineering criteria: for instance, candidate FP #13 (Give product to customers early).

This first verification against the Vincenti criteria leads to (see in Annex III-3 these color-coded groupings):

- 12 candidates FP have at least one direct mapping to a Vincenti engineering criteria.
- 21 candidates FP have only indirect mappings to a Vincenti engineering criteria.
- One candidate FP has no direct or indirect mapping to a Vincenti engineering criteria.

Annex C-2 presents the full set of mappings to the seven IEEE & ACM engineering criteria. For instance, in Annex III-2:

- Candidate FP #2 (Apply and use quantitative measurements...) has only a direct mapping to criteria #1 (Decision making) and to #2 (Measurements).
- Candidate FP #16 (Invest in the understanding of the problem) is mapped indirectly to criteria #1 (Decision making) and to #3 (Disciplined process).
- Candidate FP #4 (Build with and for reuse) is mapped directly and indirectly to criteria #7 (Reuse) and to #3 (Disciplined process).
- Finally, candidate FP #13 (Give products to customers early) is not related to any engineering criteria.

This second verification against the IEEE and ACM criteria leads to (see Annex III-4 for these color-coded groupings):

- 15 candidates FP with at least one direct mapping to an IEEE-ACM engineering criteria;
- 16 candidates FP have only indirect mappings to an IEEE-ACM engineering criteria.
- 3 candidates FP have neither direct nor indirect mapping to an IEEE-ACM engineering criteria.

#### **4.5 Analysis and consolidation using both sets of criteria: step 3**

##### **4.5.1 Analysis across each set of engineering criteria**

The candidate FP with a direct mapping to either Vincenti or IEEE-ACM criteria are listed in Table 4.3. From a comparison of both columns in Table 4.3, the candidate FP with direct mappings can then be grouped into three sets:

- Candidate FP with Vincenti mapping similar to the IEEE-ACM mapping (gray shading Table 4.3);
- Candidate FP with Vincenti mapping with no equivalent IEEE-ACM mapping;
- Candidate FP with IEEE-ACM mapping with no equivalent Vincenti mapping.

Table 4.3 Candidate FP that directly meets criteria from either set of criteria

#	Vincenti mapping	#	IEEE-ACM mapping
2	Apply and use quantitative measurements in decision making	2	Apply and use quantitative measurements in decision making
4	Build with and for reuse	4	Build with and for reuse
		5	Define software artifact rigorously
		6	Design for maintenance
7	Determine requirements now		
9	Don't try to retrofit quality		
		10	Don't write your own test plans
11	Establish a software process that provides flexibility		
		12	Fix requirements specification error now
14	Grow systems incrementally		
15	Implement a disciplined approach and improve it continuously	15	Implement a disciplined approach and improve it continuously
16	Invest in the understanding of the problem		
		18	Keep design under intellectual control
21	Quality is the top priority; long term productivity is a natural consequence of high quality	21	Quality is the top priority; long term productivity is a natural consequence of high quality
		22	Rotate (high performer) people through product assurance
23	Since change is inherent to software, plan for it and manage it		
24	Since tradeoffs are inherent to software engineering, make them explicit and document it	24	Since tradeoffs are inherent to software engineering, make them explicit and document it
		25	Strive to have a peer, rather than a customer, find a defect
		26	Tailor cost estimation methods
27	To improve design, study previous solutions to similar problems	27	To improve design, study previous solutions to similar problems
		31	Know software engineering's techniques before using development tools

**Set A:**

From Table 4.3 one observes that six candidates FP (#2, #4, #15, #21, #24, #27) are present in both columns (the highlighted ones) and therefore satisfy at least one engineering criteria in each set of criteria (Vincenti and IEEE-ACM): these six could reasonably be considered as FP that conform to engineering.

**Set B:**

From Table 4.3 there are 6 candidate FP (#7, #9, #11, #14, #16, #23) that meet the Vincenti criteria, but no IEEE-ACM criteria. Can these still be considered as FP, or are they mere instances of more fundamental principles?

To answer the above question, one could reasonably argue from the Vincenti subset that:

- Candidate FP #7 (Determine requirements now) can be deduced from candidate FP #16 (Invest in the understanding of the problem);
- Candidate FP #9 (Don't try to retrofit quality) can be deduced from candidate FP #21 (Quality is the top priority; long-term productivity is a natural consequence of high quality);
- Candidate FP #11 (Establish a software process that provides flexibility) can be deduced from FP #9 (Don't try to retrofit quality).

This would then eliminate candidate FP #7, #9 and #11 from the list of FP, since they represent specific instantiations of more general FP, while principles #16 #14 and #23 would be retained on the FP list.

**Set C:**

From Table 4.3 there remain 9 candidate FP #5, #6, #10, #12, #18, #22, #25, #26 and #31 that meet IEEE-ACM criteria, but no Vincenti criteria without a corresponding direct mapping to the Vincenti criteria: it could be reasonably argued that these 9 can be deduced from those with direct Vincenti mappings: for instance, FP #18 (Keep design under intellectual control), and FP #31 (Know software engineering techniques before using

development tools) can be deduced from FP #15 (Implement a disciplined approach and improve it continuously).

This would then eliminate candidate FP #5, #6, #10, #12, #18, #22, #25, #26 and #31 from the list of FP (see Table 4.5) since they represent specific instantiations of more general FP.

Finally, a subset of only 9 (see Table 4.4) from the list of 34 candidates identified in Seguin 2006 are recognized as software engineering FP, the remaining 25 being specific instantiations of those 9. In Table 4.4, these FP are sequenced from 1 to 9, together with their original sequence number (right-hand column) assigned when the initial list of 34 candidates was compiled.

Table 4.4 List of software engineering FP

Numbering	List of software engineering FP	Initial numbering (Séguin N. 2006)
1	Apply and use quantitative measurements in decision making	2
2	Build with and for reuse	4
3	Grow systems incrementally	14
4	Implement a disciplined approach and improve it continuously	15
5	Invest in the understanding of the problem	16
6	Quality is the top priority; long term productivity is a natural consequence of high quality	21
7	Since change is inherent to software, plan for it and manage it	23
8	Since tradeoffs are inherent to software engineering, make them explicit and document it	24
9	To improve design, study previous solutions to similar problems	27

#### 4.5.2 Identification of a hierarchy

Table 4. presents next the outcome of the analysis of the 25 remaining candidate FP as instantiations of the 9 FP identified in Table 4.5

Table 4.5 Hierarchy of candidate FP

#	Direct mapping to Vincenti criteria	Derived instantiation (= Indirect mapping) With the numbering in (Séguin N., 2006)
1	Apply and use quantitative measurements in decision making	26 Tailor cost estimation methods 8 Don't overstrain your hardware
2	Build with and for reuse	
3	Grow systems incrementally	5 Define software artefacts rigorously 20 Produce software in a stepwise fashion
4	Implement a disciplined approach and improve it continuously	1 Align incentives for developer and customer 10 Don't write your own test plans 17 Involve the customer 18 Keep design under intellectual control 20 Produce software in a stepwise fashion 31 Know software engineering's techniques before using development tools 19 Maintain clear accountability for results 29 Use documentation standards 10 Don't write your own test plans
5	Invest in the understanding of the problem	7 Determine requirements now 12 Fix requirements specification error now 17 Involve the customer
6	Quality is the top priority; long term productivity is a natural consequence of high quality	9 Don't try to retrofit quality 22 Rotate (high performer) people through product assurance 25 Strive to have a peer, rather than a customer, find a defect 30 Write programs for people first 3 Build software so that it needs a short user manual 11 Establish a software process that provides flexibility 28 Use better and fewer people

Table 4.5 Hierarchy of candidate FP (continued)

#	Direct mapping to Vincenti criteria	Derived instantiation (= Indirect mapping) With the numbering in (Séguin N., 2006)
7	Since change is inherent to software, plan for it and manage it	6 Design for maintenance 33 Choose a programming language to assure maintainability 32 Select tests based on the likelihood that they will find faults 34 In face of unstructured code, rethink the module and redesign it from scratch.
8	Since tradeoffs are inherent to software engineering, make them explicit and document it	
9	To improve design, study previous solutions to similar problems	

#### 4.6 External verification: step 4

##### 4.6.1 Design

A proposal for an external verification was prepared and submitted to the International Conference on Engineering Education – ICEE 2007 Coimbra (Portugal) 2007. This proposal was accepted and included in the ICEE conference program (see Annex VII). As a consequence, of the acceptance of the proposal, a half day session workshop on “The Engineering Foundations of Software Engineering” was planned. The plan included three parts:



### **A- Familiarization**

To familiarize researchers with the topic of “The Engineering Foundations of Software Engineering” a number of presentations were planned, to be followed by a discussion session:

- “Delphi Studies on Fundamental Principles of Software Engineering”;
- “The literature on software engineering principles + identification of criteria for selecting candidate fundamental principles”;
- “Vincenti engineering knowledge types and their mapping to software engineering concepts”;
- “The core set of fundamental principles selected using Vincenti and ACM-IEEE 2000 curriculum criteria”.

### **B- Participation on site at the workshop**

The discussion session was to consist in asking participants:

- To verify the output related to the results of the mappings in Annex III C-1 and Annex III C-2 between the mapping of the 34 CFP and both the Vincenti and IEEE&ACM engineering criteria;
- To verify the list of nine software engineering FP see Table 4.4;
- To verify the hierarchy of candidate FP see Table 4.5.

### **C- Post-workshop follow-up**

If not all topics in B- could be addressed by the participants within the workshop timeframe, they were to be asked for a further post-workshop follow-up through email.

## **4.6.2 Execution**

### **A- Familiarization**

Familiarization was conducted for the participants who attended the workshop.

### **B- Participation on site**

Because of the limited time allocated to the discussion session and to the new approach not familiar to the participants in the workshop on the fundamental principles of software engineering; the discussion session took a long time. Therefore, participants were just asked, to verify the list of nine software engineering FP (see Table 4.4).

As a result of this verification, feedback regarding the nine engineering FP were collected. The participants agreed on the nine engineering FP presented to them.

### **C- Follow-up**

Additional feedbacks from two participants about the verification of the following output were received subsequently by email.

- Result of mapping of the candidate FP to Vincenti engineering criteria Annex III C-1;
- Result of mapping of candidate FP to IEEE & ACM engineering criteria Annex III C-2;
- Result of nine software engineering FP see- Table 4.4;
- Result of the hierarchy of candidate FP see- Table 4.5;

For more detail (see Annex-VII).

## **4.7 Summary**

This chapter has taken as input, or as its object of study, the set of 34 statements identified in (Séguin N. 2006) as being candidate FP of software engineering. This set has been analyzed from an engineering perspective using the engineering criteria identified by either Vincenti or the IEEE-ACM joint effort on developing a software engineering curriculum.

The 34 candidate FP were divided into three categories: A) candidate FP that are directly linked to engineering, B) candidate FP that are indirectly linked to engineering, and C) candidate FP that have no specific link to engineering.

In the next step, candidate FP from both lists were analyzed and compared. In the final step, the set of the proposed nine engineering FP was verified at a workshop organized during the International Conference on Engineering Education – ICEE 2007 Coimbra (Portugal) 2007.

Software engineering, as a discipline, is certainly not yet as mature as other engineering disciplines and, while a number of authors have proposed over 308 distinct FP, a consensus on a set of well-recognized FP has been lacking. The proposed reduced list of 9 FP, from an engineering perspective, now needs to be further discussed by the software engineering community.

Of course, this list depends on the methodology used, and is being proposed to the engineering community for discussion and scrutiny with the aim of improvement and development of a consensus over time.

There is no claim that this list is exhaustive or that it covers the whole software engineering discipline. Even though the inputs to this analysis were derived from an extensive literature review, no guarantee is given that those authors have indeed provided full coverage of the software engineering discipline.

Similarly, the hierarchy proposed in Table 4. is derived from the engineering criteria used in this analytical approach. Further research should be carried out to verify the completeness of the criteria used

## CHAPTER 5

### IDENTIFICATION OF THE SOFTWARE ENGINEERING PRINCIPLES WITHIN THE CONTENT OF THE SWEBOK GUIDE – ISO TR 19759

#### 5.1 Introduction

This chapter takes as input the result achieved from the previous chapter that is, the list of the nine FP that were verified from an engineering perspective.

This chapter presents the analysis of the content of the SWEBOK Guide knowledge areas (ISO-TR-19759 2004) with respect to its coverage for these nine FP. These KAs were analyzed to identify whether the nine FP were present in the SWEBOK Guide.

This chapter also presents phase 4 and is organized as follows: Section 2 describes the mapping of the FP to the content of the SWEBOK Guide. Section 3 presents the FP in “Software requirements” knowledge area. Section 4 presents the FP in “Software quality” knowledge area. Section 5 presents the results in the other KAs. Section 6 presents the analysis of the mapping results and finally a summary is presented in section 7. Annex IV presents the detailed results related to the mapping of the set of the nine engineering principles to the related knowledge areas of the SWEBOK Guide.

#### 5.2 Mapping the FP to the SWEBOK KAs

In the SWEBOK Guide (ISO-TR-19759 2004) each of the ten knowledge areas is composed of subareas and topics. To identify the coverage of the nine FP in the SWEBOK Guide, these ten knowledge areas were analyzed to check if there is any mapping between the content of the SWEBOK Guide and the nine FP.

This analysis consisted of verifying if each of the nine FP were present first at the subarea level and, second, at the topics level.

This section presents the mapping results between the different SWEBOK Guide knowledge areas and the nine FP.

Also, each of these mapping results is described in a different table. These tables describe the presence of the 9 engineering fundamental principles within the knowledge areas of the SWEBOK Guide. The tables in this chapter are organized as follows: for each of the knowledge areas of the SWEBOK Guide, the subsequent columns for subareas, topics and finally the engineering fundamental principles are described. These engineering fundamental principles are numbered from #1 to #9. As one can see in these tables, the presence is mentioned for each topic where one can find the identification number related to each FP.

The next sections present two examples for “Software requirements” and “Software quality” knowledge areas with their detailed results. The “Software requirements” in section 5.3 corresponds to the knowledge area the most covered by the FP and the “Software quality” in section 5.4 corresponds to the knowledge area the least covered by the FP.

### **5.3 The FP in software requirements knowledge area**

Table 5.1 and Table 5.2 describe the engineering fundamental principle coverage for the “Software requirement” knowledge area. For instance, the topic “Elicitations techniques” under subarea “Requirements elicitation” covers the following engineering principles:

- (3) Grow systems incrementally;
- (4) Implement a disciplined approach and improve it continuously;
- (5) Invest in the understanding of the problem.

Table 5.1 describes the summary of the mapping between the FP and the “Software requirements” subareas. Table 5.2 describes the detailed mapping of the overall topics between the FP and the “Software requirements”.

Table 5.1 Summary mapping of the engineering FP in the “Software requirements” KA

Software requirements subareas	Engineering fundamental principles
Software requirements fundamentals	#1
Requirements process	#6
Requirements elicitation	#3, #4,#5
Requirements analysis	#2, #3, #4,#5,#8
Requirements specification	#3, #4
Requirements validation	#2,#3,#4, #6
Practical consideration	#1, #7

Table 5.2 Detailed mapping of the engineering FP in the “Software requirements” KA

Software requirements subareas	Software requirements topics	Engineering fundamental principles
Software requirements fundamentals	Definition of a software requirement	
	Product and process requirements	
	Functional & nonfunctional requirement	
	Emergent properties	
	Quantifiable requirements	#1
	System requirements and software requirements	

Table 5.2 Detailed mapping of the engineering FP in the “Software requirements” KA  
(continued)

Software requirements subareas	Software requirements topics	Engineering FP
<b>Requirements process</b>	Process models	
	Process actors	
	Process support and management	
	Process quality and improvement	#6
<b>Requirements elicitation</b>	Requirements sources	#3, #4,#5
	Elicitations techniques	#3, #4,#5
<b>Requirements analysis</b>	Requirements classification	#3, #4,#5
	Conceptual modeling	#2,#3, #4,#5
	Architectural design requirements allocation	#2,#3, #4
	Requirement negotiation	#3, #4,#5,#8
<b>Requirements specification</b>	System definition document	#3, #4
	Systems requirement specification	#3, #4
	Software requirement specification	#3, #4
<b>Requirements validation</b>	Requirement reviews	#3, #4, #6
	Prototyping	#3, #4, #6
	Model validation	#3, #4, #6
	Acceptance test	#2,#3,#4, #6
<b>Practical consideration</b>	Iterative nature of the requirement process	#7,
	Change management	#7,
	Requirement attributes	#7,
	Requirements tracing	#7,
	Measuring requirement	#1,

The following eight FP were identified as being present in the “Software requirements” KA:

- (1) Apply and use quantitative measurements in decision making;
- (2) Build with and for reuse;
- (3) Grow systems incrementally;
- (4) Implement a disciplined approach and improve it continuously;
- (5) Invest in the understanding of the problem;
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality;
- (7) Since change is inherent to software, plan for it and manage it;
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it;

The “Software requirement” knowledge area did not cover the following FP:

- (9) To improve design, study previous solutions to similar problems.

#### **5.4 The FP in software quality knowledge area**

Table 5.3 and Table 5.4 describe the engineering fundamental principles coverage for the “Software quality” knowledge area. For instance, the topic “Software quality measurement” under the subarea “Practical considerations” covers the following engineering principle:

- (1) Apply and use quantitative measurements in decision making.

Table 5.3 describes the summary mapping between the FP and the “Software quality” subareas and Table 5.4 describes the detailed mapping of the overall topics between the FP and the “Software Quality”.



Table 5.3 Summary mapping of the engineering FP in the “Software quality” KA

Software quality subareas	Engineering fundamental principles
Software quality fundamentals	# 4
Software quality management processes	# 4
Practical considerations	# 1

Table 5.4 Detailed mapping of the engineering FP in the “Software quality” KA

Software quality subareas	Software quality topics	Engineering FP
Software quality fundamentals	Software engineering culture and ethics	
	Value and costs of quality	
	Quality models and characteristics	
	Quality improvement	# 4
Software quality management processes	Software quality assurance	# 4
	Verification and validation	# 4
	Reviews and audits	# 4
Practical considerations	Application quality requirements	
	Defect characterization	
	Software quality management techniques	
	Software quality measurement	# 1

The following two FP were identified as being present in the “Software quality” KA:

- (1) Apply and use quantitative measurements in decision making;
- (4) Implement a disciplined approach and improve it continuously.

“Software quality” KA didn’t cover the following FP.

- (2) Build with and for reuse;
- (3) Grow systems incrementally;
- (5) Invest in the understanding of the problem;
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality;
- (7) Since change is inherent to software, plan for it and manage it;
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it;
- (9) To improve design, study previous solutions to similar problems.

## 5.5 Results in other KAs

This section presents the results of the mapping for the other knowledge areas such as “Software design”, “Software construction”, “Software testing”, with a description and the related results in Tables 5.3 to 5.9. The detailed results are presented in the Annex IV.

### 5.5.1 “Software design” knowledge area

Table 5.5 and Table 5.6 describe the engineering fundamental principles coverage for the “Software design” knowledge area.

Table 5.5 Summary mapping of engineering FP in the “Software design” KA

Software design subareas	Engineering fundamental principles
Software design fundamentals	#4
Keys issues in software design	#2, #4
Software structure and architecture	#2, #4, #9
Software design quality analysis and evaluation	#1, #6
Software design notations	#4
Software design strategies and methods	#2, #4

Table 5.6 Detailed mapping of engineering FP in the “Software design” KA

Software design subareas	Software design topics	Engineering FP
<b>Software design fundamentals</b>	General design concepts	
	The context of software design	
	The software design process	#4
	Enabling techniques	#4
<b>Keys issues in software design</b>	Concurrency	#4
	Control and handling of events	#4
	Distribution of components	#2, #4
	Error and exception handling and fault tolerance	#4
	Interaction and presentation	#4
	Data persistence	#4
<b>Software structure and architecture</b>	Architectural structures and viewpoints	#4
	Architectural styles (macro architectural patterns)	#2, #4, #9
	Design patterns (macro architectural patterns)	#2, #4, #9
	Families of programs and frameworks	#4
<b>Software design quality analysis and evaluation</b>	Quality attributes	#6
	Quality analysis and evaluation techniques	#6
	Measures	#1
<b>Software design notations</b>	Structural descriptions (static view)	#4
	Behavior descriptions (dynamic view)	#4
<b>Software design strategies and methods</b>	General strategies	#4
	Function-oriented (structured design)	#4
	Object-oriented design	#4
	Data-structured centered design	#4
	Component-based design	#2, #4
	Other methods	#4

### 5.5.2 Software construction knowledge area

Table 5.7 and Table 5.8 describe the engineering fundamental principle coverage for the “Software construction” knowledge area.

Table 5.7 Summary mapping of engineering FP in the “Software construction” KA

Software construction subareas	Engineering fundamental principles
Software construction fundamentals	#6, #7
Managing construction	#1
Practical considerations	#2, #4, #6

Table 5.8 Detailed mapping of engineering FP in the “Software construction” KA

Software construction subareas	Software construction topics	Engineering fundamental principles
Software construction fundamentals	Minimizing complexity	#6
	Anticipating change	#7
	Constructing for verification	#6
	Standards in construction	#6
Managing construction	Construction models	
	Construction planning	
	Construction measurement	#1
Practical considerations	Construction design	#4
	Construction languages	#4
	Coding	#4
	Construction testing	#4, #6
	Reuse	#2
	Construction quality	#6
	Integration	#4

### 5.5.3 Software testing knowledge area

Table 5.9 and Table 5.10 describe the engineering fundamental principle coverage for “Software testing” knowledge area.

Table 5.9 Summary mapping of engineering FP in the “Software testing” KA

Software testing subareas	Engineering fundamental principles
<b>Software testing fundamentals</b>	
<b>Test levels</b>	# 4,# 8
<b>Test techniques</b>	# 4
<b>Test related measures</b>	#4
<b>Test process</b>	# 1, # 2, # 4, # 7

Table 5.10 Detailed mapping of engineering FP in the “Software testing” KA

Software testing subareas	Software testing topics	Engineering fundamental principles
<b>Software testing Fundamentals</b>	Testing-related terminology	
	Keys issues	
	Relationships of testing to other activities	
<b>Test levels</b>	The target of the test	# 4
	Objectives of testing	# 4,# 8

Table 5.10 Detailed mapping of engineering FP in the “Software testing” KA (continued)

Software testing subareas	Software testing topics	Engineering fundamental principles
<b>Test techniques</b>	Based on tester's intuition and experience	# 4
	Specification-based	# 4
	Code-based	# 4
	Fault-based	# 4
	Usage-based	# 4
	Based on nature of application	# 4
	Selecting and combining techniques	# 4
<b>Test related Measures</b>	Evaluation of the program under test	# 1
	Evaluation of the tests performed	# 1
<b>Test process</b>	Practical considerations	# 1, # 2, # 4, # 7
	Test activities	# 4

#### 5.5.4 Software maintenance knowledge area

Table 5.11 and Table 5.12 describe the engineering fundamental principles coverage for “Software maintenance” knowledge area.

Table 5.11 Summary mapping of engineering FP in the “Software maintenance” KA

Software maintenance subareas	Engineering FP
<b>Software maintenance fundamentals</b>	
<b>Key issues in software maintenance</b>	# 1
<b>Maintenance process</b>	# 4, # 6, # 7
<b>Techniques for maintenance</b>	

Table 5.12 Detailed mapping of engineering FP in the “Software maintenance” KA

Software maintenance subareas	Software maintenance topic	Engineering fundamental principles
<b>Software maintenance fundamentals</b>	Definitions and terminology	
	Nature of maintenance	
	Need for maintenance	
	Majority of maintenance costs	
	Evolution of software	
	Categories of maintenance	
<b>Key issues in software maintenance</b>	Technical issues	
	Management issues	
	Maintenance cost estimation	# 1
	Software maintenance measurement	# 1
<b>Maintenance process</b>	Maintenance processes	# 4, # 6
	Maintenance activities	# 4,# 7
<b>Techniques for maintenance</b>	Program comprehension	
	Re-engineering	
	Reverse engineering	

### 5.5.5 Software configuration management knowledge area

Table 5.13 and Table 5.14 describe the engineering fundamental principles coverage for “Software Configuration Management” knowledge area.

Table 5.13 Summary mapping of engineering FP in the “Software configuration management” KA

Software configuration management subareas	Engineering fundamental principles
Management of the SCM process	#1

Table 5.13 Summary mapping of engineering FP in the “Software configuration management” KA (continued)

Software configuration management subareas	Engineering fundamental principles
Software configuration identification	# 4
Software configuration control	#7
Software configuration status accounting	# 4
Software configuration auditing	# 4
Software release management and delivery	# 4

Table 5.14 Detailed mapping of engineering FP in “Software configuration management”

Software configuration management subareas	Software configuration management topics	Engineering fundamental principles
<b>Management of the SCM process</b>	Organizational context for scm	
	Constraints and guidance for the scm process	
	Planning for scm	
	Scm plan	
	Surveillance of software configuration management	#1
<b>Software configuration identification</b>	Identifying items to be controlled	# 4
	Software library	# 4
<b>Software configuration control</b>	Requesting, evaluating, and approving software changes	#7
	Implementing software changes	#7
	Deviations and waivers	#7
<b>Software configuration status accounting</b>	Software configuration status information	# 4
	Software configuration status reporting	# 4



Table 5.14 Detailed mapping of engineering FP in “Software Configuration Management”  
(continued)

<b>Software configuration management subareas</b>	<b>Software configuration management topics</b>	<b>Engineering fundamental principles</b>
<b>Software configuration auditing</b>	Software functional configuration audit	# 4
	Software physical configuration audit	# 4
	In-process audits of a software baseline	# 4
<b>Software release management and delivery</b>	Software building	# 4
	Software release management	# 4

### 5.5.6 Software engineering management knowledge area

Table 5.15 and Table 5.16 describe the engineering fundamental principles coverage for “Software engineering management” knowledge area.

Table 5.15 Summary mapping of engineering FP in the “Software engineering management”  
KA

<b>Software engineering management subareas</b>	<b>Engineering fundamental principles</b>
<b>Initiation and scope definition</b>	# 4
<b>Software project planning</b>	# 4, # 6
<b>Software project enactment</b>	# 4
<b>Closure</b>	#4
<b>Software engineering measurement</b>	#1

Table 5.16 Detailed mapping of engineering FP in: “Software engineering management”

Software engineering management	Software engineering management topics	Engineering fundamental principles
<b>Initiation and scope definition</b>	Feasibility analysis	# 4
	Process for the review and revision of requirements	# 4
<b>Software project planning</b>	Process planning	# 4
	Determine deliverables	# 4
	Effort, schedule, and cost estimation	# 4
	Resource allocation	# 4
	Risk management	# 4
	Quality management	# 4, # 6
	Plan management	# 4
<b>Software project enactment</b>	Implementation of plans	# 4
	Supplier contract management	# 4
	Implementation of measurement process	# 4
	Monitor process	# 4
	Control process	# 4
	Reporting	# 4
<b>Closure</b>	Determining closure	# 4
	Closure activities	# 4
<b>Software engineering measurement</b>	Establish and sustain measurement commitment	# 1
	Plan the measurement process	# 1
	Perform the measurement process	# 1
	Evaluate measurement	# 1

### 5.5.7 Software engineering process knowledge area

Table 5.17 and Table 5.18 describe the engineering fundamental principles coverage for the ‘software engineering process’ knowledge area.

Table 5.17 Summary mapping of engineering FP in the “Software Engineering Process” KA

Software process subareas	Engineering fundamental principles
Process implementation and change	#4
Process definition	#4
Process assessment	#6
Product and process measurement	# 1

Table 5.18 Detailed mapping of engineering FP in: “Software engineering process”

Software process Subareas	Software process topics	Engineering fundamental principles
<b>Process implementation and change</b>	Process infrastructure	# 4
	Software process management cycle	# 4
	Models for process implementation and change	# 4
	Practical considerations	# 4
<b>Process definition</b>	Life cycle models	# 4
	Software life cycle processes	# 4
	Notations for process definitions	# 4
	Process adaptation	# 4
	Automation	# 4

Table 5.18 Detailed mapping of engineering FP in: “Software engineering process”  
(continued)

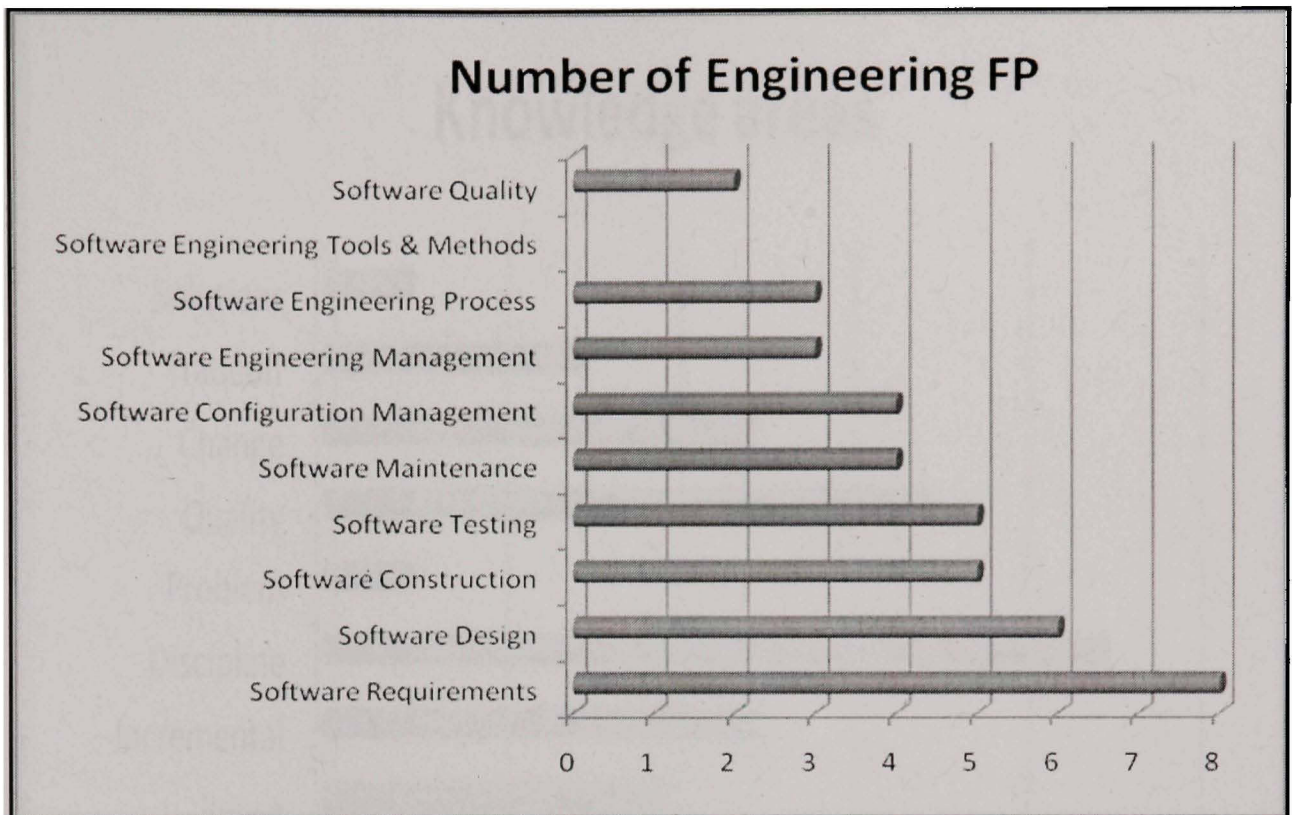
Software process Subareas	Software process topics	Engineering fundamental principles
<b>Process assessment</b>	Process assessment models	#6
	Process assessment methods	#6
<b>Product and process measurement</b>	Software process measurement	# 1
	Software product measurement	# 1
	Quality of measurement results	# 1
	Software information models	# 1
	Measurement techniques	# 1

## 5.6 Analysis of the mapping results

This section introduces the results of the analysis related to the mapping of the engineering fundamental principles and the SWEBOK Guide KAs. First, figure 5.1 presents the frequency coverage of the FP by knowledge area: That is, for each knowledge area there are different occurrences of the FP. One can see that from Figure 5.1:

- “Software requirements” covers eight FP;
- “Software configuration management”, “Software engineering management”, “Software engineering process” covers three FP;
- “Software quality” covers two FP.

These results demonstrate that the “Software requirements” KA addresses most of the engineering fundamental principles, while “Software quality” addresses the smallest number of the engineering fundamental principles.

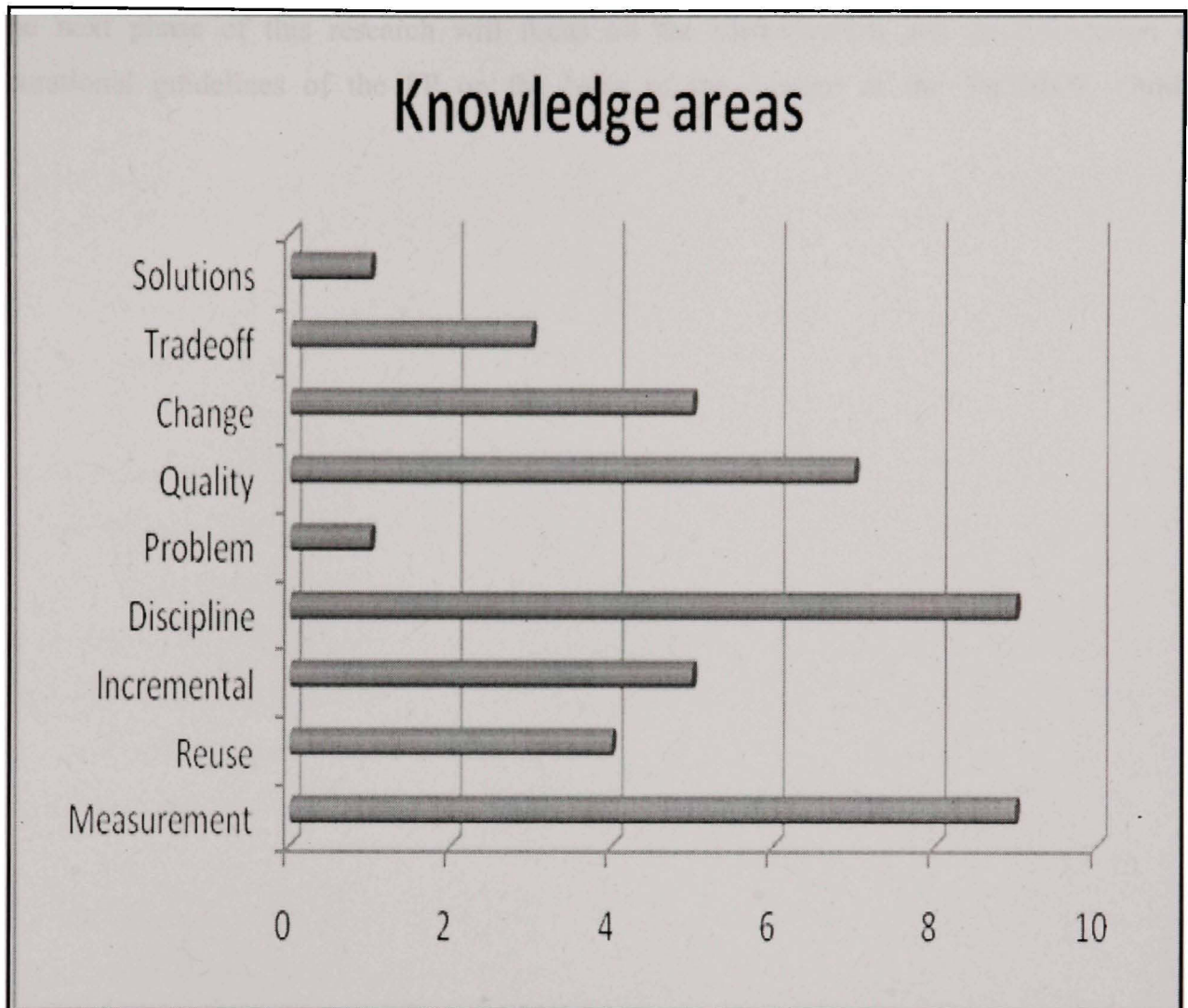


**Figure 5.1: Frequency of engineering fundamental principles by knowledge area**

Second,

Figure 5.2 shows the frequency coverage of the engineering fundamental principles for SWEBOK KAs. One can see that out of the nine engineering principles:

- “FP no. 4 - Implement a disciplined approach and improve it continuously” is covered by nine KA’s out of ten, making it the most covered FP;
- “FP no. 1 - Apply and use quantitative measurements in decision making” is covered by nine knowledge areas;
- “FP no. 8 - Since tradeoffs are inherent to software engineering, make them explicit and document it” is covered by three knowledge areas.
- These results demonstrate that:
- “FP no. 4 - Implement a disciplined approach and improve it continuously” and “FP no. 1 - Apply and use quantitative measurements in decision making” are the most covered in the SWEBOK Guide.



**Figure 5.2: Frequency of engineering fundamental principles for SWEBOK KAs**

### 5.7 Summary

In this chapter the focus was to analyze whether or not the nine FP were indeed present in the SWEBOK KAs (ISO-TR-19759 2004).

This analysis was based on mapping between the nine FP and the SWEBOK Guide knowledge areas. As a result of this mapping, out of the nine FP, eight (8) were found to be present in the “Software requirements” KA. Two (2) FP were found to be present in the “Software quality”. “Software requirements” is the KA which addresses the largest number of FP and “Software quality”, the least number of FP.

The next phase of this research will focus on the identification and documentation of operational guidelines of the FP on the basis of the content of the SWEBOK Guide.

## CHAPTER 6

### DESCRIPTION OF AN OPERATIONAL PERSPECTIVE OF THE SOFTWARE ENGINEERING PRINCIPLES ON THE BASIS OF THE CONTENT OF THE SWEBOK GUIDE

#### 6.1 Introduction

The nine engineering FPs were derived from an analysis of the literature. However, there was no description on how to operate them (Séguin N. 2006).

The SWEBOK Guide (ISO-TR-19759 2004) is composed of ten knowledge areas (KAs). The knowledge areas which are described in the SWEBOK Guide are necessary but not sufficient for a software engineer since the practitioners will have to be familiar with a number of additional domains of knowledge, such as computer science and systems engineering.

As mentioned earlier in chapters 4 and 5 of this thesis, the nine engineering FP have been identified for software engineering using (Vincenti W. G. 1990) and the (IEEE and ACM, 2004) software engineering curriculum to assess their coverage in the SWEBOK Guide knowledge areas.

This chapter presents phase 5: how each of the engineering FP that are present in each knowledge areas are described (i.e. operationalized) on the basis of the content of the SWEBOK KAs.

This chapter is organized as follows: Section 2 presents the proposed operational guidelines for the SWEBOK Guide. Section 3 presents “Software requirements” description. Section 4 presents “Software design”. Section 5 presents “Software construction”. Section 6 presents “Software testing”. Section 7 presents “Software maintenance”. Section 8 presents “Software



configuration management”. Section 9 presents “Software engineering process”. Section 10 presents “Software quality” and finally a summary is presented in section 11.

In addition, Annex V presents the proposed operational guidelines aligned with international standards such as IEEE 1362 1998 Concepts of Operations document (ConOps) (IEEE STD 1362-1998) for the main knowledge areas of the SWEBOK Guide.

## **6.2 Proposed operational guidelines for the SWEBOK (Annex V)**

The proposed operational guidelines of software engineering principles on the basis of the content of the SWEBOK Guide to provide a set of guidelines that may be used by practitioners during the software development.

This proposed operational guidelines are structured based on the (IEEE STD 1362-1998) which illustrates the operational concepts for the information technology domain: for example, this IEEE standard suggests how to build such operational guidelines by following many steps such as building:

- The set of definition concepts.
- The suggested operational scenario.
- The expected capabilities of the suggested scenario.
- The improved steps to operational scenario and its capabilities.
- The impact of these operational guidelines.

The operational concepts define the important elements on which the scenarios are built. The operational scenarios define how to operate these concepts. The operational capabilities describe the capabilities of the scenarios; operational improvements give a description of how to improve these capabilities and finally the operational impacts on the users and developers.

### 6.3 Software requirements – description of an operational perspective

The “Software requirements” KA documents four phases as follows: elicitation, analysis, specification and validation - see Figure 6.1.

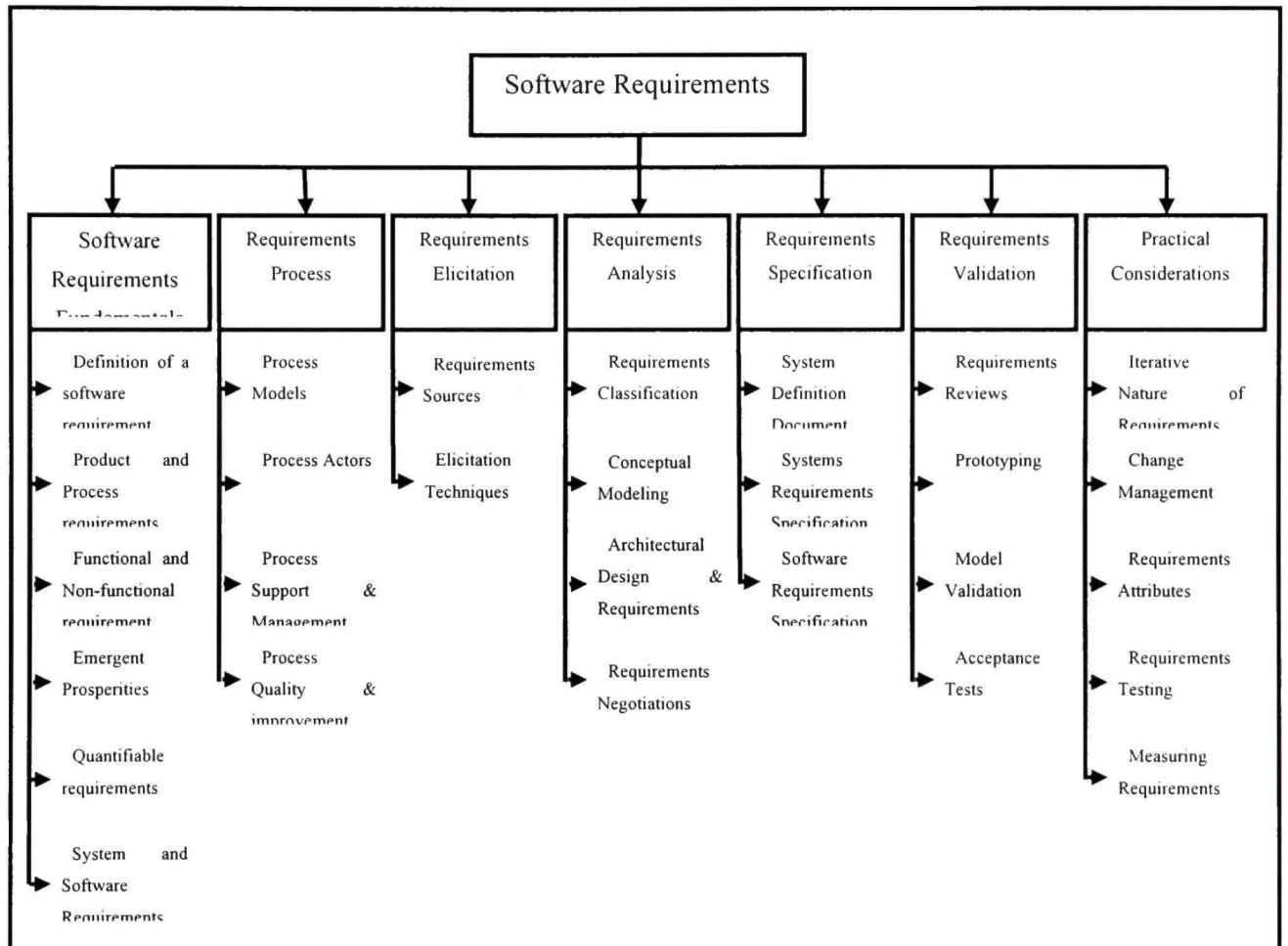


Figure 6.1 : SWEBOK Guide: “Software requirements” knowledge area (ISO-TR-19759, 2004)

#### 6.3.1 Principle # 1: “Apply and use quantitative measurements in decision making”

##### A. Presence of this FP in the taxonomy of this KA

This FP is applied within the following “Software requirements” topics:

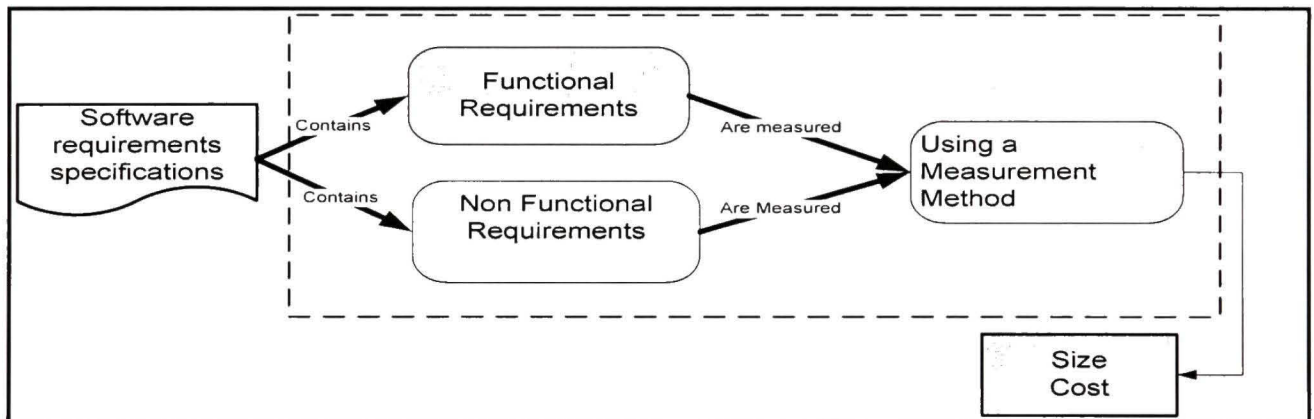
- Quantifiable requirements.
- Measuring requirements.

## B. Operational guidelines documented in this KA for this FP

**Measurement process:** At the end of the software requirements phase, the “software specification document” is produced. This document contains the functional and nonfunctional requirements.

The functional requirements can be measured right from the requirements phase, using a functional size measurements method (eg. COSMIC – ISO 19761). The output of this measurement process is the functional size which can be used as input to estimate the cost of design, development, test or the cost of maintenance tasks.

A model of these operational procedures for this measurement process in the “Software requirements” phase is described in Figure 6.2.



**Figure 6.2: Measurement process: operational view –requirements KA**

### 6.3.2 Principle # 2: “Build with and for reuse”

#### A. Presence of this FP in the taxonomy of this KA

This fundamental principle is applied within the following “Software requirements” topics:

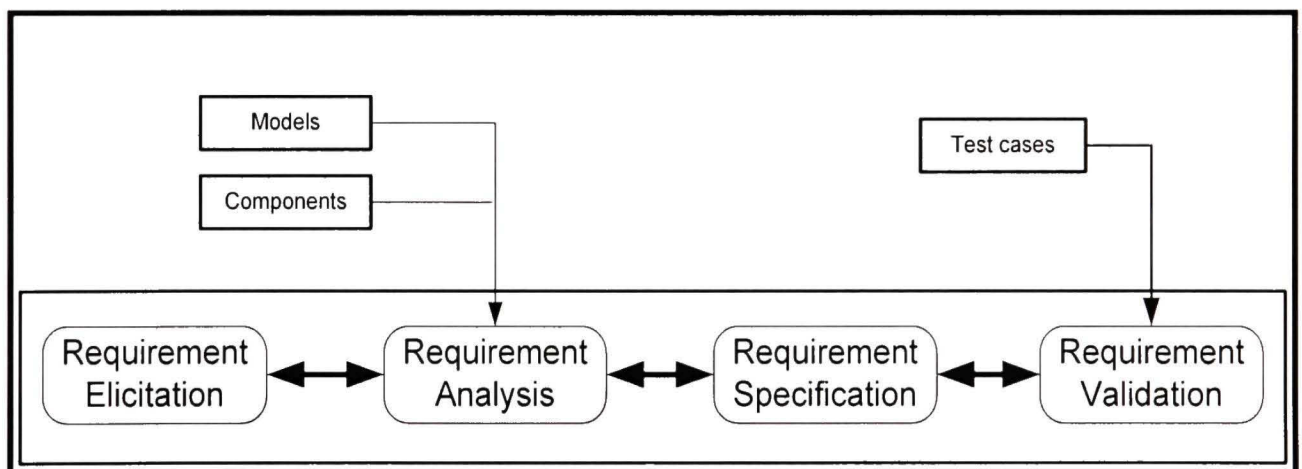
- Conceptual modeling;
- Architectural design and requirements allocation;
- Acceptance test.

## B. Operational guidelines documented in this KA for this FP

**Build with and for reuse:** after finishing the task of eliciting the requirements, the software engineer starts analyzing the requirements by classifying them, and next by modeling them using one of the following models: data and control flows, state models, event traces, user interactions, object models, data models and many others.

In “Conceptual modeling” the software engineer can be interested in developing the system by, for example, reusing the conceptual models for the set of requirements. Components can also be reused in requirements allocation and finally test cases can also be reused to conduct an acceptance test to validate that the software satisfies the requirements.

These models components and test cases can be carefully defined and documented so that they may be reused - see Figure 6.3.



**Figure 6.3: General view for applying “Build with and for reuse” in requirements KA**

### 6.3.3 Principle # 3: “Grow system incrementally”

#### A. Presence of this FP in the taxonomy of this KA

This fundamental principle is applied within the following “Software requirements” subareas:

- Requirement elicitation;
- Requirement analysis;
- Requirement specification;
- Requirement validation.

#### **B. Operational guideline documented in this KA for this FP**

The operational guidelines for the principle # 3: “Grow system incrementally” suggest that software should be build by increment: the implementation of this principle is similar to principle #4: “Implement a disciplined approach and improve it continuously”. The only difference is in the input. To implement “Grow system incrementally” the software engineer starts by focusing on a small set of requirements (that have high priority) then, by slowly increasing the number of requirements in the set.

#### **6.3.4 Principle # 4: “Implement a disciplined approach and improve it continuously”**

##### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software requirements” subareas:

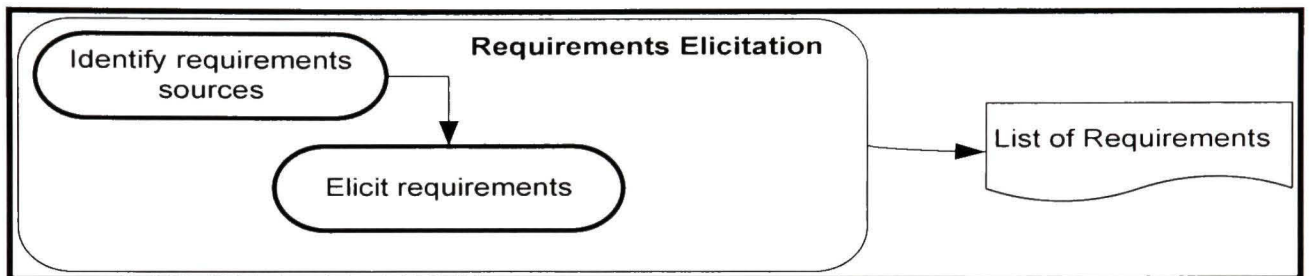
- Requirements elicitation;
- Requirements analysis;
- Requirements specification;
- Requirements validation.

##### **B. Operational guidelines documented in this KA for this FP**

**Requirements activities:** the requirements process is composed of the following four activities:

**Activity 1: requirements elicitation:** in this activity the software engineer starts by defining the sources of requirements and by identifying one of the many sources of each requirement which can be categorized into one of the following: goals, domain knowledge, stakeholders, operational environment and organizational environment. After identifying the source of the requirements software engineers can start collecting requirements by using different

elicitation techniques such as interviews, scenarios, prototypes, facilitated meetings and observations. At the end of this process a list of requirements is produced - see Figure 6.4.



**Figure 6.4: Requirements elicitation- operational view in requirement KA**

**Activity 2: requirements analysis:** in this activity the software engineer starts by classifying the requirements as follows: whether they are functional or non functional, derived requirements, type of requirements product or process, requirements priority (classified on a fixed point scale: mandatory, highly desirable, desirable and optional), the scope of requirements and finally the estimation of volatility and stability requirements.

Next, the software engineer develops a conceptual model of the real world problem using one of the following types of models: data and control flows, state models, event traces, user interactions, object models, data models and many others. Next, the software engineer allocates requirements to components, and finally he negotiates requirements.

**Activity 3: requirements specification:** in the software specification activity the software engineer produces a document. For complex systems three kinds of documents are produced: system definition, system requirements specification and software requirements specification. For simple systems only the “software requirements specification” document is produced.

**Activity 4: requirements validation:** In the “Software requirements” KA the following artefacts are subject to validation and verification: the system definition document, the system specification document and the software requirements specification document.

This activity can be done through: inspections and reviews, prototyping, validating the quality of models, identifying and designing an acceptance test to validate that the finished product satisfies the requirements.

### **6.3.5 Principle # 5: “Invest in the understanding of the problem”**

#### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software requirements” subareas:

- Requirements elicitation;
- Requirements analysis.

#### **B. Operational guidelines documented in this KA for this FP**

The operational guidelines for the principle # 5: “Invest in the understanding of the problem” are concerned with the two following activities:

**Activity 1:** requirements elicitation;

**Activity 2:** requirements analysis.

The practical details for these activities are similar to those mentioned for principle # 4: “Implement a disciplined approach and improve it continuously”.

### **6.3.6 Principle # 6: “Quality is the top priority; long term productivity is a natural consequence of high quality”**

#### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software requirements” topic and subarea:

- Process quality and improvement (SWEBOK topic);
- Requirements validation (SWEBOK subarea).

#### **B. Operational guidelines documented in this KA for this FP**

To ensure the success of the quality of the final product there is a need to perform quality checks right from the beginning in the “Software requirements” process.

### **Process quality and improvement** (SWEBOK topic)

In this topic the software engineer evaluates the quality of the requirements process with the help of quality standards. Process improvement models are used to orient the improvements of the requirements process activities.

### **Requirements validation** (SWEBOK subarea)

The practical details for this activity is similar to the one mentioned for principle # 4: “Implement a disciplined approach and improve it continuously”.

### **6.3.7 Principle # 7:** “Since change is inherent to software, plan for it and manage it”

#### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software requirements” topics:

- Iterative nature of the requirements process;
- Change management;
- Requirements attributes;
- Requirements tracing.

#### **B. Operational guidelines documented in this KA for this FP**

Change management necessitates the following tasks in the software requirements process: Identifying the requirements that possibly change, define the review, approve the process, perform the change, apply requirements tracing, apply impact analysis, apply software configuration management and report change history.

### **6.3.8 Principle # 8:** “Since tradeoffs are inherent to software engineering, make them explicit and document them”

#### **A. Presence of this FP in the taxonomy of this KA**

This fundamental principle is applied into the following topic:

- Requirements negotiation.



## B. Operational guidelines documented in this KA for this FP

Stakeholders may require incompatible features, between requirements and resources or between functional and non-functional requirements; this requires the following tasks:

- Identify conflict;
- Consult with stakeholders to negotiate an acceptable compromise;
- Trace decision back to customer;
- Implement the decision.

## 6.4 Software design– description of an operational perspective

The “Software design” KA is composed of six subareas as is illustrated in Figure 6.5.

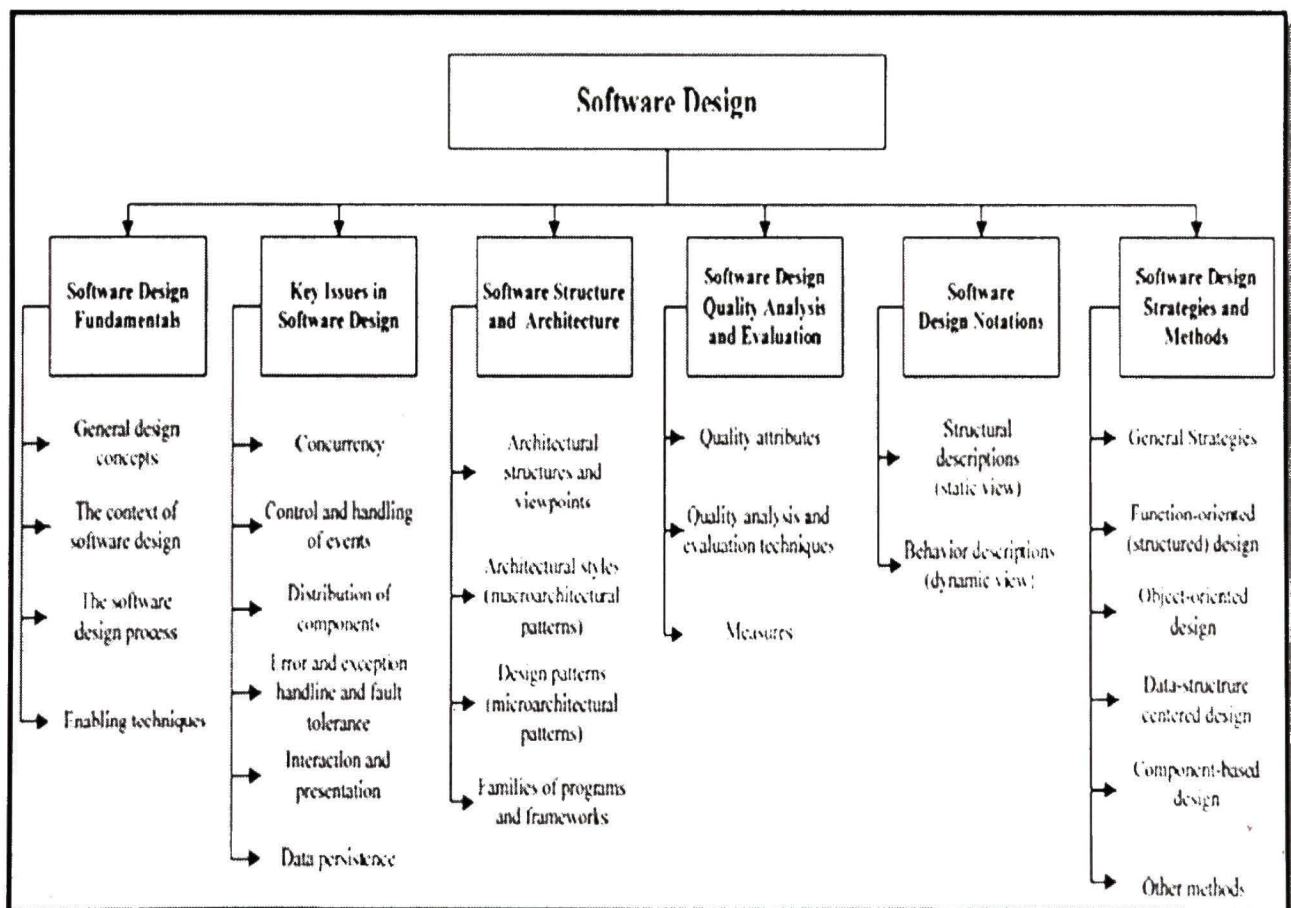


Figure 6.5: SWEBOK guide: “Software design” knowledge area (ISO-TR-19759, 2004)

#### **6.4.1 Principle #1** “Apply and use quantitative measurements in decision making”

##### **A. Presence of this FP in the taxonomy of this KA**

This fundamental principle is applied within the following “Software design” topic:

- Measures.

##### **B. Operational guidelines documented in this KA for this FP**

The measurement process can take place during the design phase to evaluate the size, structure or quality. Two types of measures are referred to in this KA:

- Function oriented (structured) design measures represented as a structure chart;
- Object oriented design measures represented as a class diagram. Different measures can be computed from both structure chart and class diagram.

#### **6.4.2 Principle #2** “Build with and for reuse”

##### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software design” topics:

- Design patterns (micro architectural patterns);
- Families of programs and frameworks;
- Component-based design.

##### **B. Operational guidelines documented in this KA for this FP**

“Build with and for reuse” can be applicable for the following elements: patterns, components, families of programs, and frameworks.

#### **6.4.3 Principle #4** “Implement a disciplined approach and improve it continuously”

##### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software design” topics and subareas:

- The software design process;
- Architectural structures and viewpoints;

- Architectural styles (macro architectural patterns);
- Design patterns (micro architectural patterns);
- Families of programs and frameworks;
- Structural descriptions (static view);
- Behavior descriptions (dynamic view);
- Software design strategies and methods (subarea);
- Enabling techniques;
- Keys issues in software design (subarea).

## **B. Operational guidelines documented in this KA for this FP**

Software design is composed of two steps: architectural design and detailed design. Software design produces solutions in the form of models.

### **Step 1: Architectural design**

In architectural design software is decomposed and organized into components using the following different elements:

- Use the general strategies to help guide the design process. For instance: divide-and-conquer and stepwise refinement, top-down vs. bottom-up strategies, data abstraction and information hiding, use of heuristics, use of patterns and pattern languages, iterative and incremental approach.
- Identify the views necessary to represent the system such as: logical view, physical view, process view and development view.
- Define the architectural style which describes the software high level organization:
  - General structure (for example, layers, pipes, and filters, blackboard);
  - Distributed systems (for example, client-server, three-tiers, broker);
  - Interactive systems (for example, Model-View-Controller, Presentation-Abstraction Control);
  - Adaptable systems (for example, micro-kernel, reflection);
  - Others (for example, batch, interpreters, process control, rule-based).

- Use the different methods for modeling the structural and behavioral descriptions of the system such as: function-oriented (structured) design, object-oriented design, data-structure-centered design, component-based design (CBD) and other methods.
- Model the structural description of the system which represents the static view using different notations. for instance: architecture description languages, class and object diagrams, component diagrams, class responsibility collaborator cards, deployment diagrams, entity-relationship diagrams, interface description languages, jackson structure diagrams, structure charts.
- Model the behavioral description of the system dynamic view such as:  
Activity diagrams, collaboration diagrams, data flow diagrams, decision tables and diagrams, flowcharts and structured flowcharts, sequence diagrams, state transition and statechart diagrams, formal specification languages, pseudo-code and program design languages

## **Step 2: Detailed design**

Detailed design describes the specific behaviour of the components already decomposed in the architectural step. In this step more low level details are given for the previous architectural design steps. Also in this step frameworks and design patterns are used to describe details at a lower level (the micro-architecture). For instance - some examples of design patterns:

- Creational patterns (builder, factory, prototype, and singleton);
- Structural patterns (adapter, bridge, composite, decorator, façade, flyweight, and proxy);
- Behavioural patterns (command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor).

## **Enabling techniques and key issues**

Enabling techniques and key issues are necessary to implement principle #4 “Implement a disciplined approach and improve it continuously” in the “Software Design”. Various enabling techniques and key issues are presented below:

**Enabling techniques:** Abstraction, coupling and cohesion, decomposition and modularization, encapsulation/information hiding, separation of interface and implementation, sufficiency, completeness and primitiveness.

**Key issues:** Concurrency, control and handling of events, distribution of components, error and exception handling and fault tolerance, interaction and presentation and data persistence.

**6.4.4 Principle #6** “Quality is the top priority; long term productivity is a natural consequence of high quality”

**A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software design” topics:

- Quality attributes;
- Quality analysis and evaluation techniques.

**B. Operational guidelines documented in this KA for this FP**

**Quality attributes:** apply the following set of quality attributes grouped by category to obtain a good quality design. For instance:

- The “ilities” (maintainability, portability, testability, traceability);
- Various “nesses” (correctness, robustness);
- Those discernable at run-time (performance, security, availability, functionality, usability);
- Those not discernable at run-time (modifiability, portability, reusability, integrability, and testability);
- Those related to the architecture’s intrinsic qualities (conceptual, integrity, correctness, and completeness, buildability).

**Quality analysis and evaluation techniques:** The following techniques can be applied to analyze and evaluate the quality of software design artefact such as:

- **Reviews**

Example (architecture reviews, design reviews, inspections, scenario-based techniques and requirements tracing).

- **Static analysis**

A static analysis technique evaluates a design (example, fault-tree analysis or automated cross-checking).

- **Simulation and prototyping**

Software design reviews, static analysis and simulation and prototyping are techniques to evaluate a design.

**6.4.5 Principle #8** “Since tradeoffs are inherent to software engineering, make them explicit and document them”.

**A. Presence of this FP in the taxonomy of this KA**

This FP is visible in the Introduction where trade-off can be used to examine and evaluate various alternative solutions. However, descriptive information is missing.

**6.4.6 Principle #9** “To improve design, study previous solutions to similar problems”.

**A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software design” topics:

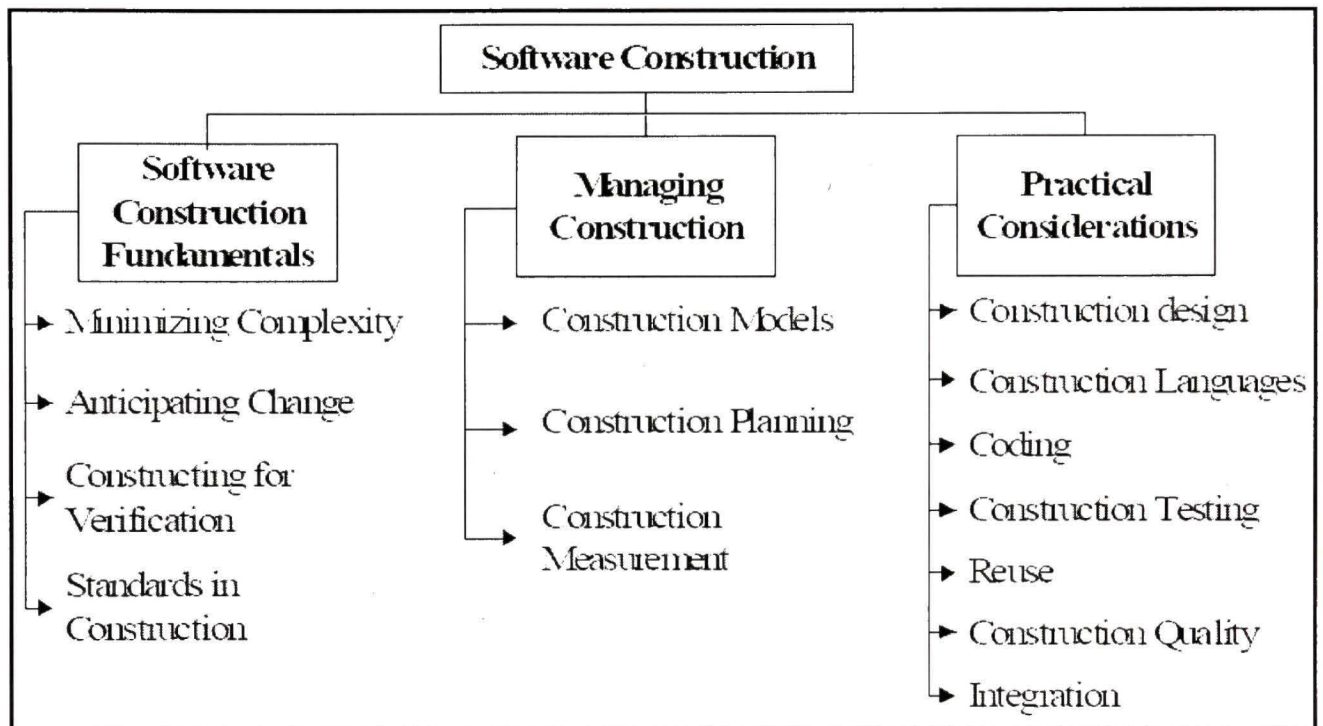
- Architectural styles (macro architectural patterns);
- Design patterns (micro architectural patterns).

**B. Operational guidelines documented in this KA for this FP**

The implementation of this FP is in part similar to principle #4 “Implement a disciplined approach and improve it continuously”. This can be found in architectural design step using architectural styles and in detailed design step using design patterns.

## 6.5 Software construction – description of an operational perspective

The “Software construction” knowledge area is composed of three subareas as illustrated in Figure 6.6



**Figure 6.6: SWEBOK Guide: Software construction knowledge area**  
(ISO-TR-19759, 2004)

**6.5.1 Principle # 1:** “Apply and use quantitative measurements in decision making”.

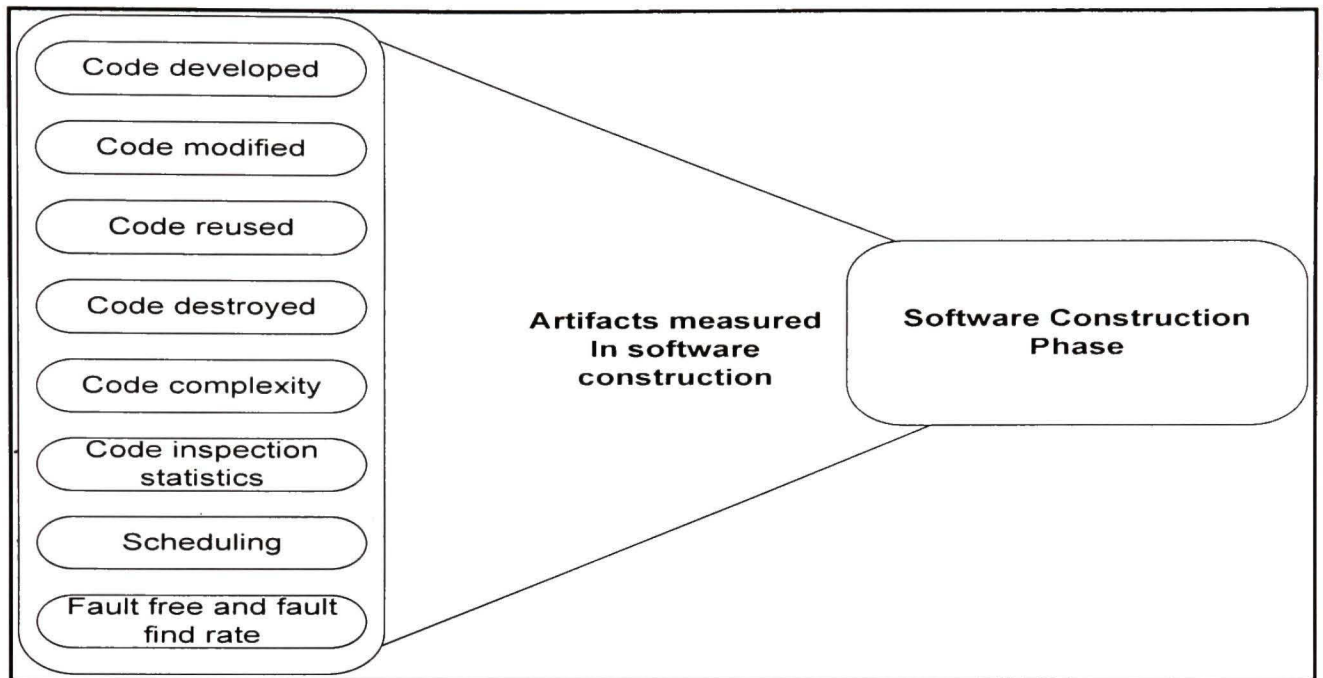
### A. Presence of this FP in the taxonomy of this KA

This fundamental principle (FP) is applied within the following topic:

- Construction measurement.

### B. Operational guidelines documented in this KA for this FP

Figure 6.7 presents the operational procedures for this FP in this Construction KA the different artifacts that can be measured during the “Software construction” phase, such as: code developed, code modified, code reused, code destroyed, code complexity.



**Figure 6.7: Measurement in the Construction KA**

### 6.5.2 Principle # 2: “Build with and for reuse”

#### A. Presence of this FP in the taxonomy of this KA

This FP is applied within the following “Software construction” topic:

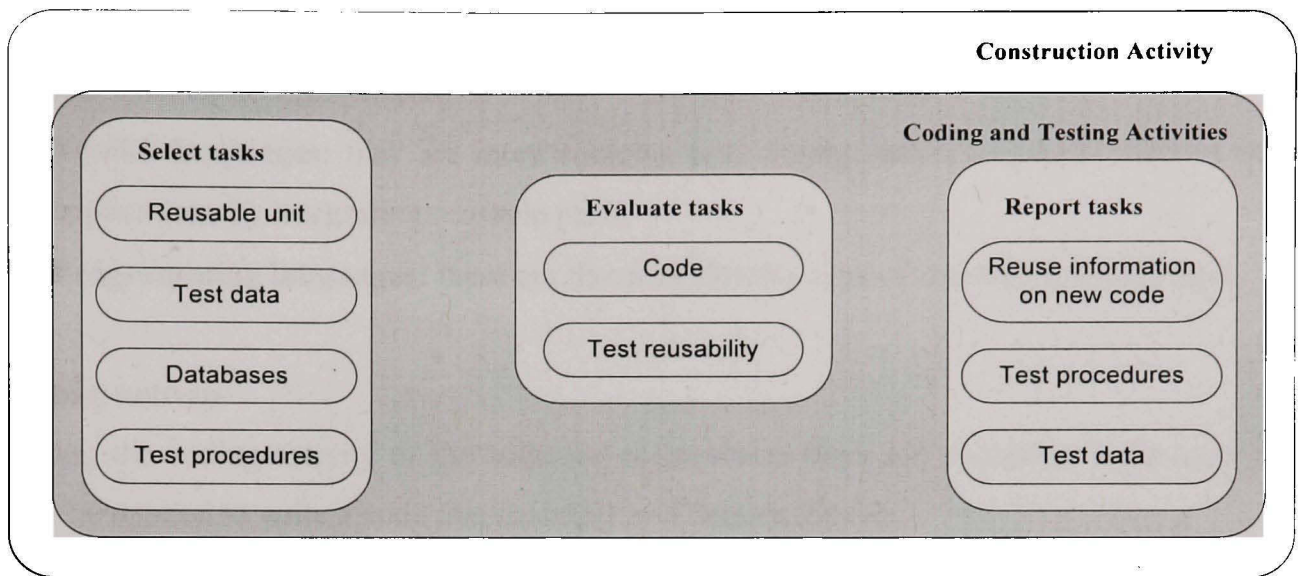
- Reuse.

#### B. Operational guidelines documented in this KA for this FP

In the “Software construction” KA during coding and testing activities, the different tasks related to the reuse activity are described as follows and are presented in Figure 6.8:

- Select tasks: reusable units, databases, test procedures, or test data;
- Evaluate tasks: code or test reusability;
- Reports tasks: reuse information on new code, test procedures, or test data.





**Figure 6.8: Construction activities- operational view in construction KA**

### 6.5.3 Principle # 4: “Implement a disciplined approach and improve it continuously”

#### A. Presence of this FP in the taxonomy of this KA

This FP is applied within the following “Software construction” topics:

- Construction design;
- Coding;
- Construction languages;
- Construction testing;
- Integration.

#### B. Operational guidelines documented in this KA for this FP

##### Construction design

In “Software construction” the design activity is similar to the “Software design” KA but, in the former, the design is done on a smaller scale.

##### Construction languages

There are three types of construction languages. Some of these construction languages include:

- **Configuration languages:** the text-based configuration files used in both the Windows and Unix operating systems;
- **Toolkit languages:** they are more complex than configuration languages used to build applications by integrating reusable parts;
- **Programming languages:** these are the most flexible type of construction languages.

### **Coding activity**

During the coding activity of the software construction there are numerous techniques that may be applied to write a code that is simple and understandable:

- Naming and code source layout;
- Use of classes, named constant;
- Control structures and handle error conditions;
- Prevent code-level security breaches;
- Source code organization into statements, classes;
- Document code;
- Code tuning;
- Resource usage via use of exclusion mechanisms and discipline in accessing serially reusable resources.

### **Testing activity**

The testing activity in the Construction KA involves two types of testing: unit and integration.

### **Integration activity**

To accomplish the integration task whether related to different routines, components, classes, and subsystems that are constructed during the construction activity (ISO-TR-19759, 2004):

- “Plan the sequence, in which components will be integrated”;
- “Create scaffolding to support interim versions of the software”;
- “Determine the degree of testing and quality work performed on components before they are integrated”;

- “Determine points in the project at which interim versions of the software are tested”.

**6.5.4 Principle # 6:** “Quality is the top priority; long term productivity is a natural consequence of high quality”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied into the following “Software construction” topics:

- Minimizing complexity;
- Constructing for verification;
- Standards in construction;
- Construction quality;
- Coding;
- Construction testing.

**B. Operational guideline documented in this KA for this FP**

#### **Minimizing complexity**

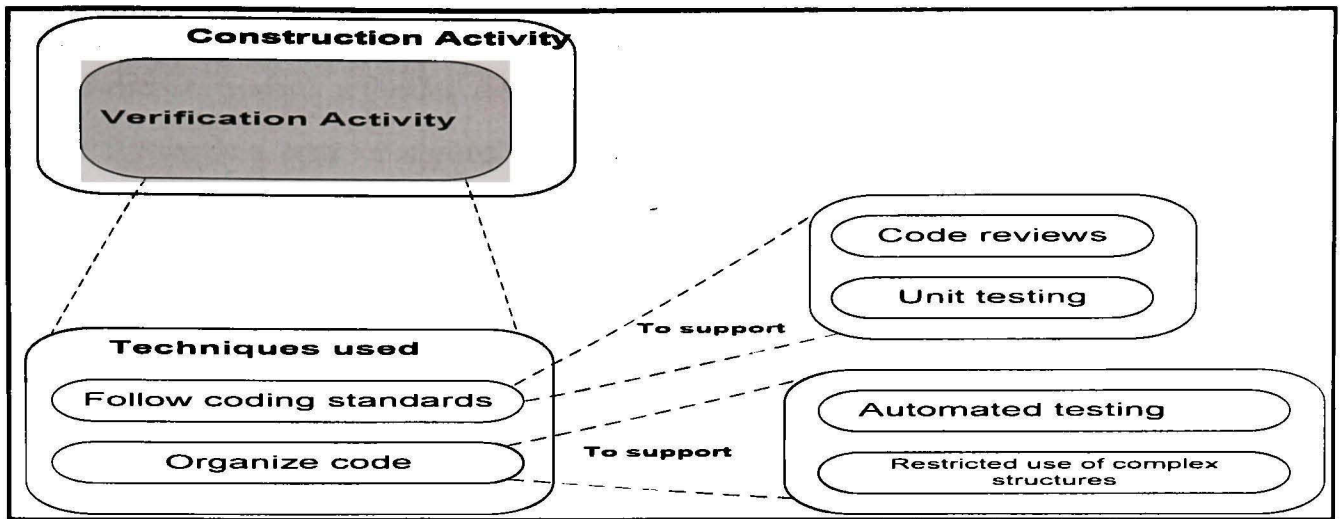
This topic is related to “standards in construction”, “coding” and “construction quality” topics. Minimizing complexity is achieved through many possibilities such as using:

- Standards described in “Standards in construction”;
- Specific techniques described in “Coding”;
- Quality techniques described in “Construction quality”.

#### **Constructing for verification**

The verification activity is important in the software construction phase. Specific tasks that support constructing for verification include the following:

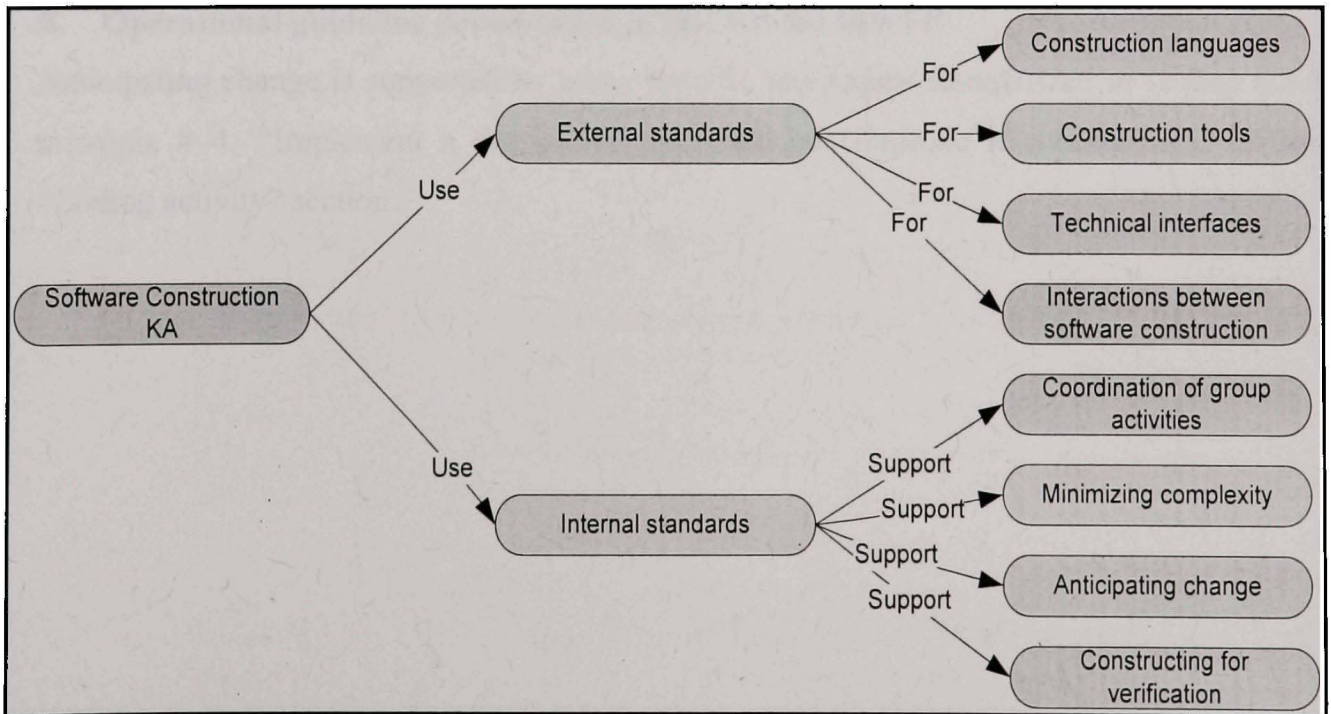
- Follow coding standards to support code reviews, unit testing;
- Organize the code to support automated testing and restricted use of complex or hard-to-understand language structures. See Figure 6.9.



**Figure 6.9: Constructing for verification- operational view in Construction KA**

**Standards in construction**

The “Software construction” KA uses external and internal standards such as: construction languages, construction models, constructing for verification etc. For more details, see Figure 6.10.



**Figure 6.10: Standards- operational view in Construction KA**

**Construction quality**

Construction quality activities are performed on code and on artifacts that are related to code. To write a code of a good quality during software construction, several techniques exist, including: unit testing and integration testing, test-first development (see also the Software Testing KA), code stepping, use of assertions, debugging, technical reviews.

**Coding activity and construction testing activities**

The related details for these two activities are already described in section 6.5.3 principle # 4: “Implement a disciplined approach and improve it continuously”.

**6.5.5 Principle (#7): “Since change is inherent to software, plan for it and manage it”**

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied into the following “Software construction” topic:

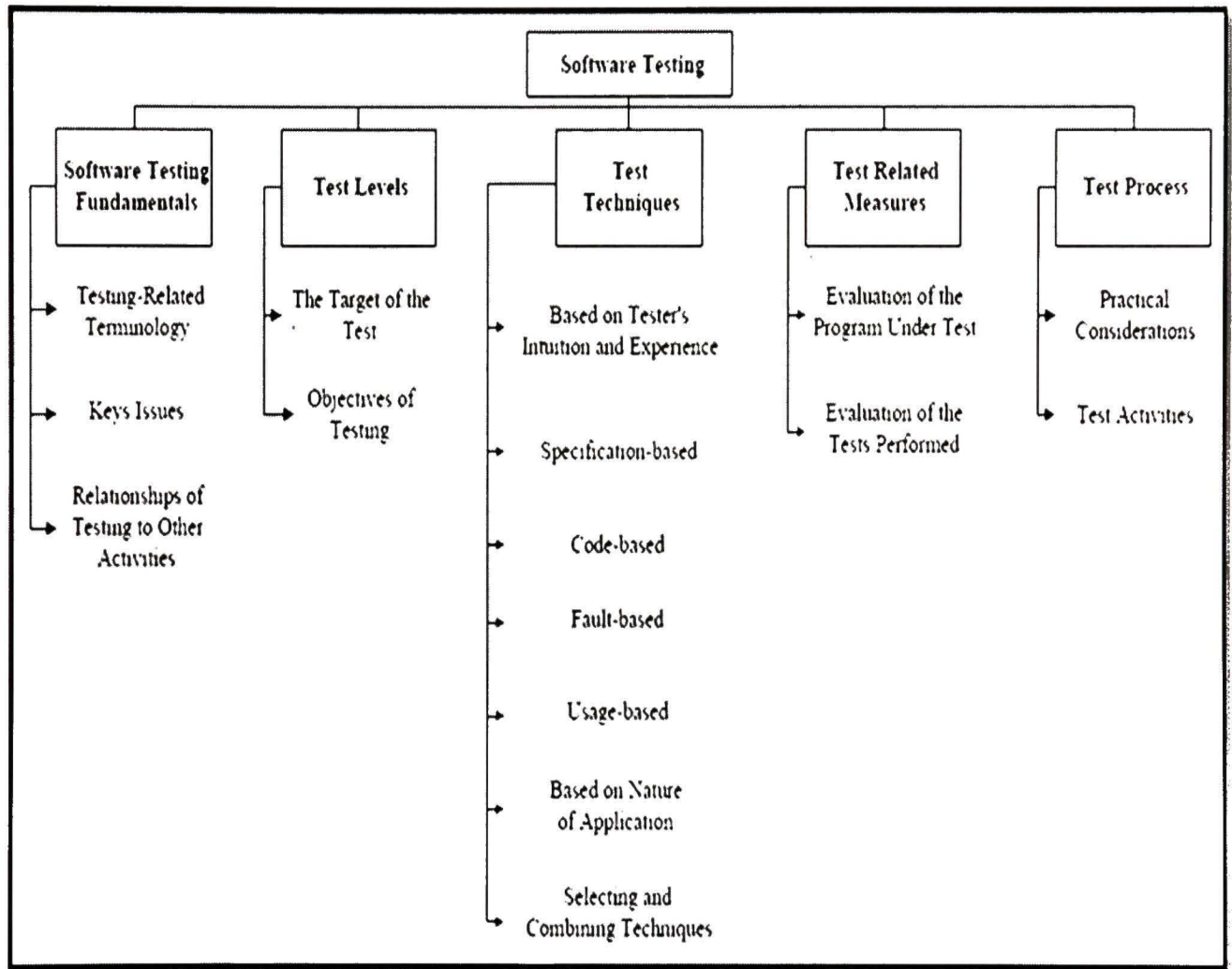
- Anticipating change.

**B. Operational guideline documented in this KA for this FP**

Anticipating change is supported by many specific techniques summarized in section 6.5.3 principle # 4: “Implement a disciplined approach and improve it continuously” in the “Coding activity” section.

## 6.6 Software testing– description of an operational perspective

The “Software testing” KA is composed of five subareas - see-Figure 6.11.



**Figure 6.11: SWEBOK guide: Software testing knowledge area**  
(ISO-TR-19759, 2004)

### 6.6.1 Principle (#1) “Apply and use quantitative measurements in decision making”

**A. Presence of this FP in the taxonomy of this KA:** this fundamental principle is applied into the following “Software testing” topics and sub-topic:

- Evaluation of the program under test;
- Evaluation of the tests performed;
- Cost/effort estimation and other process measures (sub-topic).

## **B. Operational guidelines documented in this KA for this FP**

The test-related measures evaluate the program under test based on the observed test outputs.

Figure 6.12 presents the evaluation of the program under test.

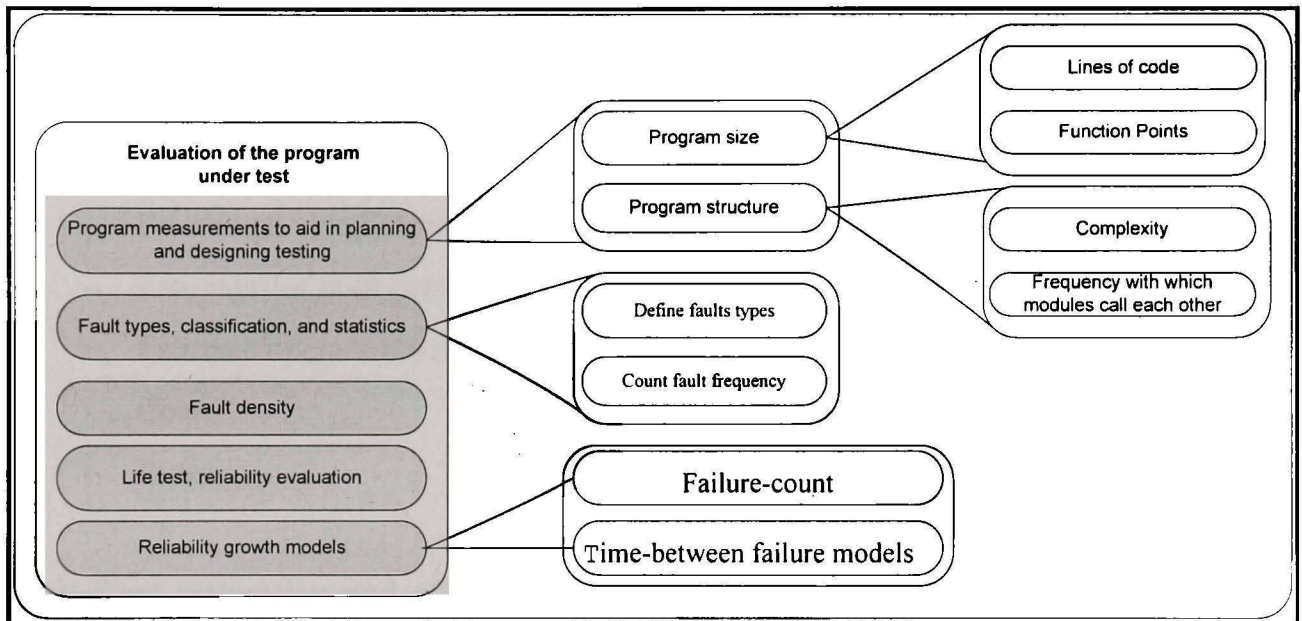
### **Evaluation of the program under test**

To evaluate a program under test, the following test related measures could be collected:

- **Program measurements to aid in planning and designing testing**  
To guide testing apply the following measures such as:
  - Program size (for example, source lines of code or function points);
  - Program structure (like complexity, frequency with which modules call each other).
- **Definition of fault types, classification and statistics**  
To make testing more effective:
  - Define faults types;
  - Count the relative fault frequency.
- **Measure of fault density:** a program under test can be assessed by:
  - Counting the discovered faults;
  - Classifying the discovered faults by their types;
  - Measuring the fault density (the ratio between the number of faults found and the size of the program) for each fault class.
- **Life test, reliability evaluation**  
To decide when to stop a test:
  - Evaluate a product by using a statistical estimate of software reliability (see SWEBOK Guide section 2.2.5).
- **Reliability growth models**

“Reliability growth models provide a prediction of reliability based on the failures observed under reliability achievement and evaluation” (see “Software testing” section 2.2.5) (ISO-TR-19759, 2004). These models are divided into:

- Failure-count;
- Time-between failure models.



**Figure 6.12: Evaluation of the program under test**

### Evaluation of the tests performed

To evaluate the tests performed the following test related measures could be done:

- **Coverage/thoroughness measures**

- Evaluate the thoroughness of the executed tests by choosing the test cases that exercise a set of elements identified in the program or in the specifications;
- Measure dynamically the ratio between covered elements and their total number.
- Example:
  - Measure the percentage of covered branches in the program flowgraph;
  - Measure the functional requirements exercised among those listed in the specifications document.



- **Fault seeding:** before test inserts fault into the program. Some measures include:
  - The number of artificial faults discovered;
  - The number of testing effectiveness;
  - Estimation of the remaining number of genuine faults.
- **Mutation score:** to measure the effectiveness of the executed test set:
  - Measure the ratio of killed mutants to the total number of generated mutants.
- **Termination:** to decide when to stop tests the following thoroughness measures can help, such as:
  - Achieved code coverage;
  - Functional completeness;
  - Estimates of fault density;
  - Estimate operational reliability;
  - Cost;
  - Risks.

**Cost/effort estimation and other process measures:** measures relative to the control and the improvement of the test process for management purposes include:

- Measure the resources spent on testing;
- Measure the relative fault-finding effectiveness of the various test phases.

These tests measures cover the following elements:

- Number of test cases specified;
- Number of test cases executed;
- Number of test cases passed;
- Number of test cases failed.

Evaluate test process effectiveness by evaluating:

- Test phase reports;
- Root cause analysis.

### 6.6.2 Principle (#2) “Build with and for reuse”

**A. Presence of this FP in the taxonomy of this KA:** this fundamental principle is applied within the following “Test process” subarea:

- Practical considerations.

#### **B. Operational guidelines documented in this KA for this FP**

**Build with and for reuse:** reuse the test material used to test the software. This test material should also be documented so that it can be reused as in: test cases.

### 6.6.3 Principle (#4) “Implement a disciplined approach and improve it continuously”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied into the following “Software testing” topics and subarea:

- The target of the test (topic);
- Objectives of testing (topic);
- Test activities (topic);
- Test techniques (subarea).

#### **B. Operational guidelines documented in this KA for this FP**

The test process is composed of several activities from planning to defect tracking. Figure 6.13 presents the different testing activities. The operational details related to those activities are as follows:

- **Plan the testing activities**

The different steps for one baseline of the software include:

- Coordinate personnel;
- Manage available test facilities and equipment (which may include magnetic media, test plans and procedures);
- Plan for possible undesirable outcomes.

- **Generate test-cases :**to generate test cases the following tasks should be done:

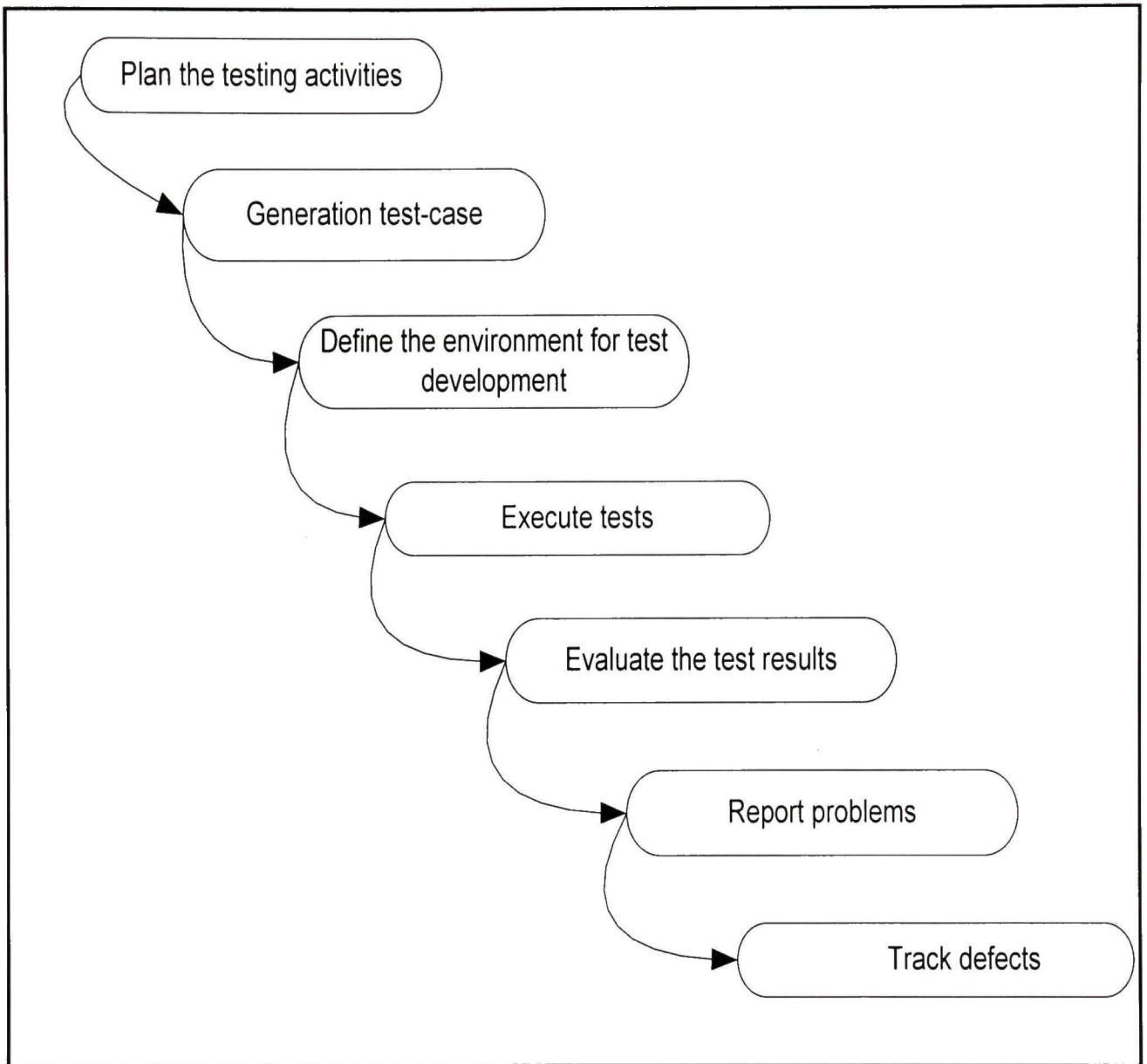
- Define the target of the test - see test levels section;
- Define the objective of the test - see test levels section;
- Identify the techniques that are used for testing - see test techniques section;
- Put the control of test cases under the software configuration management;
- For each test case include the expected results.
- **Define the environment for test development**
  - Check the compatibility for the testing environment with the software engineering tools;
  - Test environment should facilitate development and control of test cases, logging and recovery of expected results, scripts, and other testing materials.
- **Execute the tests** :during the execution of tests everything done should be:
  - Performed and documented clearly enough that another person could replicate the results;
  - Performed in accordance with documented procedures using a clearly defined version of the software under test.
- **Evaluate the test results**

The results of tests are determined by success or failure. When a failure is identified before it can be removed:

- Analyze and debug to isolate, identify, and describe a failure;
  - Evaluate the test result with a formal review board if they are important.
  - **Report problems related to testing activities/ Test log:**
- Below is presented a list of information that can be reported into a test log or a problem-reporting system:
- When a test was conducted;
  - Who performed the test;
  - What software configuration was the basis for testing;
  - And other relevant identification information;
  - Record incorrect test results in a problem-reporting system;
  - Document anomalies not classified as faults.

Tests reports are also an input to the change management requests process (see the Software Configuration Management KA, subarea 3: Software Configuration Control).

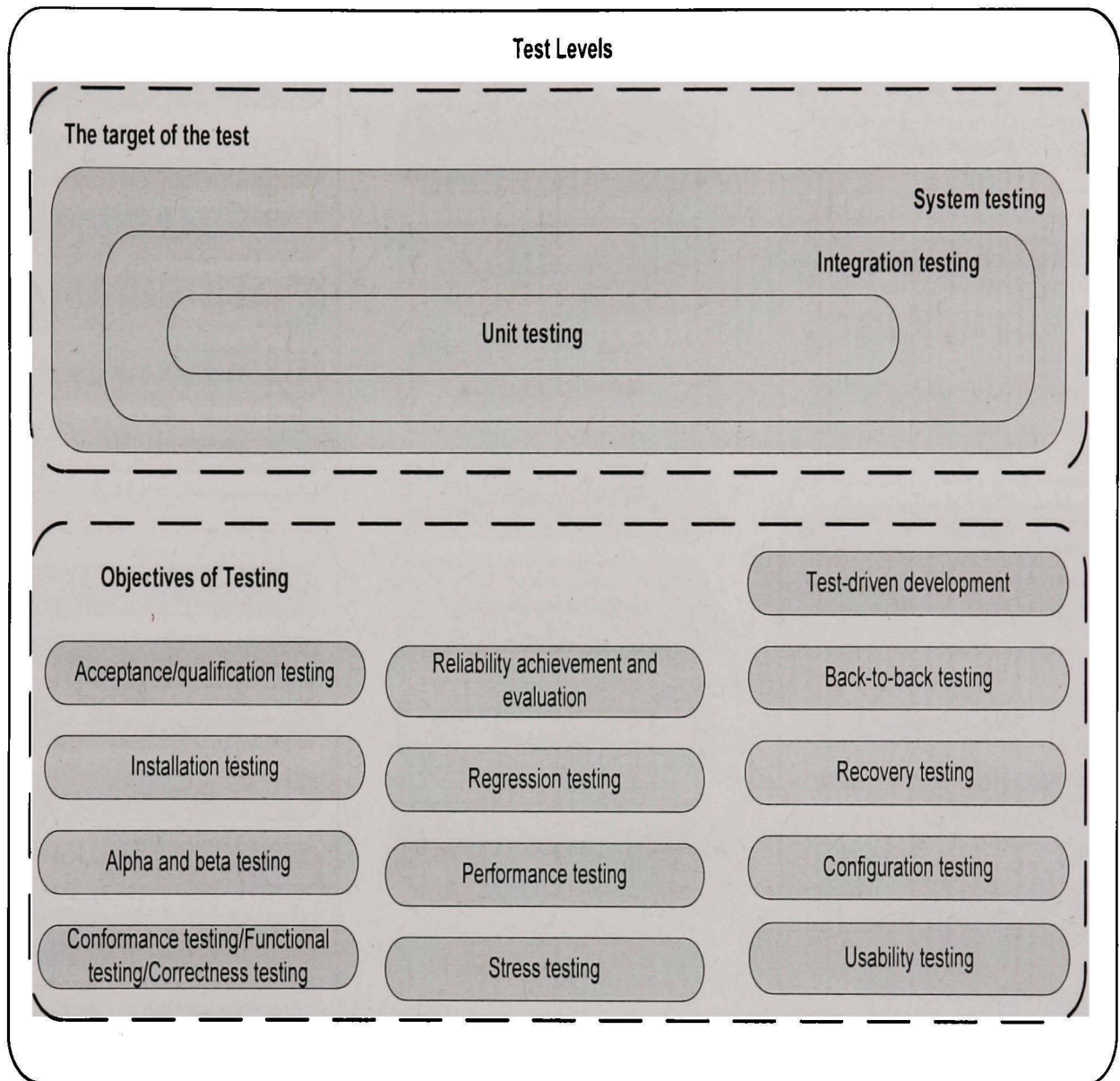
- **Track defects for later analysis :**
  - When they were introduced into the software;
  - What kind of error caused them to be created (poorly defined requirements, incorrect variable declaration, memory leak, programming syntax error);
  - When these errors could have been first observed in the software.
- **Defect-tracking information is used to determine:**
  - What aspects of software engineering need improvement;
  - How effective previous analyses and testing have been.



**Figure 6.13: Phases for the testing activities- operational view in Testing KA**

### Test levels

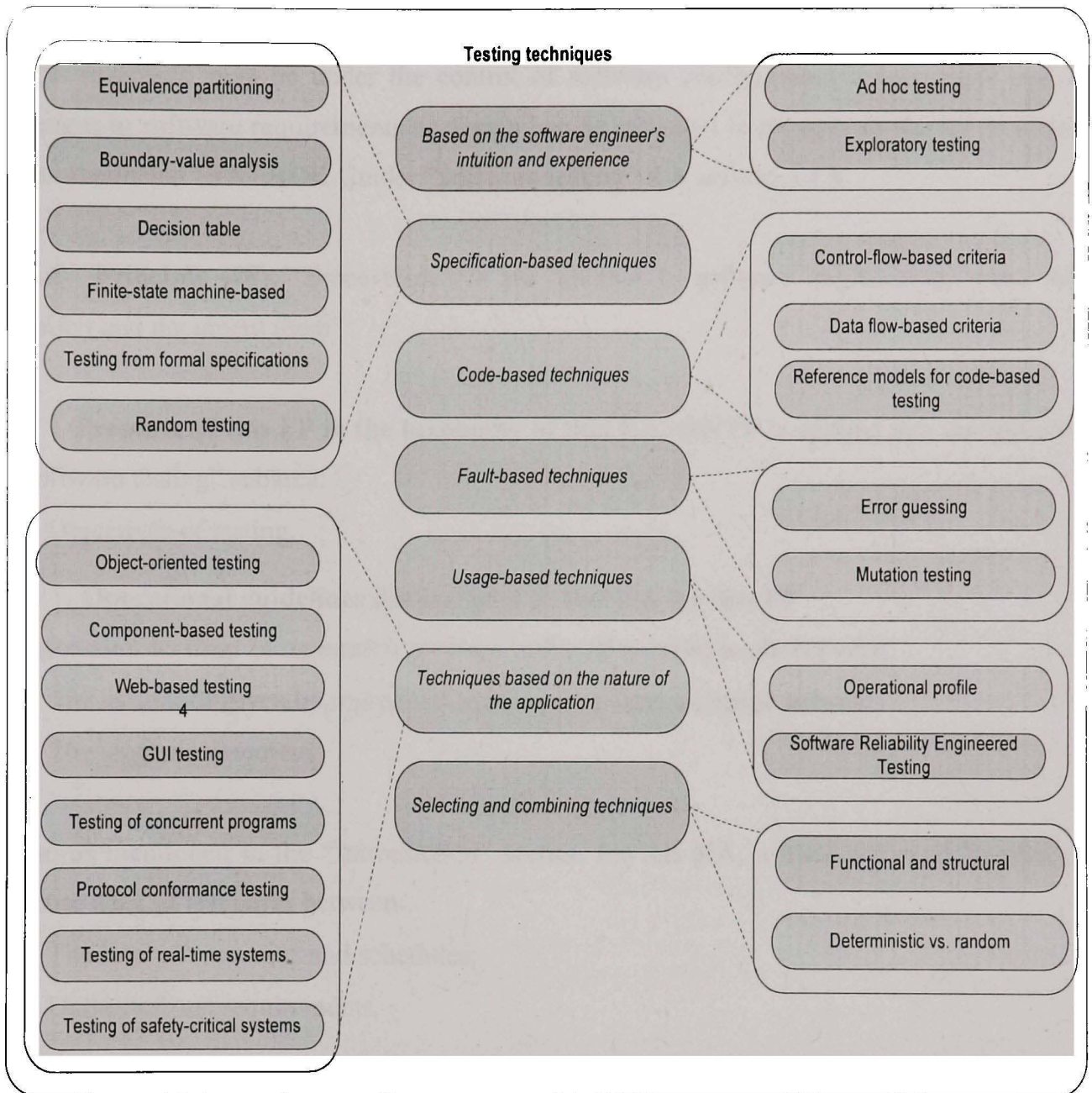
The test level defines the target of the test and the objectives of the test. The target of the test is divided into three levels; unit, integration and system testing. Figure 6.14 illustrates a model related to the test levels.



**Figure 6.14: Phases for the testing levels - operational view in testing KA**

### Test techniques

To test software various techniques are defined. Some of these techniques include specification based techniques, code based techniques and techniques based on the nature of the application. More details about those techniques are presented in Figure 6.15.



**Figure 6.15: Testing techniques- operational view in testing KA**

#### 6.6.4 Principle (#7): “Since change is inherent to software, plan for it and manage it”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied into the following “Test process” subarea:

- Practical considerations.

**B. Operational guidelines documented in this KA for this FP**

“Test materials must be under the control of software configuration management, so that changes to software requirements or design can be reflected in changes to the scope of the - tests conducted” SWEBOK Guide. “Software testing” KA section 5.1.8.

**6.6.5 Principle (#8):** “Since tradeoffs are inherent to software engineering, make them explicit and document them”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied into the following “Software testing” subarea:

- Objectives of testing.

**B. Operational guidelines documented in this KA for this FP**

**Regression testing:** In regression testing a trade-off must be made between:

- The assurance given by regression testing every time a change is made;
- The required resources.

Also as mentioned in the “Introduction” section for this KA, a trade off must be made to choose a set of test cases between:

- The limited resources and schedules;
- Unlimited test requirements.



## 6.7 Software maintenance– description of an operational perspective

The “Software maintenance” KA is composed of four subareas – see- Figure 6.16.

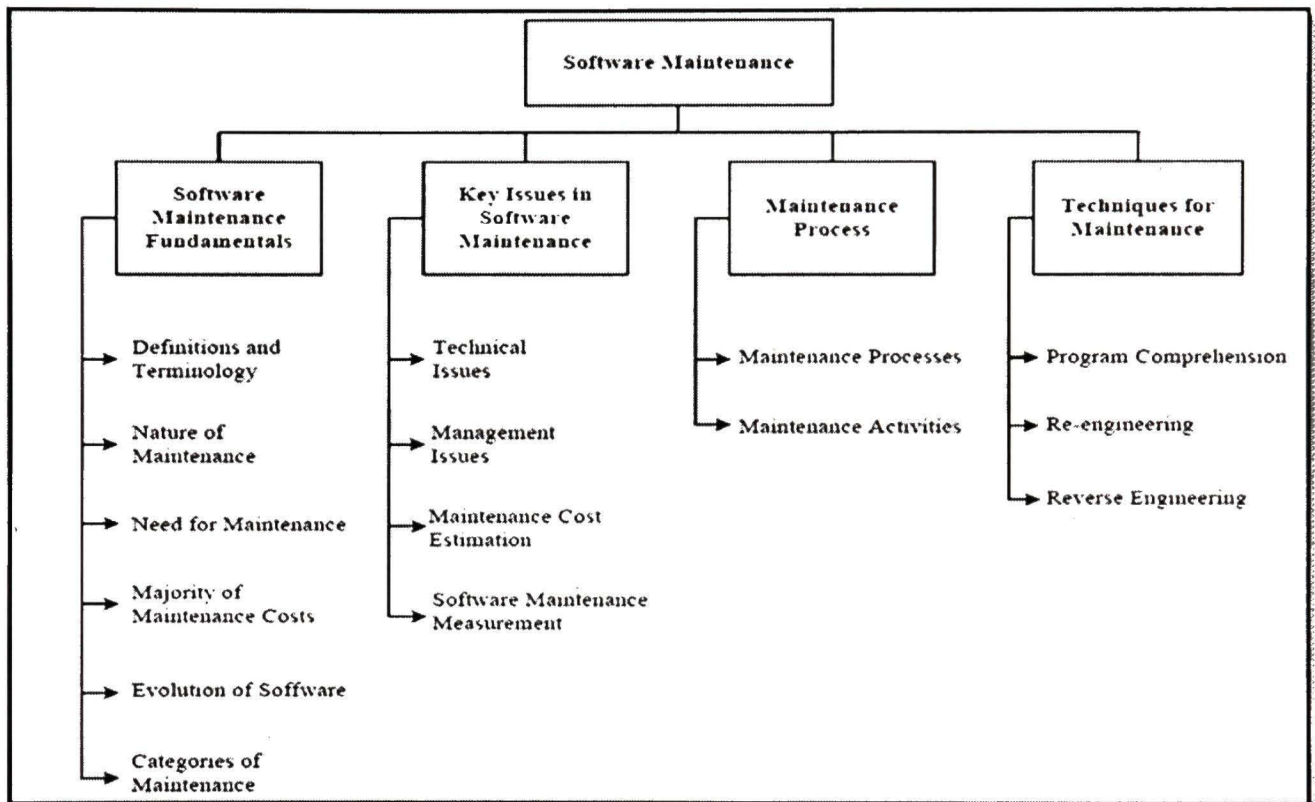


Figure 6.16: SWEBOK Guide: “Software maintenance” knowledge area (ISO-TR-19759, 2004)

### 6.7.1 Principle (#1): “Apply and use quantitative measurements in decision making”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software maintenance” topics:

- Maintenance cost estimation;
- Software maintenance measurement.

**B. Operational guidelines documented in this KA for this FP**

**Maintenance cost estimation:** The maintenance cost estimation could be done for planning purposes. To estimate resources for software maintenance apply the following approaches:

- Parametric models;
- Experience:
  - Expert judgment (for example Delphi technique);
  - Analogies;
  - A work breakdown structure;
  - Combine empirical data and experience.
- Combine both approaches.

### **Software maintenance measurement**

- Measures common to all endeavors: The software engineering Institute (SEI) has identified the following measures that will be useful for the maintainer: size, effort, schedule and quality.
- Internal benchmarking techniques: The maintainers determine which of the following specific measures: analyzability, changeability, stability and testability fit for the organization.

#### **6.7.2 Principle (#4): “Implement a disciplined approach and improve it continuously”**

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software maintenance” topics:

- Maintenance processes;
- Maintenance activities.

#### **B. Operational guidelines documented**

**Maintenance processes:** The Maintenance Process subarea provides references and standards used to implement the software maintenance process.

- Standard for software maintenance (IEEE1219-98);
- ISO 1476.

**Maintenance activities:** Maintenance activities are composed of the same activities that are in the software development for instance: analysis, design, coding, testing and documentation. In addition there are some activities that are unique to software maintenance and other supporting activities. See - Figure 6.17.

- a. Unique activities :** The unique activities for “Software maintenance” are described as
- Transition: Transfer software from the developer to the maintainer;
  - Modification request acceptance/rejection;
  - Modification request and problem report help desk;
  - Impact analysis;
  - Software support;
  - Service level agreements (SLAs) and specialized (domain-specific): maintenance contracts which are the responsibility of the maintainers.
- b. Supporting activities:** Below is a list of activities that support maintenance, such as:
- Software maintenance planning;
  - Software configuration management;
  - Software quality.

**b1 Maintenance planning activity:** There are four perspectives to consider for maintenance activities as follows:

○ **The individual (request level)**

- Planning is carried out during the impact analysis.

○ **The release/version planning activity (software level)**

- Collect the dates of availability of individual requests;
- Agree with users on the content of subsequent releases/versions;
- Identify potential conflicts and develop alternatives;
- Assess the risk of a given release and develop a back-out plan in case problems should arise;
- Inform all the stakeholders.

○ **Maintenance planning (transition level)**

- Estimates resources;
- Include those resources in the developers' project planning budgets;
- Decide to develop a new system;
- Consider quality objectives;
- Develop a concept document;
- Develop a maintenance plan.

Prepare the concept document for maintenance (ISO12207) [s7.2] that addresses:

- The scope of the software maintenance;
- Adaptation of the software maintenance process;
- Identification of the software maintenance organization;
- An estimate of software maintenance costs;

Prepare the maintenance plan during software development, and specify:

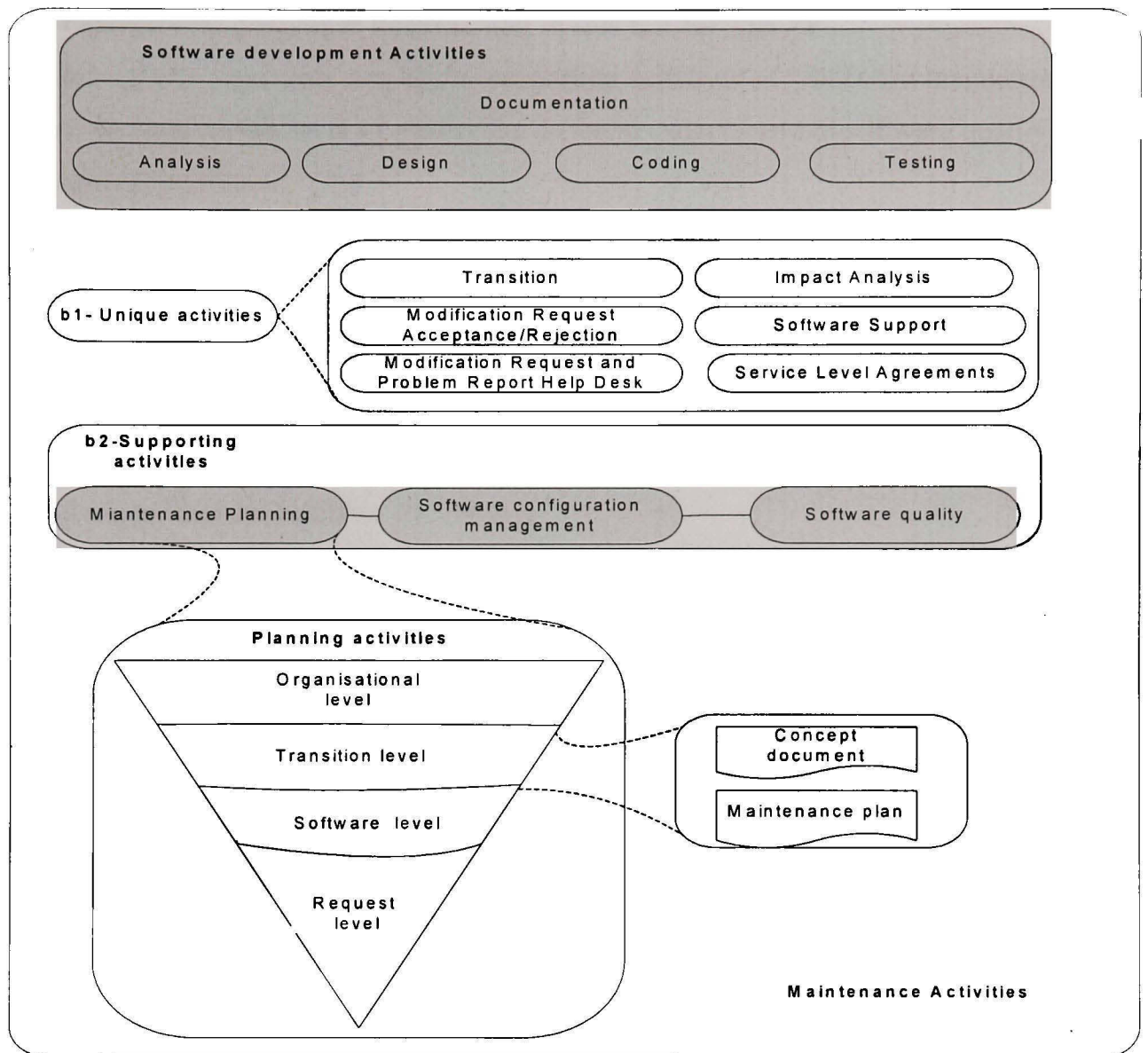
- How users will request software modifications;
- How users will report problems.

○ **Business planning (organizational level)**

- Conduct business planning activities (budgetary, financial, and human resources).

**b2. Software configuration management:** Software configuration management procedures should:

- Verify, validate and audit every step essential to identify, authorize, implement and release the software product.



**Figure 6.17: Maintenance activities- operational view in maintenance KA**

**6.7.2 Principle (#6):** “Quality is the top priority; long term productivity is a natural consequence of high quality”.

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following topic:

- Maintenance activities.

## **B. Operational guidelines documented in this KA for this FP**

Software quality represents one of the supporting activities of “Software maintenance”. To achieve the appropriate level of quality the different tasks related to software quality need to be completed as follow:

- Plan quality;
- Plan processes implemented to support the maintenance process;
- Select the activities and techniques for software quality assurance (SQA), verification & validation, reviews, and audits;
- A recommendation: The maintainer should adapt the software development processes, techniques and deliverables, for instance:
  - Testing documentation;
  - Test results.

### **6.6.4 Principle (#7): “Since change is inherent to software, plan for it and manage it”**

#### **A. Presence of this FP in the taxonomy of this KA**

This FP is applied within the following “Software maintenance” subtopics under “maintenance activities”:

- Software configuration management (maintenance activities).

#### **B. Operational guidelines documented in this KA for this FP**

Here are presented a few steps on how to perform software configuration management:

- Control the changes made to a software product;
- Establish the control by implementing and enforcing an approved software configuration management process.

## 6.8 Software configuration management: operational perspective of the software engineering FP

The “Software configuration management” KA is composed of six subareas - see Figure 6.18.

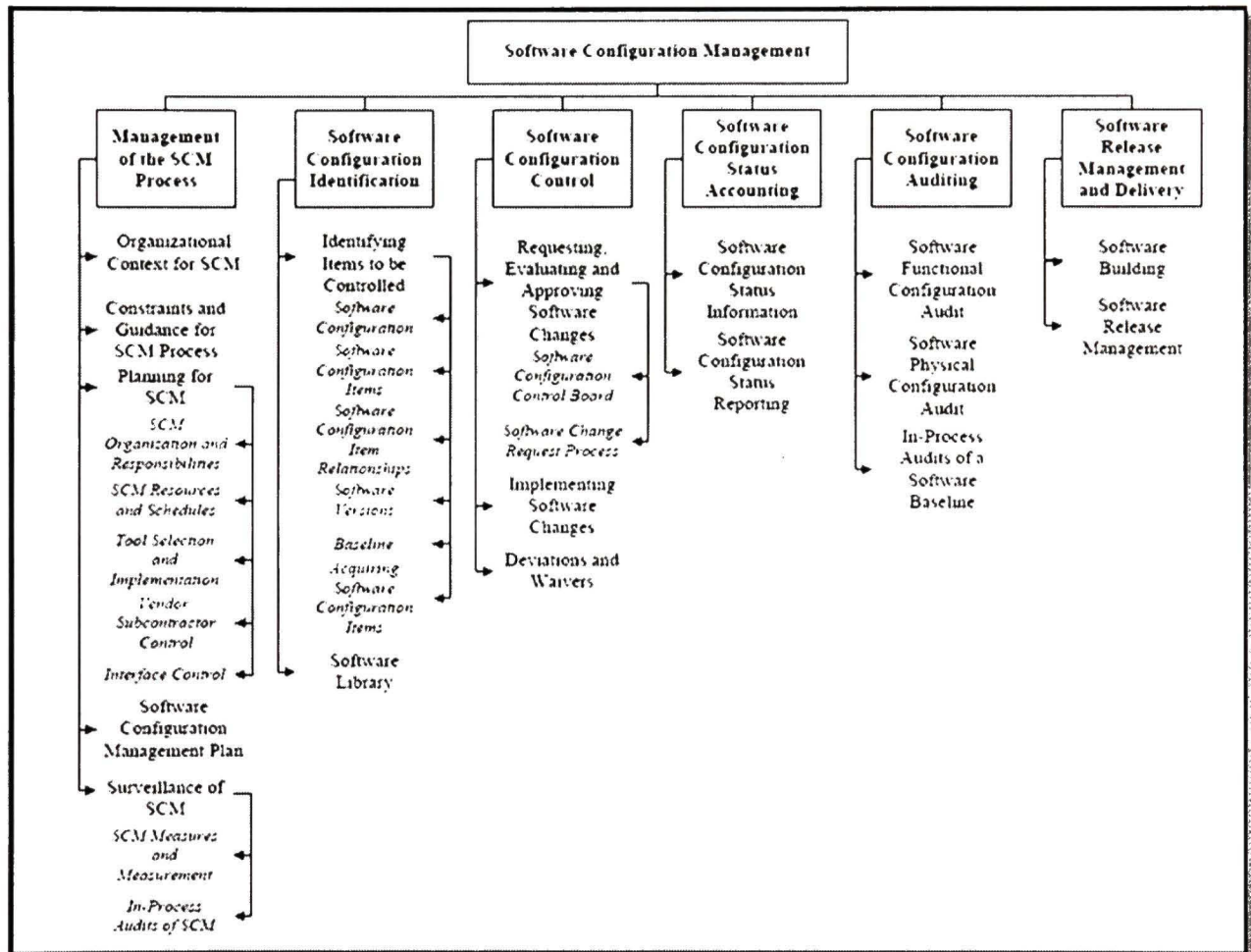


Figure 6.18: SWEBOK guide: “Software configuration management” knowledge Area (ISO-TR-19759, 2004)

### 6.8.1 Principle (#1): “Apply and use quantitative measurements in decision making”

A. **Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software Configuration Management” topic:

- The measurement “FP no. 1 - Apply and use quantitative measurements in decision making” is listed on “Surveillance of software configuration management” topic under “Management of the SCM process” subarea but is not described.

**B. Operational guidelines documented in this KA for this FP:** there is no description on how to apply the measures.

#### 6.8.2 Principle (#4): “Implement a disciplined approach and improve it continuously”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Configuration management” subareas:

- Software configuration identification;
- Software configuration control;
- Software configuration status accounting;
- Software configuration auditing;
- Software release management and delivery.

**B. Operational guidelines documented in this KA for this FP**

##### **Software configuration identification**

The software configuration identification tasks are as follows:

- Identifies items to be controlled;
- Establishes identification schemes for the items and their versions;
- Establishes the tools and techniques to be used in acquiring and managing controlled items.

**Software configuration control:** the practical details for these activities are similar to those mentioned later for principle # 7: “Since change is inherent to software, plan for it and manage it”.



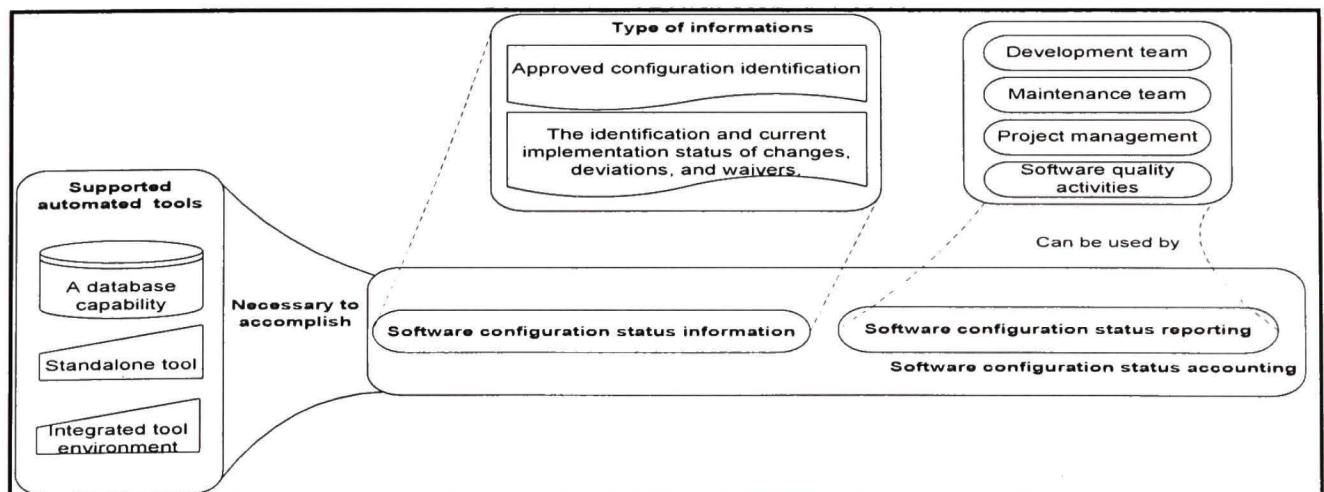
**Software configuration status accounting:** the software configuration status accounting is composed of two parts: the software configuration status information and the software configuration status reporting. The software configuration status information and reporting necessitate the support of various automated tools. For more details

Figure 6.19 presents a model for software configuration status accounting.

**Software configuration auditing:** software configuration auditing is composed of two parts: software functional configuration audit and software physical configuration.

**Software release management and delivery:** the different tasks related for software release management and delivery are presented as follows:

- **Software building**
  - Build software using compilers.
- **Software release management:** software release management contains the following tasks:
  - Identification;
  - Packaging.
  - Identify which product items are to be delivered.



**Figure 6.19: Software configuration status accounting- operational view in the Configuration Management KA**

- Select the correct variants of those items, given the intended application of the product.
- Delivery of the elements of a product.

### 6.8.3 Principle (#7): “Since change is inherent to software, plan for it and manage it”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Configuration management” subarea:

- Software configuration control.

### **B. Operational guidelines documented in this KA for this FP**

Software configuration control is concerned with managing changes during the software life cycle. Software configuration control covers the following tasks:

#### **Determine what changes to make**

- Initiate a corrective action in response to problem reports;
- Record the change request on the SCR (software change request) which may include a suggested solution and requested priority;
- Submit recorded change requests;
- Evaluate the potential cost;
- Perform a technical evaluation to evaluate the impact of a proposed change also known as impact analysis.

#### **Software configuration control board**

- An established authority, commensurate with the affected baseline, the SCI and the nature of the change;
- An established authority will evaluate the technical and managerial aspects of the change request and either accept, modify, reject, or defer the proposed change.

#### **Implementing software changes**

- Implement approved SCRs using the defined software procedures in accordance with the applicable schedule requirements;

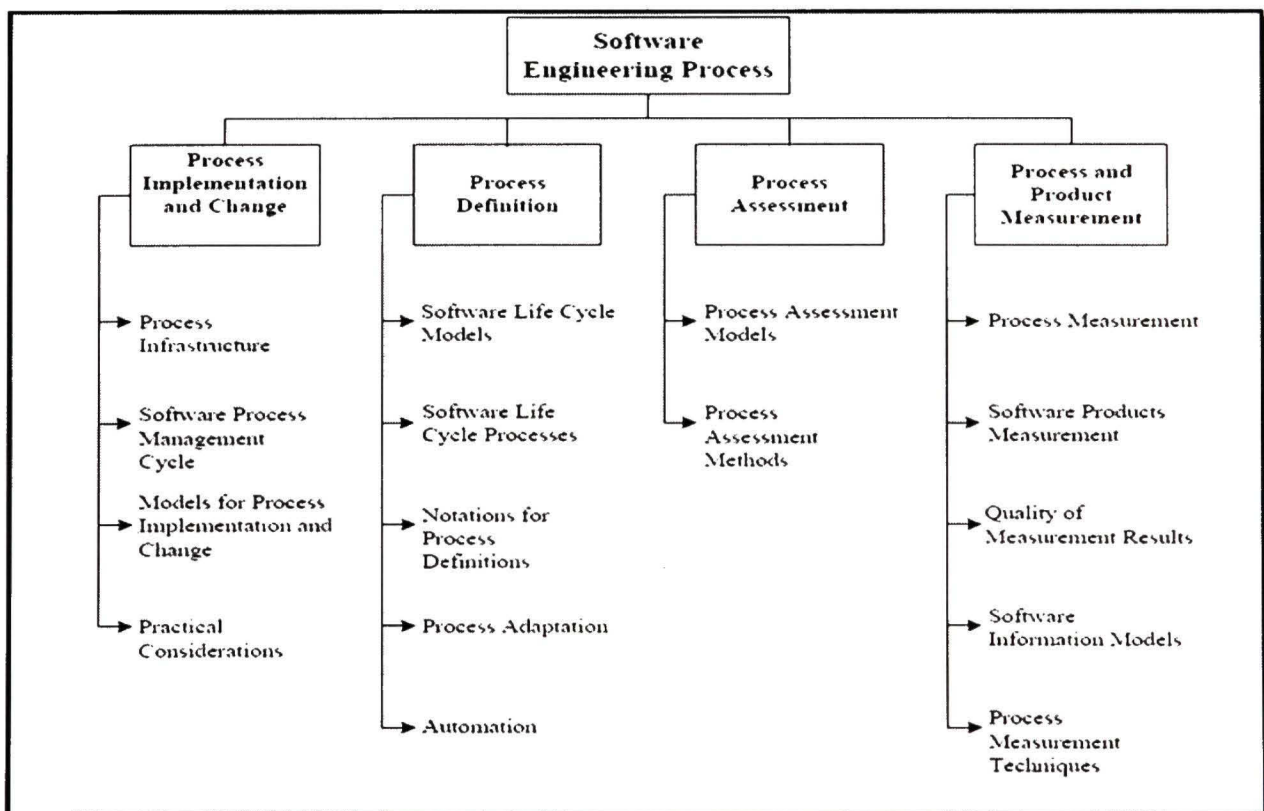
- Provide a means for tracking which SCRs are incorporated, since a number of approved SCRs might be implemented simultaneously into particular software versions and baselines;
- As part of the closure of the change process;
  - Completed changes may undergo configuration audits;
  - Software quality verification to ensure that only approved changes have been made.

### Deviations and waivers

- Identify the exception of deviation;
- Get the authorization of a waiver.

## 6.9 Software engineering process– description of an operational perspective

The “Software engineering process” KA is composed of four subareas see - Figure 6.20.



**Figure 6.20: SWEBOK guide: Software engineering process knowledge area**  
(ISO-TR-19759, 2004)

### 6.9.1 Principle (#1): “Apply and use quantitative measurements in decision making”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software engineering process” subarea:

- Process and product measurement.

### **B. Operational guidelines documented in this KA for this FP**

Measurement could be applied on processes and product as in the “Software process” KA.

#### **To measure processes**

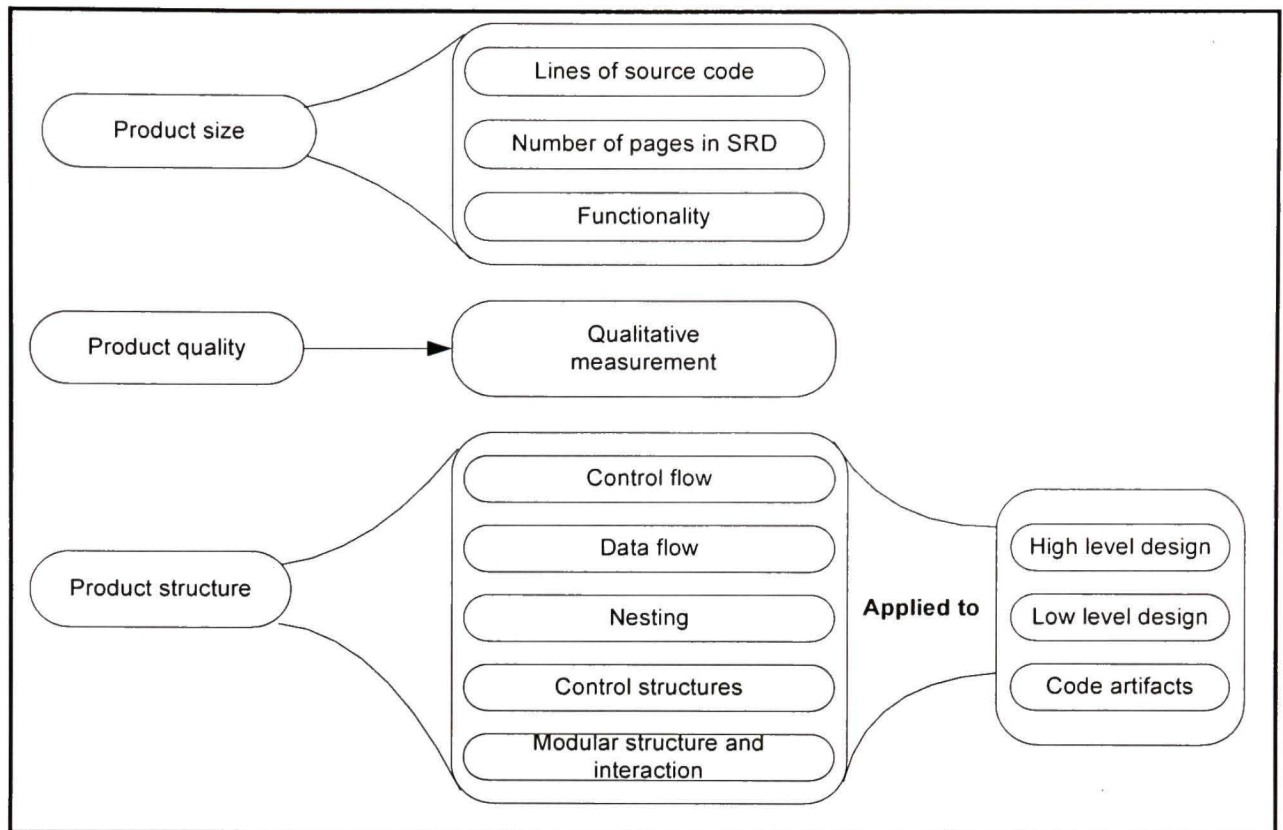
- Apply measures on processes productivity or team productivity. For example,
  - Measure function points produced per unit of person-effort.
- Apply measures on process outcomes. For example:
  - Product quality (faults per KLOC (Kilo lines of code) or per Function point (FP));
  - Maintainability (the effort to make a certain type of change);
  - Productivity (LOC (Lines of code) or Function points per person-month);
  - Time-to-market or customer satisfaction (as measured through a customer survey).

#### **To measure product**

Measure product size (lines of code, number of pages in software requirements documents and functionality):

- Measure product structure;
- Measure product quality.

More details on product measurement are presented in Figure 6.21.



**Figure 6.21: Related product measurements- operational view in the Process KA**

### Assess the quality of the following measurement results

- Accuracy;
- Reproducibility;
- Repeatability;
- Convertibility;
- Random measurement errors.

### Build information models

Build information models based on the measurement data collected.

### Use of techniques

Use the analytical techniques and the benchmarking techniques to analyze processes and to identify their strengths and weaknesses.

- **Analytical techniques**
  - Experimental studies:
    - Set up controlled or quasi experiments in the organization;
    - Compare a new process with the current one to evaluate if the former has a better outcome.
  - Process simulation;
  - Process definition review:
    - Review the process definition;
    - Identify deficiencies and potential process improvements.
  - Orthogonal defect classification;
  - Root cause analysis technique;
  - Statistical process control;
  - The personal software process.
  
- **Benchmarking techniques**
  - Identify an ‘excellent’ organization in a field;
  - Document its practices and tools;
  - Assess the maturity of an organization or the capability of its processes.

#### **6.8.2 Principle (#4): “Implement a disciplined approach and improve it continuously”**

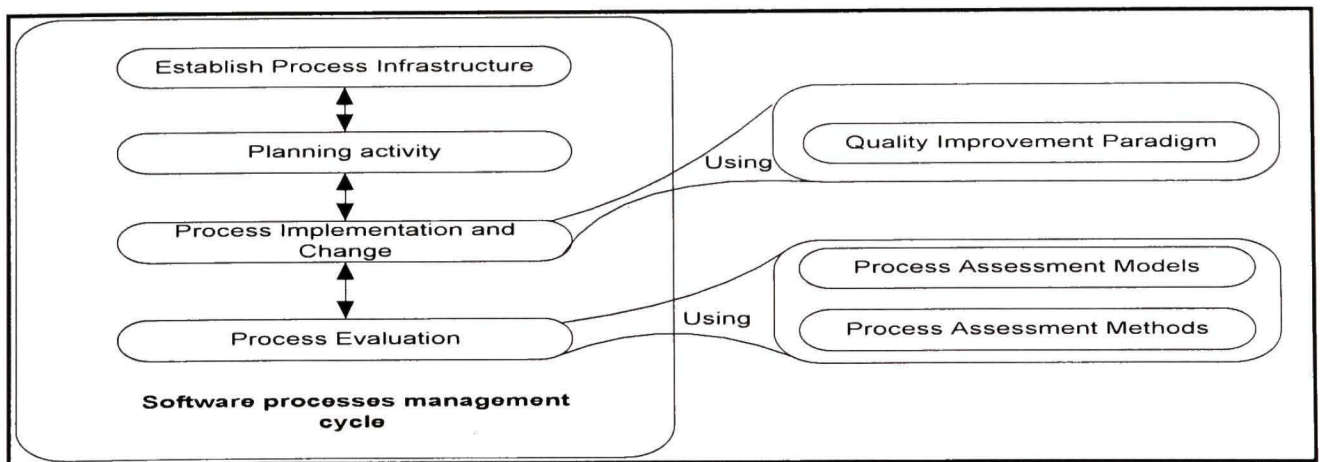
**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software engineering process” subarea:

- Process implementation and change.

#### **B. Operational guidelines documented in this KA for this FP**

Software process management cycle is composed of the four activities. Figure 6.22 describes the different software process management activities. The activities are defined as follows:

- **The establish process infrastructure activity**
  - Establish commitment to process implementation and change (including obtaining management buy-in);
  - Put in place an appropriate infrastructure resources (competent staff, tools, and funding);
  - Assign responsibilities.
- **The planning activity**
  - Understand the current business objectives and process needs of the individual, project, or organization;
  - Identify its strengths and weaknesses;
  - Make a plan for process implementation and change.
- **Process implementation and change**
  - Execute the plan;
  - Deploy new processes (which may involve, for example, the deployment of tools and training of staff);
  - Change existing processes.
- **Process evaluation**
  - Evaluate the process implementation change;
  - Use the results as input for subsequent cycles.



**Figure 6.22: Software process management cycle- operational view in the process KA**

**6.8.3 Principle (#6):** “Quality is the top priority; long term productivity is a natural consequence of high quality”.

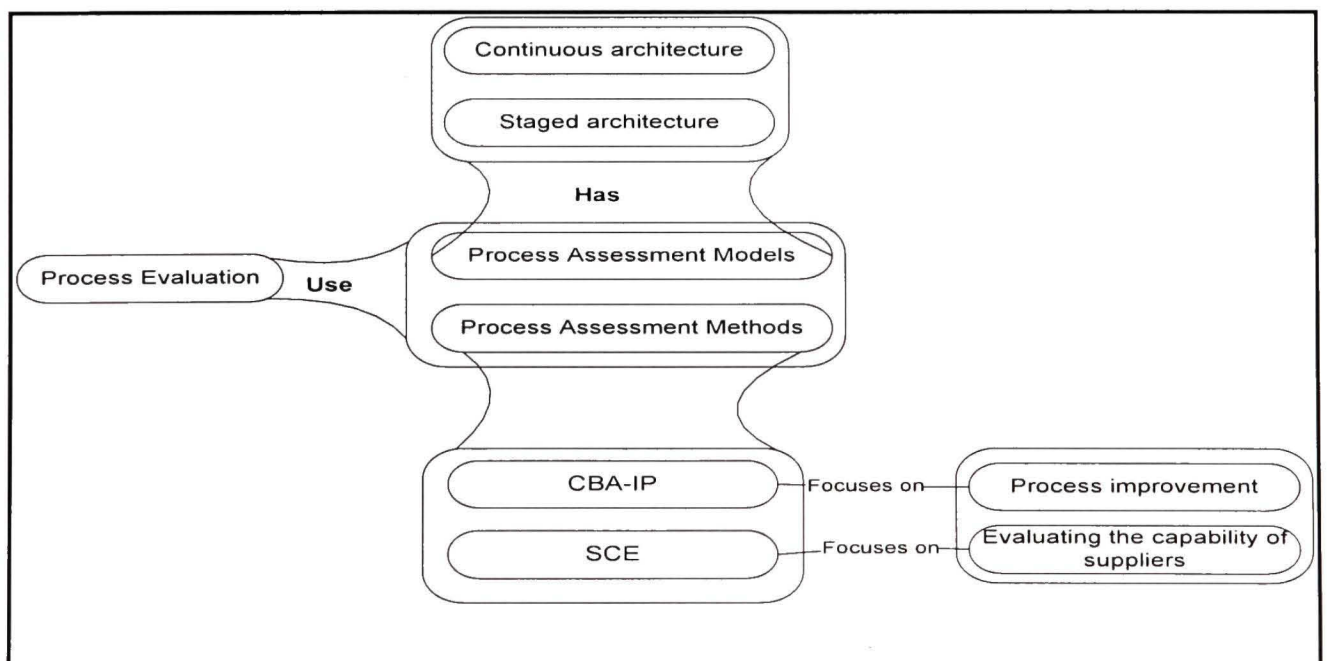
**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software process” topics:

- Process assessment models;
- Process assessment methods.

**B. Operational guidelines documented in this KA for this FP**

Process assessment makes use of models and methods:

- The organization should evaluate which architecture to choose for an assessment: continuous model or staged architectures depending on needs.
- To assess a process, choose an assessment method. For instance, there is the CPA-IPI method used for process improvement, the SCE methods used for the capability of suppliers. Figure 6.23 presents the model related to the process assessment models and methods.

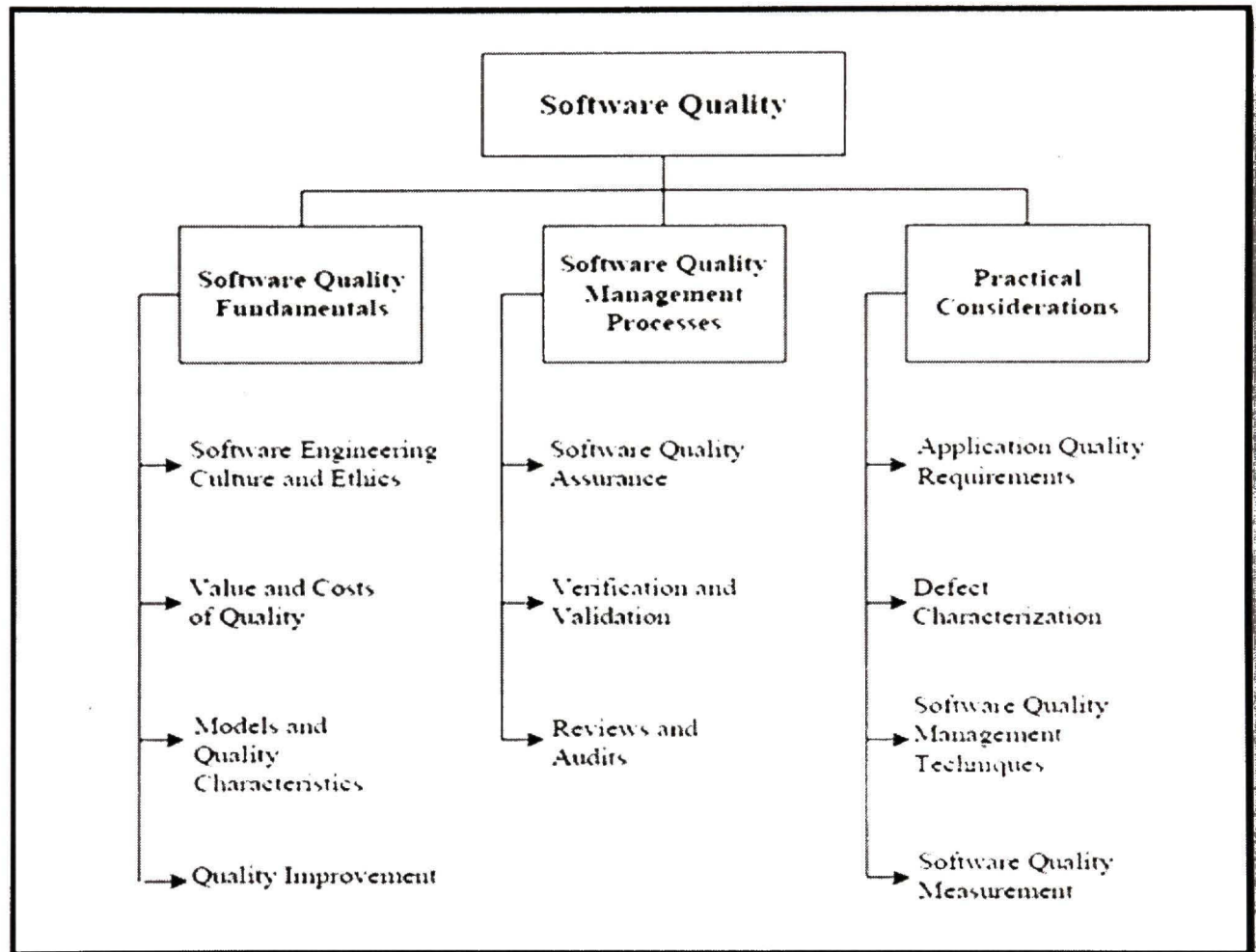


**Figure 6.23: Process assessment models and methods - operational view in the Process KA**



## 6.10 Software quality– description of an operational perspective

The “Software quality” KA is composed of three subareas see Figure 6.24.



**Figure 6.24: SWEBOK guide: “Software quality” knowledge area (ISO-TR-19759, 2004)**

**6.10.1 Principle (#1):** “Apply and use quantitative measurements in decision making”

**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software quality” topic:

- Software quality measurement.

**B. Operational guidelines documented in this KA for this FP**

Some guidelines to follow for “Software quality measurement” such as:

- **Assistance when to stop testing using**
  - Reliability models;
  - Benchmarks.
- **Use of generic models for cost of SQM processes based on**
  - When a defect is found;
  - How much effort it takes to fix a defect.
- **Use of mathematical and graphical techniques to help interpret the measures**
  - Statistically based;
  - Statistical tests;
  - Trend analysis;
  - Prediction.
- **References for measurement methods**
  - Measure defect occurrences with defect analysis;
  - Apply statistical methods to understand the types of defects that are frequent;
  - Measure the test coverage to estimate how much test effort to be done and to predict possible remaining defects.

**6.9.2 Principle (#4): “Implement a disciplined approach and improve it continuously”**

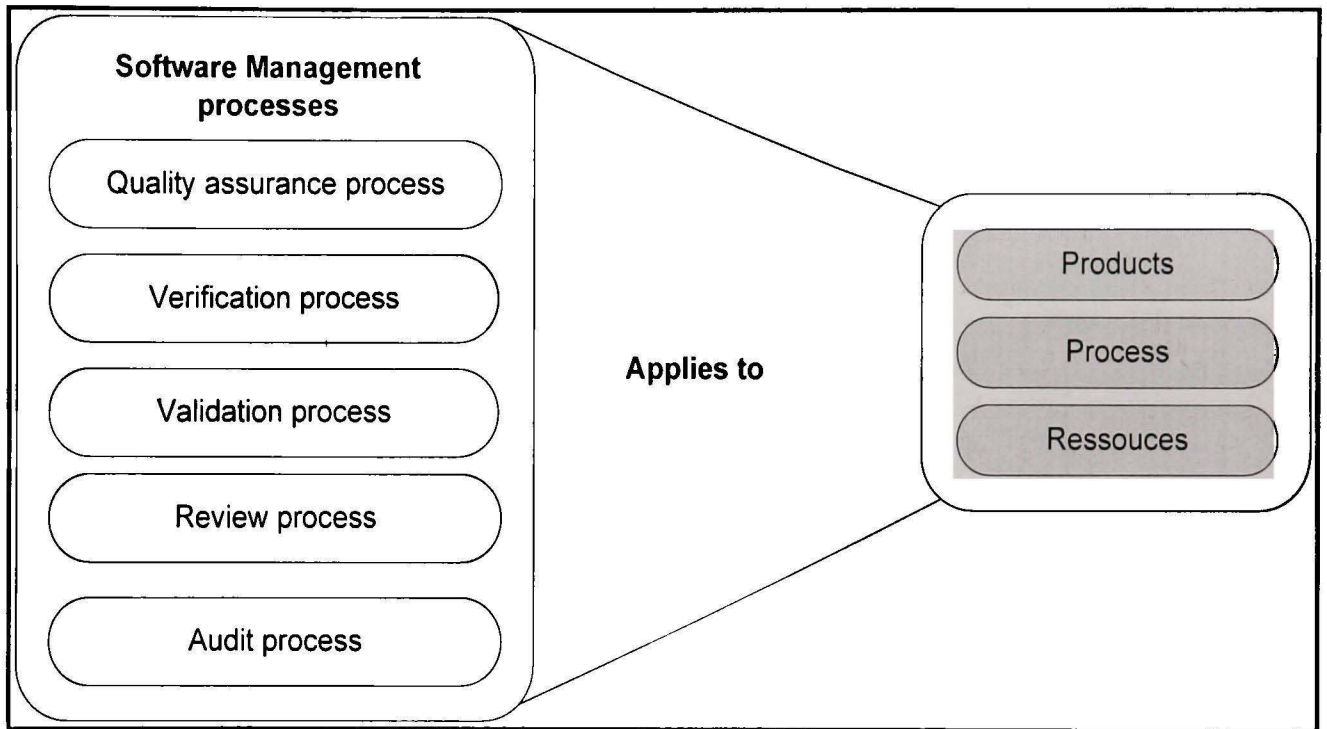
**A. Presence of this FP in the taxonomy of this KA:** this FP is applied within the following “Software quality” topic and subarea:

- Quality improvement (topic).
- Software quality management processes (sub-area).

**B. Operational guidelines documented in this KA for this FP**

**Quality improvement:** Total Quality management (TQM) process of Plan, Do, Check, and Act (PDCA) are approaches used to improve quality.

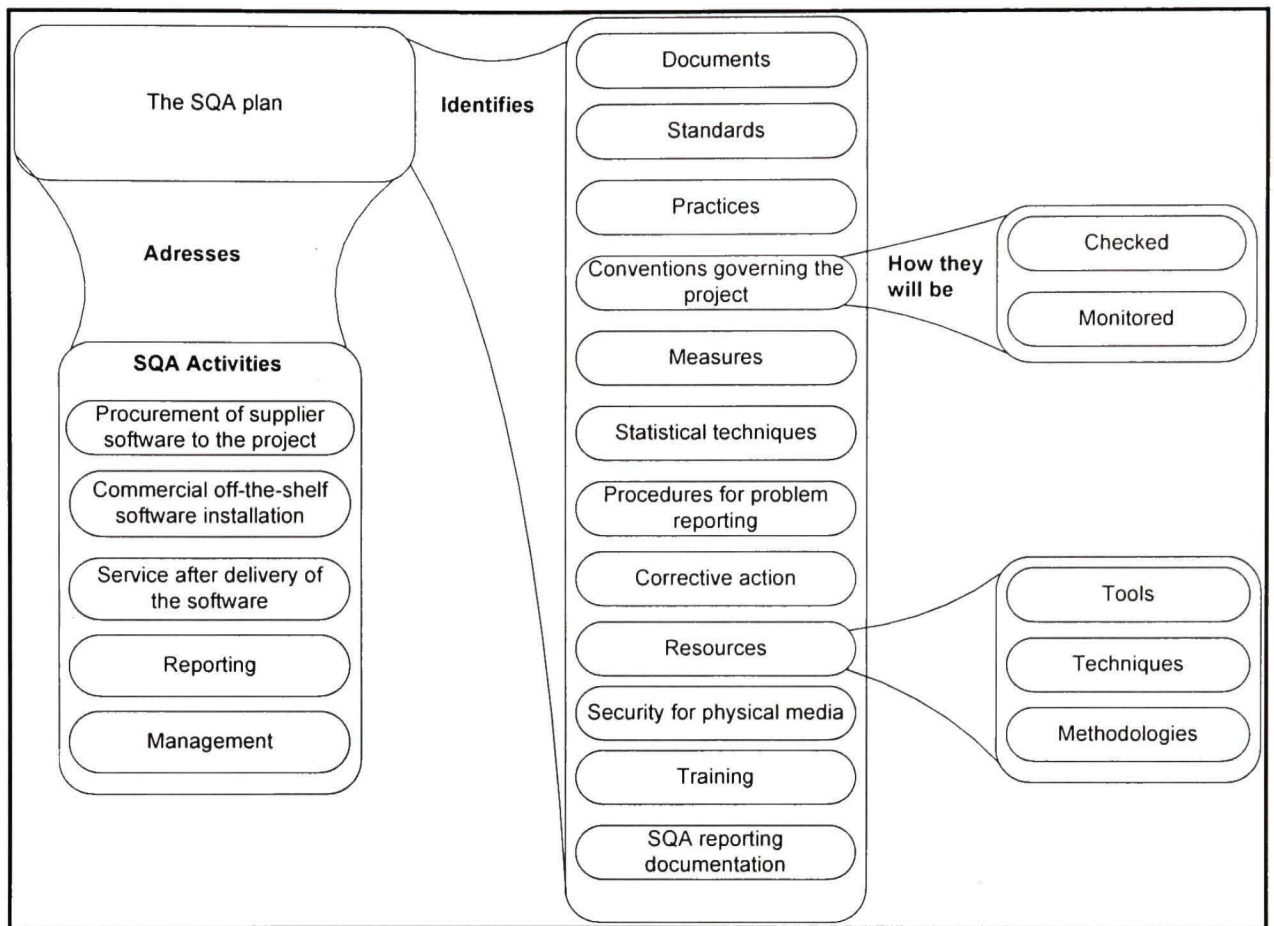
**Software quality management processes:** The software quality management processes apply to products, processes and resources. Figure 6.25 presents a model related to the different software management processes.



**Figure 6.25: Software management processes**

- **Software quality assurance:** in the software quality assurance plan the following steps should be carried out defined:
  - Define the quality target;
  - Define the specific activities with their cost and resource requirements;
  - Define management objectives for the activities and their schedule in relation to those objectives.

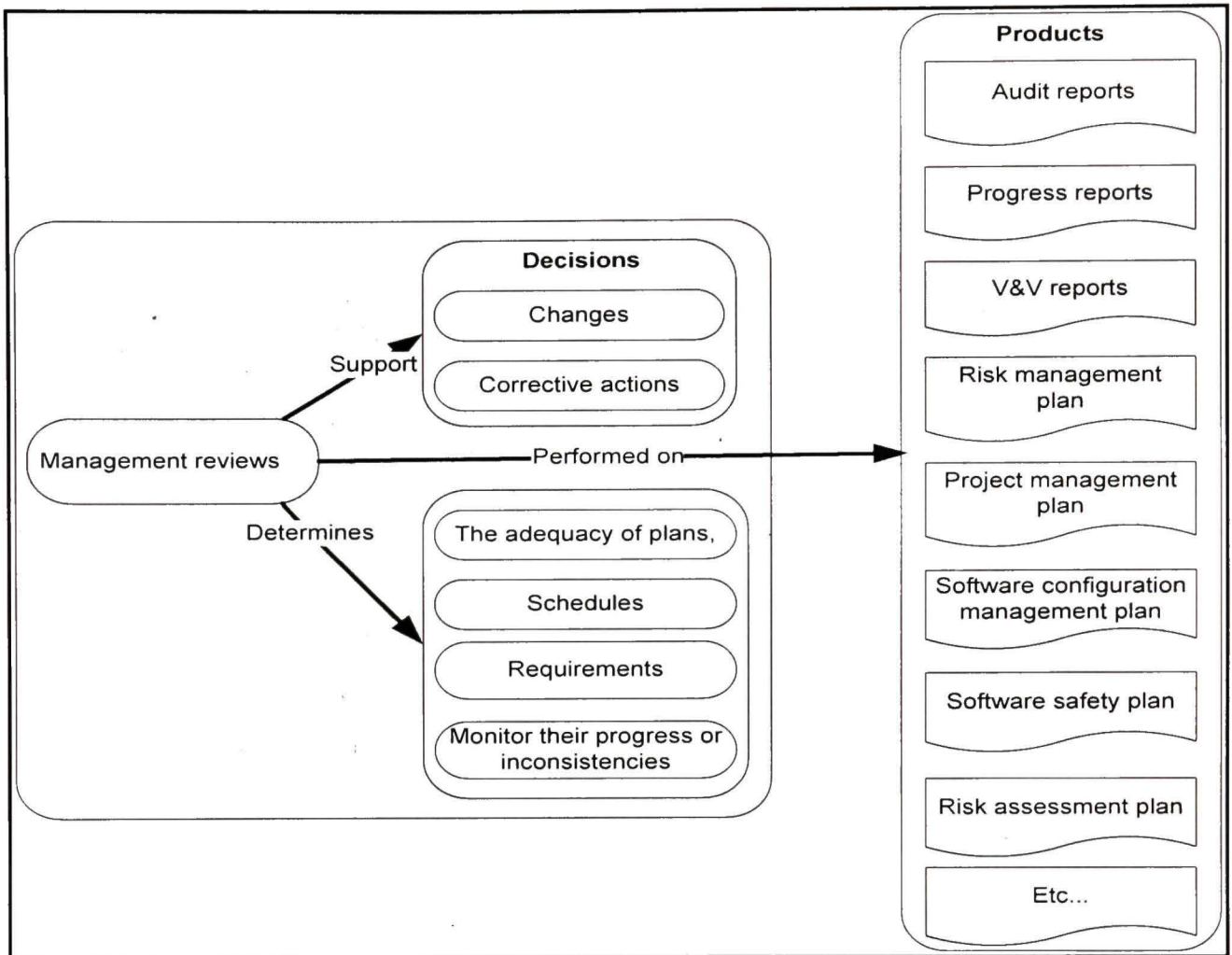
Figure 6.26 presents model related to “Software quality assurance”. The software quality assurance plan identifies documents, standards, practices and addresses the following activities such as: service after delivery to the software.



**Figure 6.26: Software quality assurance**

- **Verification and validation:** Specify the planning and the execution of the verification and the validation activities.
- **Reviews and audits:** five types of reviews are defined in (ISO-TR-19759, 2004) such as: management reviews, technical reviews, inspections, walkthrough and audits.

**Management reviews:** The management reviews support decisions and are performed on many reports such as: audit report, progress report and plans such as risk management plan, and project management. Management reviews establish the adequacy of plans, schedules, requirements and monitor the progress of inconsistencies. More details are presented in Figure 6.27.

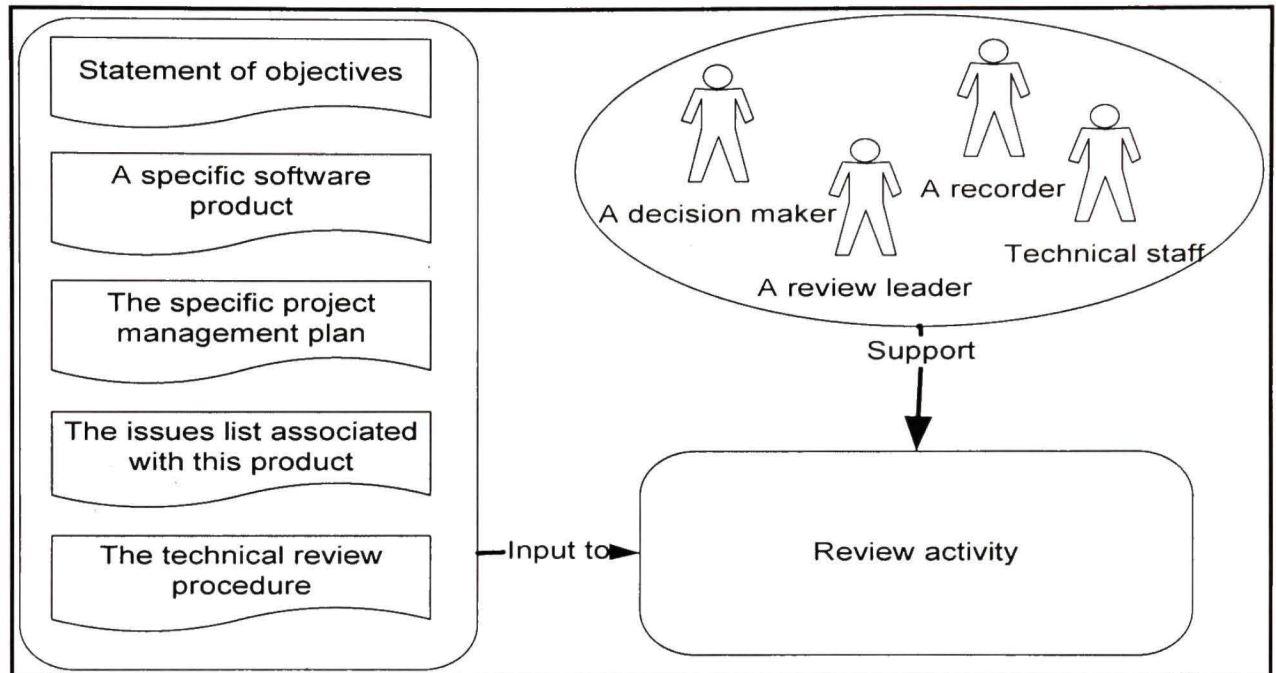


**Figure 6.27: Management reviews- operational view in the quality KA**

**Technical reviews:** to set up a technical review in process the following two requirements are necessary:

- The definition of specific roles which includes: a decision maker, a review leader, a recorder, and technical staff;
- The mandatory input which includes: statement of objectives, a specific software product, the specific software management plan, the issues list associated with the product and the technical review procedure.

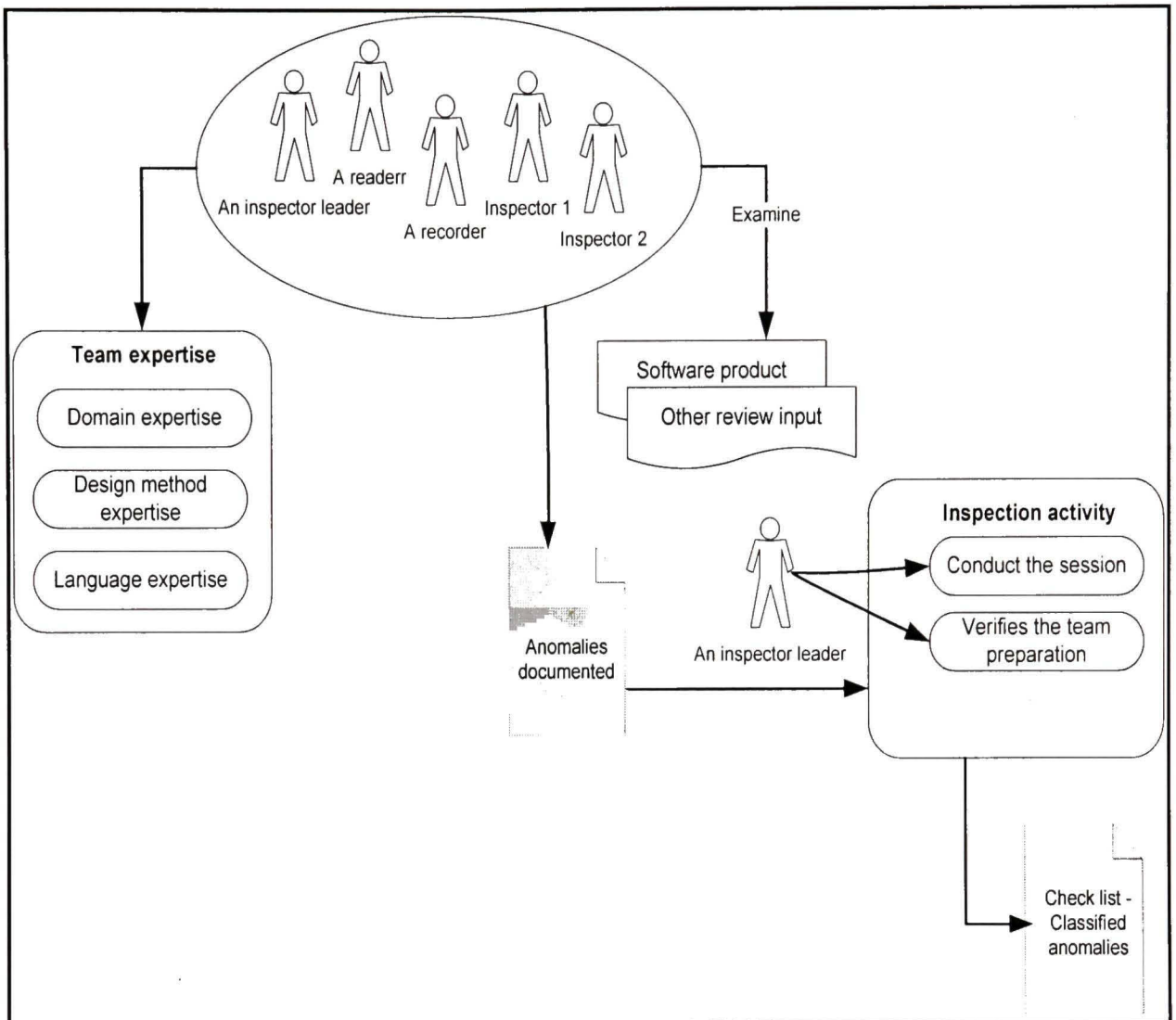
Figure 6.28 presents the model related to “Technical reviews”.



**Figure 6.28: Technical reviews - operational view in the quality KA**

**Inspections:** an inspection is conducted with the participation of the author of an intermediate or a final product, a leader, a recorder, a reader and 2 to 5 inspectors. These team members may have different expertise such as domain expertise, design method expertise and language expertise.

The inspection leader will receive the lists of anomalies prior to the inspection meeting. This list of anomalies is produced by examining the software product by every team member. The inspection leader conducts the inspection and verifies the team preparation during the inspection. As a result, a list is produced that categorizes anomalies. Figure 6.29 presents the model related to “Inspections”.



**Figure 6.29: Inspections - operational view in the quality KA**

### Walkthroughs and Audit

- A software engineer conducts walkthroughs. It is less formal than an inspection;
- The audit identifies the instances of nonconformance as a result a report is produced.

The audit activity is conducted formally with collaboration of a team that includes a leader auditor, another auditor, a recorder, an initiator and a representative of the audit organization.

## **6.11 Summary**

This chapter has presented the operational perspective for each FP present in the KA on the basis of the content of the SWEBOK Guide (ISO-TR-19759, 2004) for the following KAs: Requirements, Design, Construction, Testing, Maintenance, Configuration Management, Engineering Process and Quality.

Annex V includes details related to the operational guidelines aligned with the IEEE Std 1362-1998 (Concept of Operations (ConOps) Document). These detailed operational guidelines are described for the main knowledge areas of the SWEBOK Guide.

These operational guidelines are composed of five elements based on the (IEEE STD 1362-1998) standard. These suggested elements define the operational guidelines for any software.



## CHAPTER 7

### DEVELOPMENT OF A CONSOLIDATED SWEBOK VIEW FOR THE MEASUREMENT FP

#### 7.1 Introduction

In the previous chapter operational guidelines for each FP were provided for the engineering fundamental principles based on the content of the SWEBOK Guide: However, such operational guidelines were dispersed, unevenly across all KA, making comprehension and consolidation difficult.

The goal of this chapter is to present a consolidated view about the measurement FP “FP no. 1 - Apply and use quantitative measurements in decision making” within the KAs of the SWEBOK Guide (ISO-TR-19759, 2004) .

This chapter presents phase 6 and is organized as follows: section 7.2 presents the coverage of the measurement FP in the SWEBOK Guide. Section 7.3 presents a consolidated view of measurement FP. Section 7.4 presents a consolidated view model of the measurement FP. Section 7.5 presents a measurement process. Section 7.6 presents a summary.

#### 7.2 Coverage of the measurement principle in the KAs of the SWEBOK guide

Table 7.1 describes the presence of the measurement principle in the knowledge areas of the SWEBOK guide (ISO-TR-19759, 2004). This table is divided into three columns: each of these columns lists the SWEBOK knowledge areas, subareas and topics.

One notes from Table 7.1 that the measurement FP is present in all the knowledge areas of the SWEBOK guide. For instance, the measurement FP is described in one topic each for “Software design”, “Software construction”, “Software configuration management” and

“Software quality”; and is described in more than two topics for “Software engineering process”, “Software engineering management” and “Software testing”. The knowledge area the most covered by this measurement FP is “Software engineering process”.

Table 7.1 The measurement FP in the SWEBOK guide KA

<b>SWEBOK Knowledge areas</b>	<b>SWEBOK subareas</b>	<b>SWEBOK topics</b>
<b>Software requirements</b>	Software Requirements fundamentals	Quantifiable requirements
	Practical considerations	Measuring requirements
<b>Software design</b>	Software design quality analysis and evaluation	Measures
<b>Software construction</b>	Managing construction	Construction measurement
<b>Software testing</b>	Test related Measures	Evaluation of the program under test Evaluation of the tests performed
	Test process	Practical considerations
<b>Software maintenance</b>	Key issues in software maintenance	Maintenance cost estimation
		Software maintenance measurement
<b>Software configuration management</b>	Management of the scm process	Surveillance of software configuration management
<b>Software engineering process</b>	Product and process measurement	Software process measurement
		Software product measurement
		Quality of measurement results
		Software information models
<b>Software quality</b>	Practical considerations	Software quality measurement

### 7.3 Consolidated view of measurement FP

This section presents a consolidated view for the measurement FP in the SWEBOK Guide (ISO-TR-19759, 2004) and includes the consolidated view for the measurement FP for the engineering processes, management process and for the secondary processes:

- The engineering processes refers to “Software requirements”, “Software design”, “Software construction”, “Software testing” and “Software maintenance” KAs.
- The management process refers to “Software engineering management” KA.
- The secondary processes refer to “Software configuration management”, “Software engineering process” and “Software quality” KAs.

Table 7.2 illustrates from right to left the different SWEBOK KAs, the measurements topics related to each KA and the consolidated view.

Table 7.2 Consolidated view of the measurement FP

SWEBOK knowledge areas	SWEBOK measurement topics	Consolidated view for SWEBOK guide
Software requirements	- Quantifiable requirements -Measuring requirements	A- Engineering processes
Software design	- Design measures	
Construction measurement	-Construction measurement	
Software testing	-Evaluation of the program under test -Evaluation of the tests performed -Practical considerations	
Software maintenance	-Maintenance cost estimation -Software maintenance measurement	
Software engineering management	-Establish and sustain measurement commitment -Plan the measurement process -Perform the measurement process - Evaluate measurement	

Table 7.2 Consolidated view of the measurement FP(continued)

SWEBOK knowledge areas	SWEBOK measurement topics	Consolidated view for SWEBOK guide
Software configuration management	-Surveillance of software configuration management	C- Secondary processes
Software engineering process	- Software process measurement	
	- Software product measurement	
	- Quality of measurement results	
	- Software information models	
	- Measurement techniques	
Software quality	-Software quality measurement	

#### 7.4 Consolidated view model of the measurement FP

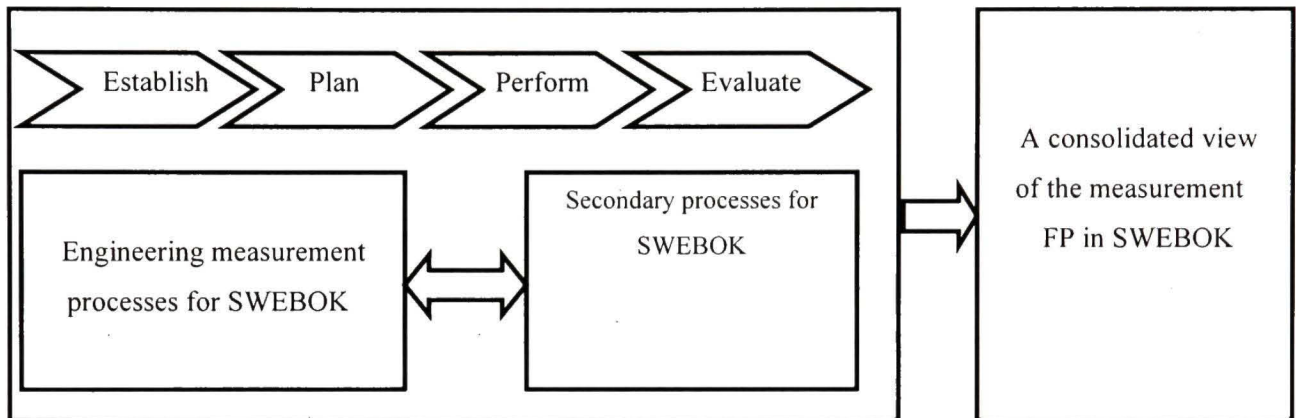
In the SWEBOK Guide (ISO-TR-19759, 2004), the “Software Engineering Management” KA describes the measurement process. In this knowledge area the measurement process is composed of the four activities that are: “Establish and sustain measurement commitment”, “Plan the measurement process”, “Perform the measurement process” and “Evaluate the measurement process”. This measurement process will allow the integration of all measurement topics and subareas contained in the SWEBOK Guide. Figure 7.1 presents the model for the measurement FP consolidated view. This model contains the engineering processes, the secondary processes for the SWEBOK Guide.

#### 7.5 Measurement process:

This section presents the different activities based on the measurement process as follows:

##### 7.5.1 Establish and sustain measurement commitment activity

This section presents the topics or subareas related to the operational guidelines already described in chapter 6 for the measurement FP for “establish and sustain the measurement commitment” measurement process. These measurements topics or subareas are composed from the engineering process and the secondary processes as follows: “Software information models” and “Software quality measurements”.



**Figure 7.1: Model of a consolidated SWEBOK view of the measurement FP**

### 7.5.2 Plan the measurement process

This section presents the topics or subareas related to the operational guidelines already described in chapter 6 for the measurement FP related to “plan the measurement process”. These measurements topics or subareas are composed of the engineering process and the secondary processes as follows: “Measuring requirements”, “Design measures”, “Construction Measurements”, “Evaluation of the program under test”, “Maintenance cost estimation”, “Software maintenance measurement”, “Software product measurement”, “Software information models” and “Software quality measurement”.

### 7.5.3 Perform the measurement process

This section presents the topics or subareas related to the operational guidelines already described in chapter 6 for the measurement FP related to “perform the measurement process”. These measurements topics or subareas are composed of the engineering process

and the secondary processes as follows: “Design measures”, “Evaluation of the tests performed”, “Cost/effort estimation and other process measures” (Practical consideration in testing), “Software maintenance measurement”, “Software product measurement”, “Quality of measurement results” and “Process measurement techniques”.

#### **7.5.4 Evaluate the measurement process**

This section presents the topics or subareas related to the operational guidelines already described in chapter 6 for the measurement FP related to “Evaluating the measurement process”. These measurements topics or subareas are composed of the engineering process and the secondary processes as follows: “Quantifiable requirements”, “Design measures”, “Evaluation of the program under test”, “Practical considerations in testing”, “Maintenance cost estimation”, “Software process measurement”, “Quality of measurements results”, “Process measurement techniques” and “Software quality measurement”.

### **7.6 Summary**

In this chapter a consolidated view was given for the measurement FP “Apply and use quantitative measurements in decision making”.

This chapter illustrated first the coverage of the measurement FP in the ten knowledge areas of the SWEBOK Guide. For instance, the chapter most covered by the measurement FP is the “Software engineering process” and among the least covered chapters by the measurement FP are: “Software design” and “Software construction”.

This chapter provided a consolidated view for the measurement FP based on the software engineering management, the engineering processes and the secondary processes for the SWEBOK Guide.

In addition, a model for the consolidated view was provided for the measurement FP. This model allows for the integration of all the measurements topics and subareas that are contained in the SWEBOK Guide (ISO-TR-19759, 2004) based on the management process.

## CHAPTER 8

### ANALYSIS OF A SWEBOK KA FROM AN ENGINEERING PERSPECTIVE WITH RESPECT TO THE ENGINEERING FUNDAMENTAL PRINCIPLES

#### 8.1 Introduction

This chapter undertakes the identification of Vincenti's classification of the six categories of engineering knowledge in the "Software requirements" KA with respect to the presence of the set of fundamental principles.

In the literature survey, no work exists to map the principles and the categories of engineering knowledge with the SWEBOK KA (ISO-TR-19759 2004). This mapping will allow the identification of the missing engineering knowledge with respect to each FP.

Furthermore, the analysis undertaken in this chapter is based on chapters 3, 4 and 5 of this research study:

- Chapter 3 presented the mapping between the six categories of engineering knowledge defined by Vincenti and the "Software requirements" knowledge area, Annex I;
- Chapter 4 analysed the fundamental principles from an engineering perspective;
- Chapter 5 documented the coverage of the nine fundamental principles in the SWEBOK guide.

This chapter presents phase 7 and is organized as follows: Section 2 introduces the identification of the engineering concepts in the "Software requirements" knowledge area with respect to the set of the engineering fundamental principles. Section 3 presents Vincenti's categories and FP in the Requirements KA. Section 4 presents the mapping results from Vincenti's viewpoint. Section 5 presents the mapping results from the fundamental principles viewpoint. A summary is presented in section 6. Annex VI presents the mapping



between the Vincenti, the nine FP and the “Software requirements”, “Software design” and “Software construction” KAs.

## **8.2 Identification of the engineering concepts in the “Software requirements” KA with respect to the FP and Vincenti.**

### **8.2.1 Mapping 1: Vincenti categories of engineering knowledge and “Software requirements” KA**

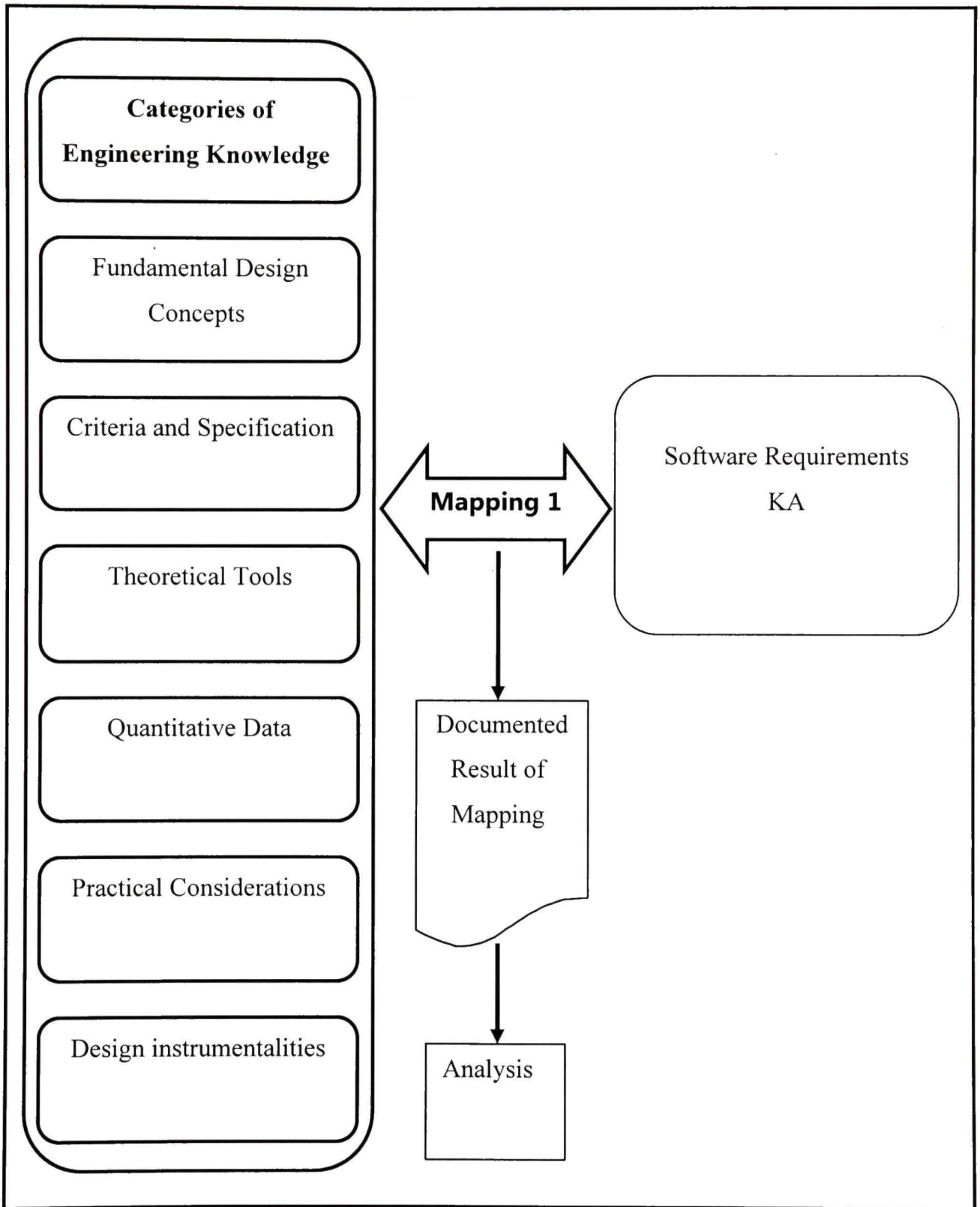
As mentioned in chapter 3 of this thesis, the “Software requirements” KA was analyzed from an engineering perspective using the six categories of engineering knowledge (Vincenti W. G. 1990). The output of this analysis is the identification of the engineering knowledge addressed within the “Software requirements” KA – see Figure 8.1.

Mapping 1 consists of the analysis between the categories of engineering knowledge and the “Software requirements” KA done in chapter 3 of this thesis. The results of this mapping are presented in terms of what is present and what is missing in the “Software requirements” KA from an engineering perspective.

### **8.2.2 Mapping 2: The list of FP to each of the SWEBOK KA (see Chapter 5)**

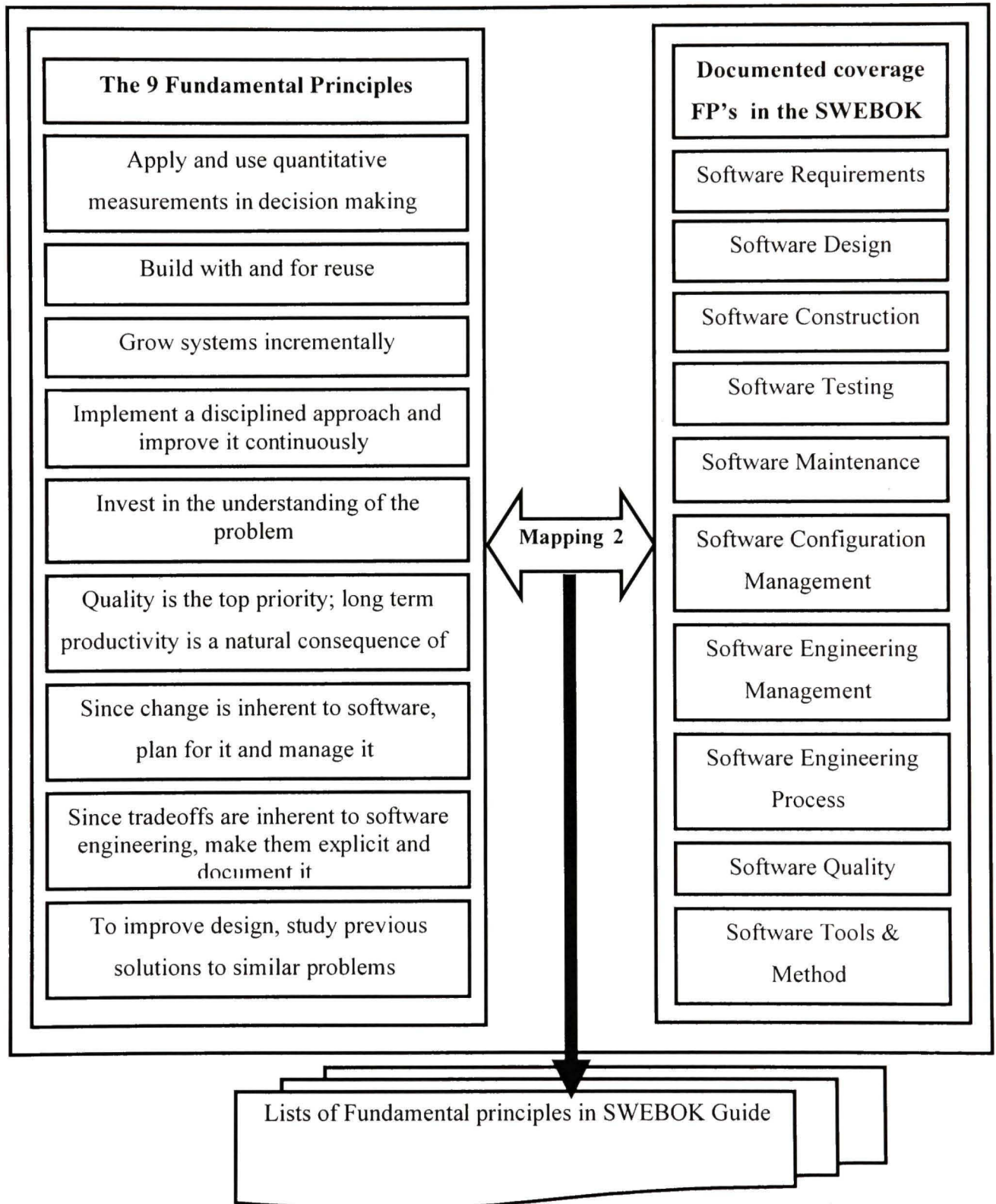
The analysis of the fundamental principles from an engineering perspective was done in chapter 4; the output of chapter 4 is the list of the nine engineering fundamental principles.

Meanwhile, chapter 5 documented the coverage of the nine fundamental principles in the SWEBOK guide. This documentation was done through mapping the list of FP to each of the SWEBOK KA – see Figure 8.2.



**Figure 8.1: Mapping of the Vincent's engineering knowledge to the SWEBOK Guide**

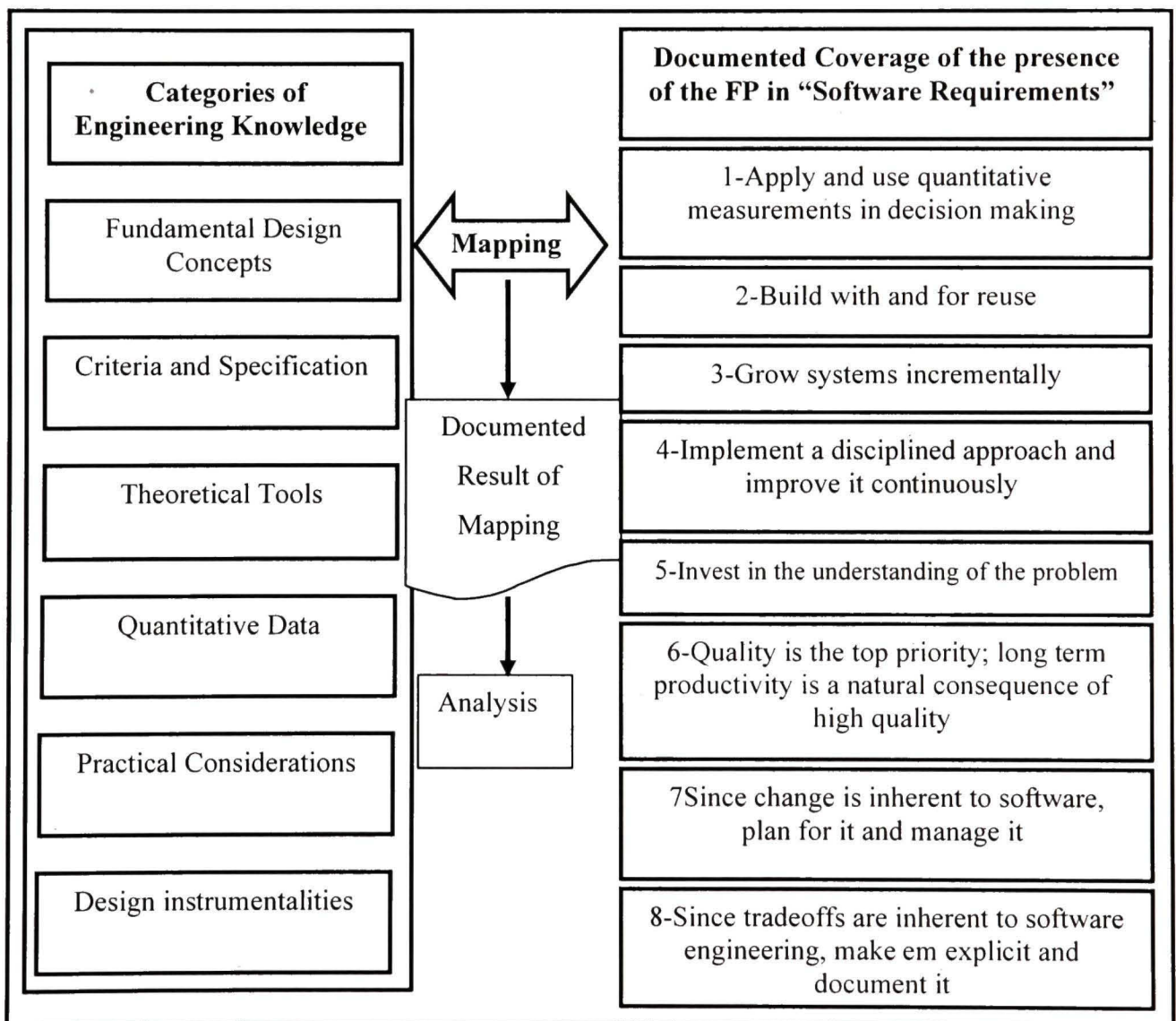
The output of chapter 5 is the list of the FP present within each KA of the SWEBOK Guide.



**Figure 8.2: Mapping the set of the FP to the SWEBOK Guide - see chapter 5**

### 8.2.3 Mapping 3: Vincenti's categories of engineering knowledge the "Software requirements" with respect the FP

Chapter 8 presents now a mapping between the set of the categories of engineering knowledge (Vincenti W. G. 1990) into the "Software requirements" KA with respect to the set of engineering fundamental principles. This mapping allows the investigation of the maturity of the "Software requirements" KA from an engineering perspective - see Figure 8.3.



**Figure 8.3: Mapping of the categories of engineering knowledge to the set of FP in "Software requirements" KA**

### 8.3 Vincenti's categories and FP in the requirements KA

This example illustrates the mapping between the following: the Vincenti's categories of engineering knowledge (Vincenti W. G. 1990), the list of the presence of the FP in the "Software requirements" KA. The results through this example will illustrate the missing categories of engineering knowledge with respect to each FP that is present in the Requirements KA.

This mapping is based on the different models of the six categories of engineering knowledge described in Vincenti and provides a description of what is present completely and what is missing as engineering knowledge with respect to each of the engineering fundamental principles.

Furthermore, the details of the mapping are presented in Annex VI; the table describes the mapping of all fundamental principles covered within the "Software requirements" KA with respect to the six categories of engineering knowledge.

The results of this mapping process between the Vincenti's categories of engineering knowledge and the list of the presence of FP in the "Software requirement" is summarized in Table 8.1 where the fundamental principles are presented in rows while the categories of engineering knowledge are presented in columns.

Table 8.1 Mapping results of the Vincenti's categories of engineering knowledge to the FP in the "Software requirements" KA.

Engineering Fundamental Principles	Vincenti's Categories of Engineering Knowledge					
	Fundamental Design oncepts	Criteria And Specification	Theoretical Tools	Quantitative Data	Practical Considerations	Design Instrumentalities
#1 Apply and use quantitative measurements in decision making					X	
# 2 Build with and for reuse	X		X			X
# 3 Grow systems incrementally	X	X	X		X	X
# 4 Implement a disciplined approach and improve it continuously	X	X	X		X	X
#5 Invest in the understanding of the problem	X	X	X		X	
# 6 Quality is the top priority; long term productivity is a natural consequence of high quality		X				X
#7 Since change is inherent to software, plan for it and manage it		X			X	X
#8 Since tradeoffs are inherent to software engineering, make them explicit and document it					X	
#9 To improve design, study previous solutions to similar problems						

#### 8.4 Mapping results from Vincenti's viewpoint

The results of this mapping between the Vincenti's categories of engineering knowledge (Vincenti W. G. 1990) and the engineering fundamental principles are described as follows

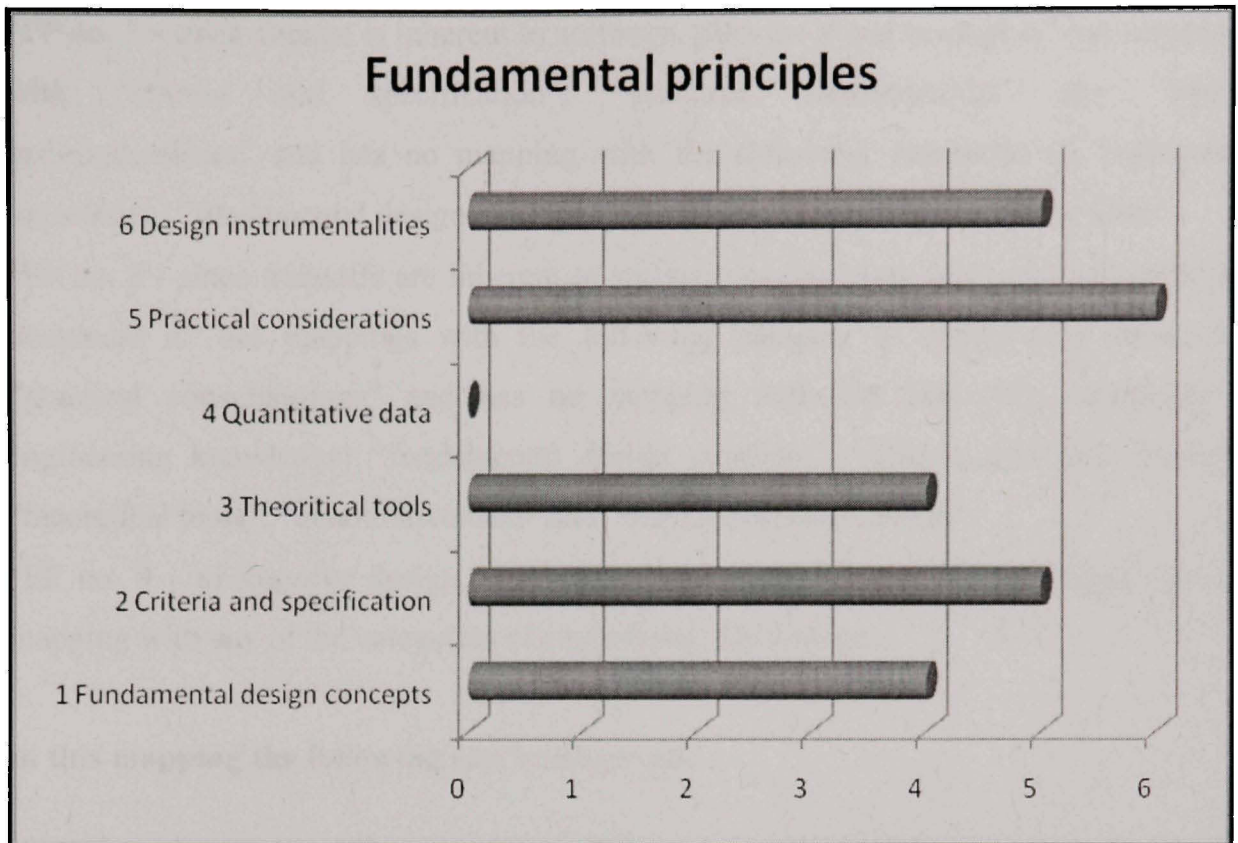
from Vincenti's viewpoint see Figure 8.4: Vincenti's six categories and the FP frequencies in the Requirements KA.

- The “fundamental design concepts” has mapping with fundamental principles 2 to 5, whereas the other fundamental principles have no mapping to this engineering knowledge category.
- The “criteria and specification” has mappings with fundamental principles 3 to 7, whereas the other fundamental principles have no mapping to “criteria and specification”.
- The “theoretical tools” has mappings with fundamental principles 2 to 5, whereas the other fundamental principles have no such mapping.
- The “practical considerations” has mappings with fundamental principles 1, 3, 4, 5, 7 and 8, whereas the other fundamental principles have no such mapping.
- The “design instrumentalities” has mappings with fundamental principles 2, 3, 4, 6 and 7, whereas the other fundamental principles have no such mapping.

### **8.5 Mapping results from the fundamental principles viewpoint**

Figure 8.5 summarizes the results of the mapping between the Vincenti's categories of engineering knowledge and the engineering fundamental principles from the FP viewpoint. Following is a description of these results:

- “FP no. 1 - Apply and use quantitative measurements in decision making” has mappings with the following category: “Practical considerations”; and has no mapping with the following categories: “fundamental design concepts”, “criteria and specification”, “quantitative data”, “theoretical tools” and “design instrumentalities”.
- “FP no. 2 - build with and for reuse” has mappings with the following categories of engineering knowledge: “fundamental design concepts”, “theoretical tools” and “design instrumentalities” and has no mapping with the following categories of engineering knowledge: “criteria and specification”, “quantitative data” and “practical considerations”.



**Figure 8.4: Vincenti's six categories and the FP frequencies in the Requirements KA.**

- “FP no. 3 - grow systems incrementally” and “fp no. 4 - implement a disciplined approach and improve it continuously” have mappings with “fundamental design concepts”, “criteria and specification”, “theoretical tools”, “practical considerations” and “design instrumentalities” and has no mapping with “quantitative data”.
- “FP no. 5 - invest in the understanding of the problem” has mappings with “fundamental design concepts”, “criteria and specification”, “theoretical tools”, and “practical considerations”; it has no mapping with the following categories of engineering knowledge: “quantitative data” and “design instrumentalities”.
- “FP no. 6 - quality is the top priority; long term productivity is a natural consequence of high quality” has mappings with the following categories, “criteria and specification” and “design instrumentalities”; it has no mapping with the following categories of engineering knowledge: “fundamental design concepts”, “quantitative data”, “theoretical tools”, and “practical considerations”.

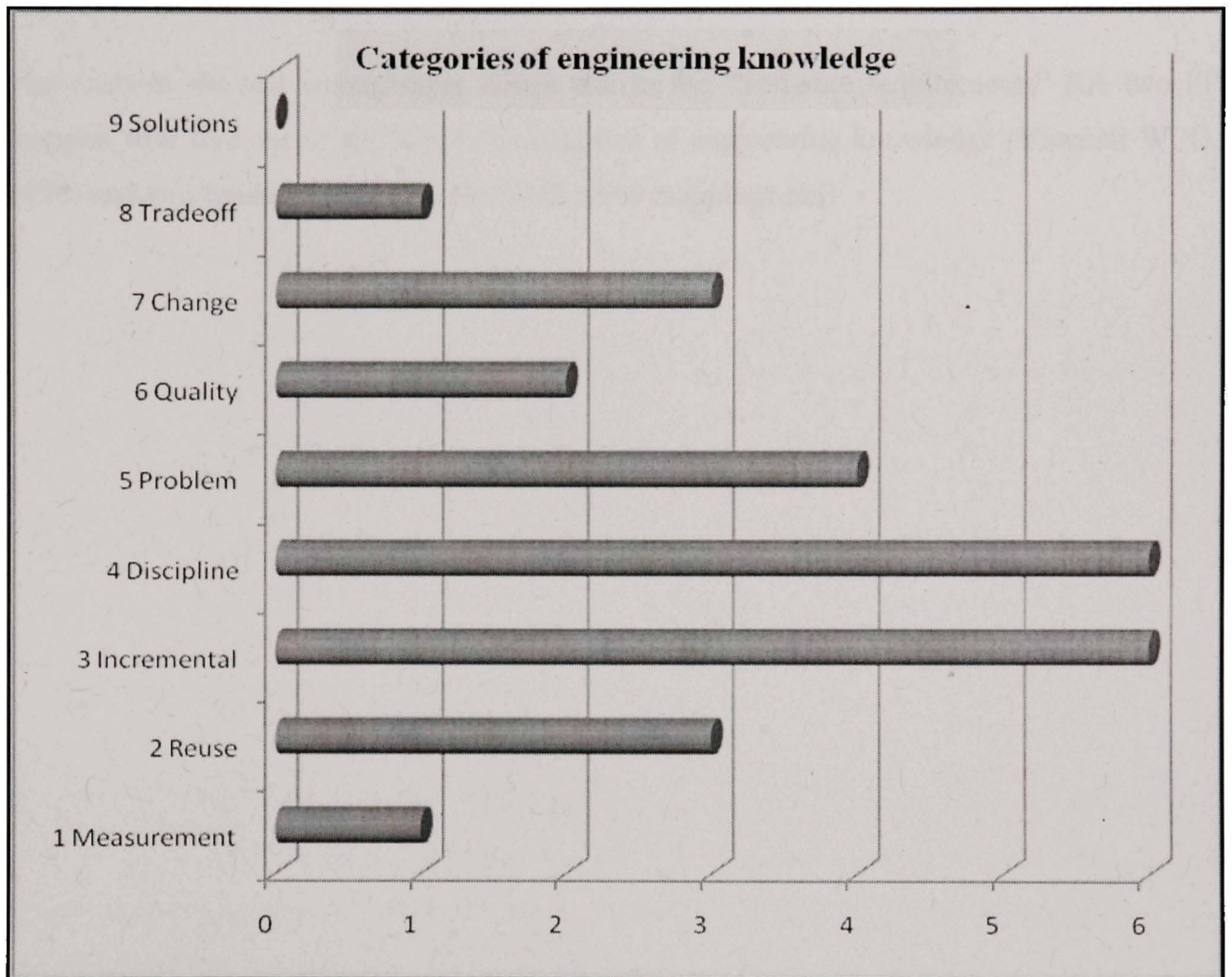


- “FP no. 7 - since change is inherent to software, plan for it and manage it” has mappings with “criteria and specification”, “practical considerations” and “design instrumentalities” and has no mapping with the following categories of engineering knowledge: “fundamental design concepts”, “theoretical tools”, “quantitative data”.
- “FP no. 8 - since tradeoffs are inherent to software engineering, make them explicit and document it” has mappings with the following category of engineering knowledge “practical considerations” and has no mapping with the following categories of engineering knowledge: “fundamental design concepts”, “criteria and specification”, “theoretical tools”, “quantitative data” and “design instrumentalities”.
- “FP no. 9 - to improve design, study previous solutions to similar problems” has no mapping with any of the categories of engineering knowledge.

**From this mapping the following can be observed:**

- The principle “FP no. 3 - Grow systems incrementally” and “FP no. 4 - Implement a disciplined approach and improve it continuously” has almost a full coverage of Vincenti categories of engineering knowledge within the requirements KA. This means that these two principles mapped five engineering categories and contain considerable engineering knowledge.
- “FP no. 2 -Build with and for reuse”, “FP no. 5 - Invest in the understanding of the problem”, “FP no. 6 - Quality is the top priority; long term productivity is a natural consequence of high quality” and “FP no. 7 - Since change is inherent to software, plan for it and manage it” partially cover the categories of engineering knowledge. This means there are gaps of engineering knowledge with regards to these FPs within the requirements KA.
- “FP no. 1 - Apply and use quantitative measurements in decision making” covers “practical considerations” category of engineering knowledge while “FP no. 8 - Since tradeoffs are inherent to software engineering, make them explicit and document it” covers only the “practical considerations” category of engineering knowledge. This

shows that the coverage of engineering knowledge is very weak for these 2 FPs within the requirements KA.



**Figure 8.5: Frequency of fundamental principles for “Software requirements” KA**

## 8.6 Summary

The work presented here has involved the mapping between the “Software requirements” KA and the Vincenti categories of engineering knowledge (Vincenti W. G. 1990) with respect to the presence of each of the engineering fundamental principles. The results of this mapping were presented from both the Vincenti’s viewpoint and from the fundamental principles viewpoint.

The analysis of the mapping results exposed the status of maturity of the “Software requirements” knowledge area from an engineering perspective. It actually showed the lacking of engineering knowledge with regards to each of the fundamental principles.

The analysis showed among other things that in the “Software requirements” KA two FP mapped with five out of six Vincenti categories of engineering knowledge (Vincenti W. G. 1990) and two fundamental principles with a few mappings only.

## CHAPTER 9

### DEVELOPING AN EVALUATION METHOD TO VERIFY THE OPERATIONAL GUIDELINES IN THE SWEBOK GUIDE

#### 9.1 Introduction

Chapter 6 proposed the operational guidelines for the SWEBOK Guide along with IEEE STD 1362-1998. Chapter 7 proposed the consolidated view for the measurement FP as can be noticed, the proposed operational guidelines for the measurement FP are not completely covered.

In this chapter, phase 8, the detailed development process for the evaluation method is presented. This evaluation method will be used to evaluate the operational guidelines for the measurement FP. This will allow the evaluation of the operational guidelines coverage for this FP.

The evaluation of the operational guidelines will be undertaken for the measurement FP for each of the SWEBOK knowledge areas where it has been described. This evaluation will allow the verification of the coverage of the operational guidelines related to the measurement FP.

This chapter is organized as follows: section 9.2 presents the operational guidelines evaluation method and a summary is presented in section 9.3

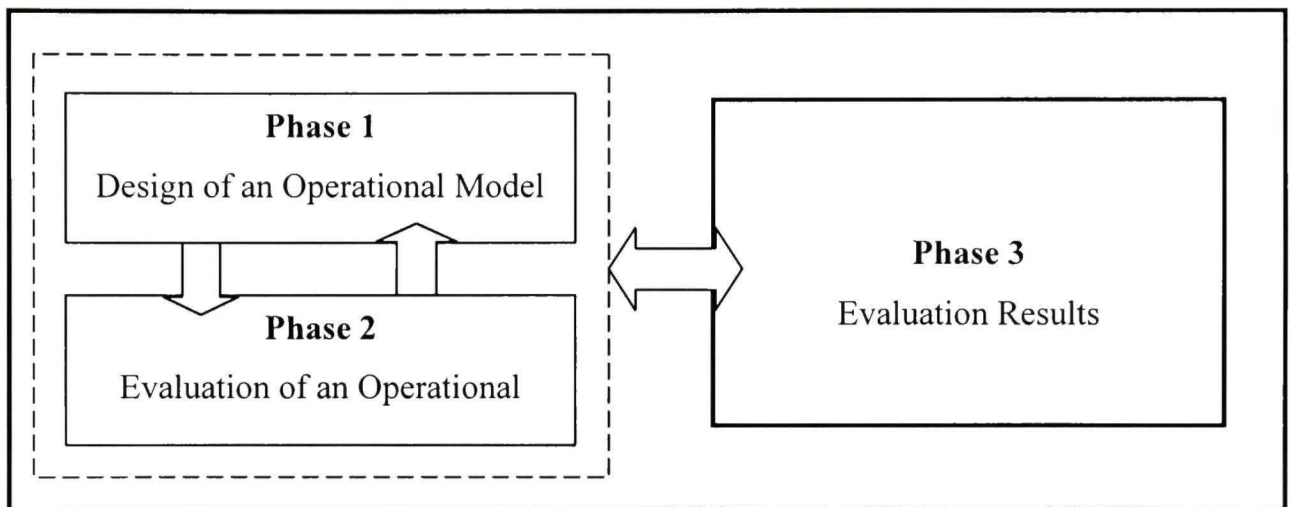
#### 9.2 Evaluation method for the operational guidelines

The evaluation method for the operational guidelines is composed of three phases:

- The first phase is the design of an operational model;
- The second phase is the evaluation procedure;

- The third phase evaluates the results of the operational guidelines after implementation through the first two phases.

Figure 9.1 illustrates the phases of the evaluation procedure - see figure 9.1.



**Figure 9.1: The three phases of the evaluation procedure of operational guidelines**

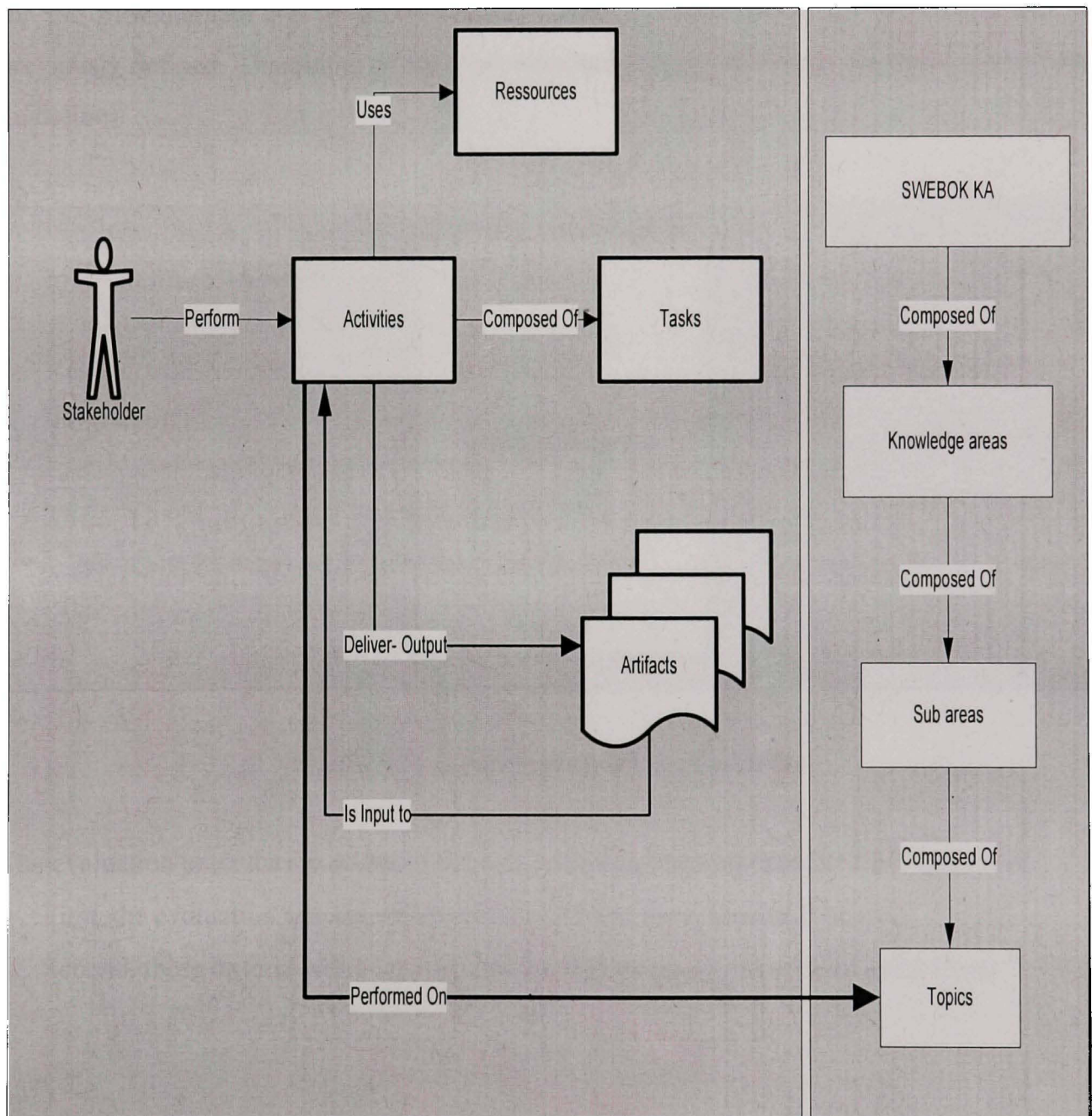
### 9.2.1 Phase 1: Design of an operational model of operational guidelines

The operational model for the application of the FP is illustrated in

Figure 9.2. This model is composed of the following six elements: activities, steps, resources, artifacts-input, artifact-output and stakeholders. Moreover the activities use resources (like techniques and methods), are composed of tasks, deliver artifact output and are performed by a stakeholder. The definitions of each of these elements are provided in (ISO-12207 1995 ) as follows:

- **Activities:** In software engineering SE activities are strategically oriented and add real value. SE activities are equally important during creation, maintenance and end-of-life phases of an IT solution as well as “A set of cohesive tasks of a process”.
- **Task:** “Requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more outcomes of a process”.

- **Resources:** “Asset that is utilized or consumed during the execution of a process”.
- **Stakeholder** “Individual or organization having a right, share, claim or interest in a system or in its possession of characteristics that meet their needs and expectations”.
- **Artifacts-input:** Refer to activity and produced by performing the tasks.
- **Artifacts-output:** Refer to activity and produced by performing the tasks.

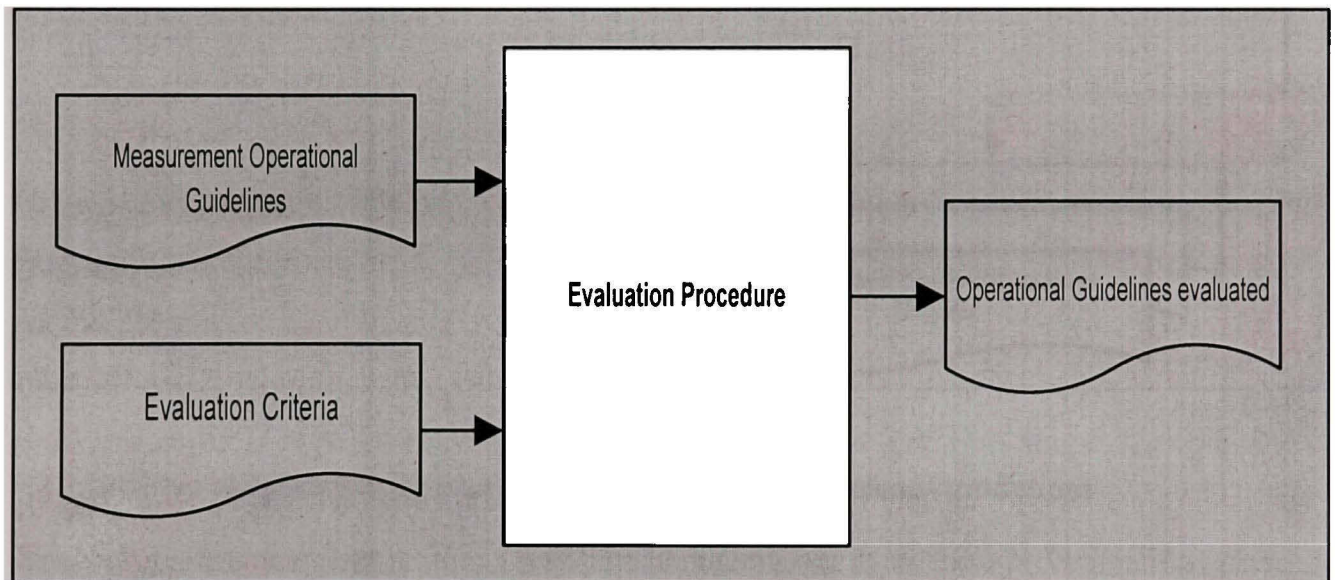


**Figure 9.2: Operational model of operational guidelines**

These six elements of the operational model will be taken as criteria in the evaluation procedure that is defined next.

### 9.2.2 Phase 2: Conduct the evaluation procedure

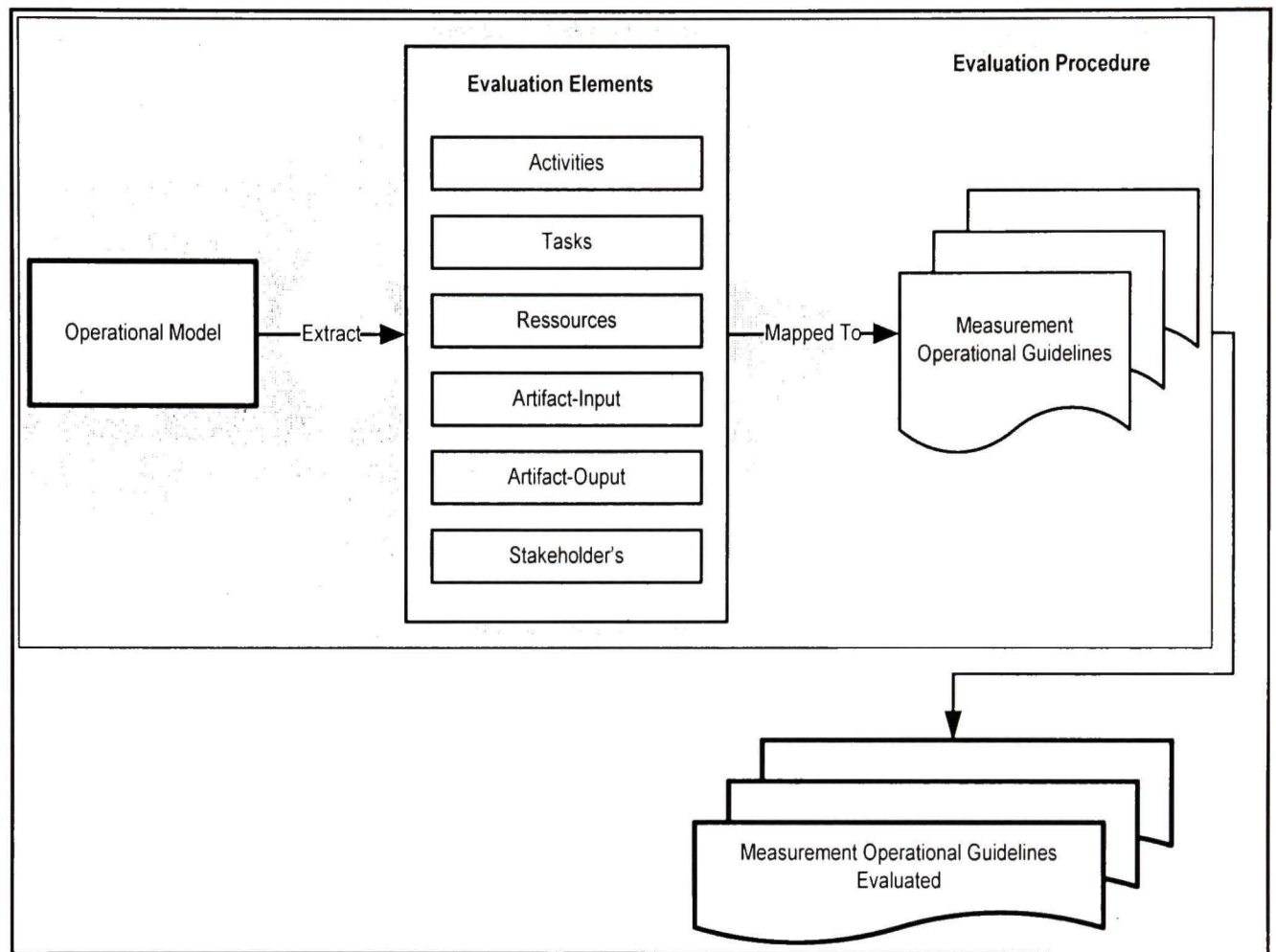
Figure 9.3 describes the evaluation procedure and takes as input the operational guidelines for the measurement FP in the SWEBOK Guide as well as the six evaluation criteria previously defined. The output of the evaluation procedure will be the evaluated operational guidelines.



**Figure 9.3: Generic evaluation procedure**

The evaluation procedure consists of the two following steps as described in Figure 9.4:

- First, the evaluation criteria will be extracted from the operational model;
- Second, these criteria will be mapped to the measurement operational guidelines.



**Figure 9.4: Evaluation procedure of operational guidelines**

### 9.2.3 Phase 3: Evaluation results

The result of the evaluation of the operational guidelines for the measurement FP within the SWEBOK Guide is presented in Table 9.1.

This table is organized as follows: the ten SWEBOK knowledge areas are presented in the rows while the six evaluation criteria are presented in the columns (activity, steps, resources, artifacts-input, artifact-output and stakeholder).



The results of this evaluation are classified into categories ranging from high coverage (the knowledge areas that have a good mapping result compared to the operational model) to low coverage.

These categories are composed of the following:

- Category-A includes the knowledge areas that cover more than 80% of the six evaluation criteria;
- Category-B includes the knowledge areas that cover over 50%;
- Category-C includes the knowledge areas that cover over 30%;
- Category-D includes the knowledge areas that cover over 16 %;
- Category-E includes the knowledge areas with zero coverage.

Some examples for these categories are presented as follows:

**Category-A:** coverage of more than 80% of the six evaluation criteria

“Software engineering management” and “Software process” KA cover respectively six and five out of six criteria.

**Category-B:** 50%- coverage

‘Software requirements’, ‘Software testing’, ‘Software maintenance’ and ‘Software quality’ cover three out of six criteria. As an example, the coverage for two of the KA is described as follows:

- ‘Software requirements’ covers ‘resources’ with size measurement method for functional requirements; with software specification document as ‘artifact-input’ and with functional size as ‘artifact-output’.
- ‘Software quality’ covers ‘activities’ for: stop testing, cost of SQM, and interpretation of results. It also covers ‘resources’ for techniques, methods such as (defect analysis, statistical test coverage) and generic models; and finally, using effort as ‘artifact-output’.

**Category-C: 30%- coverage**

“Software design” and “Software construction” respectively cover two out of six and one out of six criteria.

- “Software design” covers “artifact-input” by using structure chart, class diagram and List of measures as “artifact-output”.

**Category-D: 16%- coverage**

“Software construction” covers one out of six evaluation criteria.

- “Software construction” covers “artifact-output” with a list of measured artifacts such as code developed.

**Category-E: 0- coverage**

“Software configuration management” cover zero out of six criteria.

- The measurement FP in “Software configuration management” is present but not described, therefore none of the six criteria’s is being covered.

Table 9.1 Evaluation results of the measurement- FP in the SWEBOK KA

SWEBOK Chapters	Activities	Tasks	Resources	Artifacts -Input	Artifacts- Output	Stakeholders
Software requirements			✓	✓	✓	
Software design				✓	✓	
Software construction				✓		

Table 9.1 Evaluation results of the measurement- FP in the SWEBOK KA (continued)

SWEBOK Chapters	Activities	Tasks	Resources	Artifacts -Input	Artifacts-Output	Stakeholders
Software testing	✓	✓			✓	
Software maintenance			✓	✓	✓	
Software configuration management						
Software engineering management	✓	✓	✓	✓	✓	✓
Software process	✓	✓	✓	✓	✓	
Software quality	✓		✓	✓		

This evaluation results provide five categories ranging from high coverage to low coverage. This shows that the operational guidelines for the measurement FP are not completely covered and there is missing information related to the description of the operational guidelines for the requirements, design, construction, maintenance and quality KAs, For instance, in most of these KAs there is a lack of activity and task descriptions. Once all the missing elements related to activities, tasks, resources etc, will have been described for each KA, a complete operational guidelines can be prepared for this FP across the full software engineering body of knowledge – of course on the basis of the revisions subsequent to 2004 version of the SWEBOK Guide.

### 9.3 Summary

This chapter illustrated the evaluation of the different operational guidelines related to the measurement FP for each of the SWEBOK knowledge areas. It was based on a three-phase

evaluation procedure. Each operational guidelines is taken as input to this evaluation procedure and is evaluated against the six evaluation criteria that were extracted from the operational model.

As a result, this evaluation exposed the missing elements related to the description of operational guidelines for the measurement FP so that steps can be taken to further complete those elements.

## CONCLUSION

The research work presented in this thesis had two main research objectives:

- The identification of the engineering fundamental principles of software engineering from the 34 candidates identified by (Séguin N. 2006),
- The description of the operational guidelines of these engineering fundamental principles on the basis of the content of the SWEBOK Guide.

In this research study, these two objectives were achieved by using Vincenti's, the SWEBOK Guide and the IEEE standard 1362-1998 Concept of Operations (ConOps) Document.

In addition, the research issues that have been addressed in this thesis are aligned with Vincenti's categories of engineering knowledge to analyze the software engineering and analyze software engineering principles from an engineering perspective.

Moreover, operational guidelines for the SWEBOK Guide knowledge areas were described in alignment with the IEEE 1362-1998. As a result of this research, several contributions have been made, as discussed in the section "Contributions of the research".

### **This research work opens avenues to:**

- Build a consensual analysis of the software engineering principles from an engineering perspective;
- Identification of the software engineering principles;
- Improvement of the operational guidelines for the SWEBOK Guide;
- Improvement of the SWEBOK Guide from an engineering perspective and from the software engineering principles perspective as well.

### **Contributions of the research**

This section summarizes the various contributions achieved throughout this research study. These contributions are classified into four categories:

### **Engineering perspective in software engineering**

The research contributions related to “Engineering perspective in software engineering” are as follows:

- The models representing the relationships between the different categories of engineering knowledge as well as the models for each of the categories of engineering knowledge that have been extracted from Vincenti: these models make these categories more understandable (Chapter 3);
- The analysis of software engineering from an engineering perspective;
- The mapping of the knowledge contained in the SWEBOK KA to Vincenti’s categories of engineering knowledge;
- The presentation of the new breakdown of the SWEBOK KA based on the categories of engineering knowledge.

### **Identification of FP**

The research contributions related to “Identification of FP” are as follows:

- Identification of 9 software engineering principles from the 34 candidate principles selected by Seguin;
- Identification of the hierarchy of the set of the other 25 candidates.

### **Description of FP**

The research contributions related to “Description of FP” are as follows:

- The identification of software engineering principles within the content of the SWEBOK Guide – ISO TR 19759;
- Each of the principles verified from an engineering perspective and then described from an operational view point based on the content of the SWEBOK Guide and structured according to the international standard IEEE 1362 1998 Concepts of Operations document (ConOps);
- Consolidated view for the measurement FP;
- Evaluation method for the operational guidelines.

### **SWEBOK: Identification of potential gaps**

The research contributions related to “SWEBOK: Identification of potential gaps” are as follows:

- Documentation of the principles using as a basis the SWEBOK KAs;
- Identification of the missing engineering FP within the content of the SWEBOK Guide;
- Identification of the missing categories of engineering knowledge within 3 SWEBOK KA.

Some of the initial outcomes of this thesis (from phase 2 and phase 3) have been published / submitted in the following journals or conferences:

- Abran, Alain; Meridji, Kenza, “Analysis of Software Engineering from An Engineering Perspective”, European Journal for the Informatics Professional ,vol. 7, No. 1, February , 2006 , pp. 46-52 . [www.upgrade-cepis.org](http://www.upgrade-cepis.org) Upgrade: ISSN 1684-5285 Novática: ISSN 0211-2124 .
- Abran, Alain; Meridji, Kenza, “Analysis de la Ingenieria del Software desde de la perspective de la Ingenieria”, in Novatica, Ed. ATI - Asociacion de Technicos de Informatica , Vol. 32 No. 179 , 2006 , pp. 7-20.
- Abran., A., Meridji, K., Dolado, J., “Software Engineering from an Engineering Perspective: SWEBOK as a Study Object”, Apoyo a la Decision en Ingenieria del Software - ADIS workshop, Congreso Espanol de Informatica - CEDI Conference, Zaragoza, Spain, Sept 11-14, 2007.
- Kenza Meridji, Alain Abran “Software Engineering Principles: Do they Meet Engineering Criteria?” (Submitted to the Journal of Systems and Software ELSEVIER).

### **Workshop presentations**

- Meridji K.; Abran, A., “Software Engineering Principles? Do they Meet Engineering Criteria”, in Workshop: Engineering Foundations of Software Engineering’, International Conference on Software Engineering Education ICEE , Coimbra, Portugal , 2007.

- Meridji K.; Abran, A., “Software requirements: Application of Fundamental Principles”, in Workshop: ‘Engineering Foundations of Software Engineering’, International Conference on Software Engineering Education ICEE, Coimbra, Portugal, 2007 .

The research outcomes from phase 4 to phase 7 still have to be prepared into a publication format and submitted for publication.

As can be noticed from the literature review in this thesis, there was not much previous work done on the principles of software engineering. This thesis took into consideration the engineering perspective for the identification from the literature of software engineering principles. This led to the description of operational procedures of these principles from an engineering view point such a study was not conducted in the literature.

To carry out the descriptions of the operational procedures, the SWEBOK Guide was a candidate for this study. A number of verifications have been done in this thesis on the SWEBOK Guide such as the following:

- Analysis of the SWEBOK Guide from an engineering perspective through comparing the design concepts in SWEBOK vs. the design concepts in engineering;
- Mapping the SWEBOK KAs to the categories of engineering knowledge;
- Mapping the 9 engineering FP to the SWEBOK Guide from an engineering perspective to verify the presence of the engineering FP in the SWEBOK Guide;
- Mapping the engineering knowledge, SWEBOK and the engineering principles this illustrates that this mapping was possible.

The SWEBOK Guide has been verified from different perspectives. This research concludes that the SWEBOK Guide maps with the engineering perspective. Also operational guidelines have been built on the basis of this SWEBOK body of knowledge. Therefore the SWEBOK Guide could be considered as an engineering body of knowledge and could be taken as a foundation for software engineering.



This conclusion does not mean however that all foundations are included, not that what is included is entirely mature.

**Research limitations:**

Some of the limitations of this research work are described as follows:

**A- Engineering perspective in software engineering**

Most of the analysis is based on an engineering perspective. Still to be addressed are all engineering types of knowledge relevant and applicable to software engineering.

**B- Identification of FP**

- A limited number of references on the fundamental principles of software engineering
- The list of 34 CFP taken as input to phase 3 is not necessarily exhaustive.
- The nine engineering FP were identified through criteria; these criteria are not necessarily exhaustive.
- The number of experts to verify the selected engineering FP was limited.
- The expertise of the participants to verify the selected engineering FP was limited.
- The research did not include experimentation of each FP identified.

**C- Description of FP**

The content of the operational description is not exhaustive and is limited to the content of the SWEBOK Guide.

**D- SWEBOK: Identification of potential gaps**

Should the missing FP within the content of the SWEBOK Guide be covered or not within each KA?

### **Further Research work**

This thesis was of an exploratory nature in the same way the methodology adopted was of an exploratory nature. Exploratory research does not give a definitive result by itself, rather it opens new research directions such as:

#### **A-Engineering perspective in software engineering**

Analysis of the content of software engineering discipline, such as measurement.

#### **B-Identification of FP**

- Identification of gaps in software engineering principles;
- Each principle can be the subject of a more in-depth research study;
- Revisions with additional experts could be conducted to further strengthen the nine engineering FP and their hierarchy;
- Identification of engineering criteria within the software engineering body of knowledge;
- Identification of engineering criteria addressed within the software engineering curriculum;
- Experimentation of these FP through successful projects to observe how the FP were applied in such projects.

#### **C-Description of FP**

Develop an international standard for operational guidelines.

#### **D- SWEBOK: Identification of potential gaps**

The engineering FP could be used to improve the SWEBOK content in the upcoming revision.

Addressing the missing categories of engineering knowledge exposed in this research study could contribute to improve from an engineering perspective the upcoming update to the content of the SWEBOK Guide.

**Research impacts**

Software engineering lacks well recognized fundamental principles and reflects its immature status of development. Defining universal recognized fundamental principles could reform our view of software engineering and software engineering education. This work could eventually have an impact at the educational level as well as the industrial level.

Nowadays the industrial sector of software development is facing a lot of problems due to many reasons. One of the reasons is that the software engineering discipline is not as mature as other engineering disciplines. Software projects run over time and over budget. A universally recognized and well documented set of fundamental principles applicable to industry could have an industrial impact on the methodologies, the standards and the tools used. For instance, the following questions can be raised:

- Do the methodologies proposed to the industry cover the fundamental principles?
- Do the software engineering standards, such as those of ISO and IEEE, cover the full set of fundamental principles?

On the other hand, the definition and establishment of universally recognized fundamental principles could have an impact on the design of software engineering curricula and could provide material at the educational level for teaching courses related to the software engineering discipline and could facilitate transmitting the appropriate knowledge and skills to students in alignment with engineering knowledge, preparing them to be professional engineers.

## ANNEX I

### CLASSIFICATION OF THE KNOWLEDGE CONTAINED IN THE SWEBOK GUIDE FOR THE 3 KA.

This Annex presents the mappings between the corresponding concepts for the classification of engineering knowledge types (Vincenti for the following 3 KAs in the SWEBOK Guide

- 1-Requirements
- 2-Design
- 3-Construction

#### 1. Software Requirements KA – SWEBOK 2004

Engineering knowledge Category	Corresponding Criteria	Software Requirements fundamentals	Requirements process	Requirements elicitation	Requirements analysis	Requirements specification	Requirements validation	Requirements considerations	Practical
<b>General design</b>		Definition of Functional & nonfunctional requirements	Process models		Architectural design and requirements allocation				
<b>Operational principles</b>		Definition of Functional & nonfunctional requirements	Process models		Architectural design and requirements allocation				

Engineering concepts in the software requirements from the SWEBOK Guide

<b>Fundamental design concepts</b>	<b>Concepts in people minds</b>	Product and process requirements
	<b>Normal configuration</b>	Process models Architectural design and requirements allocation
	<b>Normal technology design</b>	Process models Architectural design and requirements allocation

**Engineering concepts in the software requirements from the SWEBOOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software Requirements fundamentals</b>	<b>Requirements process</b>	<b>Requirements elicitation</b>	<b>Requirements analysis</b>	<b>Requirements specification</b>	<b>Requirements validation</b>	<b>Requirements</b>	<b>Practical considerations</b>
	<b>Specific requirement of an operational principle</b>	System requirements and software requirements	Process quality and improvement			-System and Software requirements specification		-System definition document	Requirement attributes
	<b>General qualitative goals</b>	Functional & Nonfunctional requirements	Process quality and improvement		Requirements classification				
	<b>Specific quantitative goals lay out in concrete technical terms</b>								
<b>Criteria and specification</b>	<b>The design problem must be "well defined"</b>	Functional & Nonfunctional requirements		Requirements classification		System and Software requirements specification			Requirement attributes

Unknown criteria or partially understood	Requirement attributes
Assignments of values to appropriate criteria	System requirements and software requirements Requirements classification System and Software requirements specification
This task takes place at the project definition level	-Functional & Nonfunctional requirements -Emergent properties -Requirements classification -System definition document

**Engineering concepts in the software requirements from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software Requirements fundamentals</b>	<b>Requirements process</b>	<b>Requirements elicitation</b>	<b>Requirements analysis</b>	<b>Requirements specification</b>	<b>Requirements validation</b>	<b>Requirements practical considerations</b>
Mathematical methods and theories for making design calculation	Intellectual concepts for thinking about design.	Precise and codifiable	Conceptual modeling	Model validation				

**Engineering concepts in the software requirements from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software Requirements fundamentals</b>	<b>Requirements process</b>	<b>Requirements elicitation</b>	<b>Requirements analysis</b>	<b>Requirements specification</b>	<b>Requirements validation</b>	<b>Requirements practical considerations</b>
---------------------------------------	-------------------------------	---	-----------------------------	---------------------------------	------------------------------	-----------------------------------	--------------------------------	--

Specify manufacturing process for production	
Display the detail for the device	
Data essential for design	
Obtained empirically	
Calculated theoretically	
Represented in tables or graphs	
Descriptive knowledge	
Prescriptive knowledge	
Precise and codifiable	

**Quantitative data**

**Engineering concepts in the software requirements from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	Software Requirements fundamentals	Requirements process	Requirements elicitation	Requirements analysis	Requirements specification	Requirements validation	Requirements Practical considerations
Theoretical tools and quantitative data are not sufficient- Designers also needs considerations derived from experience.							
They are hard to find them documented				Requirement negotiation			Iterative nature of the

**Engineering knowledge Category**

Theoretical tools and quantitative data are not sufficient- Designers also needs considerations derived from experience.

They are hard to find them documented

Requirement negotiation

Iterative nature of the

	requirement process	
<b>Practical considerations</b>	Requirements tracing	They are also derived from production and operation.
	Iterative nature of the requirement process	Knowledge difficult to define
	Iterative nature of the requirement process	Its defies codification
	Requirements tracing	The practical consideration coming from operation is judgment.
	Measuring requirement	Rules of thumb.

**Engineering concepts in the software requirements from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software Requirements fundamentals</b>
Knowing how	Process support and management	Requirements process elicitation
<b>Design Instrumentalities</b>	Procedural knowledge	Requirements analysis
	Change management	Requirements specification
	Change management	Requirements validation
	Change management	-Requirement reviews -Acceptance test -Prototyping
	Change management	-Requirement reviews -Acceptance test



<b>Way of thinking</b>	Process actors	Prototyping	Change management
<b>Judgmental skills</b>	Process actors	-Acceptance test -Prototyping	Change management

**2. Software design KA – SWEBOK 2004**

**Engineering concepts in the software design from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Software design fundamentals</b>	<b>Key issues in software design</b>	<b>Software structure and architecture</b>	<b>Software design quality analysis and evaluation</b>	<b>Software design notations</b>	<b>Software design strategies and methods</b>
<b>Corresponding Criteria</b>	<ul style="list-style-type: none"> <li>-General design concepts</li> <li>-The context of software design</li> <li>-The software design process</li> <li>-Enabling techniques</li> </ul>	<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling and fault tolerance</li> <li>-Interaction</li> <li>-presentation</li> <li>- data persistence</li> </ul>	<ul style="list-style-type: none"> <li>-Architectural structure and viewpoint</li> <li>-Families of programs and frameworks</li> </ul>			
<b>General design</b>						
<b>Operational principles</b>	<ul style="list-style-type: none"> <li>-The context of software design</li> <li>-Enabling techniques</li> </ul>					
<b>Concepts in people minds</b>	<ul style="list-style-type: none"> <li>-General design concepts</li> <li>-The context of software design</li> </ul>		-Architectural structure and viewpoint			

**Fundamental design concepts**

<p><b>Normal configuration</b></p>	<p>-The context of software design -The software design process</p>	<p>Architectural style (Macro architectural patterns)</p>
<p><b>Normal technology design</b></p>	<p>-The context of software design</p>	
<p><b>Engineering concepts in the software design from the SWEBOK Guide</b></p>		
<p><b>Engineering knowledge Category</b></p>	<p><b>Corresponding Criteria</b></p>	<p>Software design fundamentals      Software design quality analysis and evaluation      Software design notations      Software design strategies and methods</p>
<p>Specific requirement of an operational principle</p>	<p>Key issues in software design</p>	<p>Quality attributes</p>
<p>General qualitative goals</p>		
<p>Specific quantitative goals lay out in concrete technical terms</p>		<p>Measures</p>
<p>The design problem must be "well defined"</p>	<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling and fault tolerance</li> <li>-Interaction</li> <li>-presentation</li> <li>- data persistence</li> </ul>	<p>Quality attributes</p>
<p>Unknown criteria or partially understood</p>		<p>Quality attributes</p>
<p>Assignments of values to appropriate criteria</p>		<p>Quality attributes</p>

	<p>-General strategies -Function-oriented structured design -Object oriented design -Data structure centered design -Component based design</p>	<p>Quality attributes</p>	
--	---	---------------------------	--

**Engineering concepts in the software design from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Software design fundamentals</b>	<b>Key issues in software design</b>	<b>Software structure and architecture</b>	<b>Software design quality analysis and evaluation</b>	<b>Software design strategies and methods</b>
<p><b>Corresponding Criteria</b></p> <p>Mathematical methods and theories for making design calculation</p>		<ul style="list-style-type: none"> <li>-Concurrency</li> <li>-Control and handling of events</li> <li>-Distribution of components</li> <li>-Error and exception handling and fault tolerance</li> <li>-Interaction</li> <li>-presentation</li> <li>- data persistence</li> </ul>		<ul style="list-style-type: none"> <li>-Measures</li> <li>-Quality analysis and evaluations techniques</li> </ul>	
<p><b>Theoretical tools</b></p>				<p>Measures</p>	<ul style="list-style-type: none"> <li>-structural descriptions (static view)</li> <li>- behavior description (dynamic view)</li> </ul>
				<p>Precise and codifiable</p>	

**Engineering concepts in the software design from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software design fundamentals</b>	<b>Key issues in software design</b>	<b>Software structure and architecture</b>	<b>Software design quality analysis and evaluation</b>	<b>Software design notations</b>	<b>Software design strategies and methods</b>
<b>Quantitative data</b>	Specify manufacturing process for production						
	Display the detail for the device						
	Data essential for design						
	Obtained empirically				Measures		
	Calculated theoretically				Measures		
	Represented in tables or graphs						
<b>Engineering knowledge Category</b>	Descriptive knowledge				Measures		
	Prescriptive knowledge						
	Precise and codifiable				Measures		

**Engineering concepts in the software design from the SWEBOOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Software design fundamentals</b>	<b>Key issues in software design</b>	<b>Software structure and architecture</b>	<b>Software design quality analysis and evaluation</b>	<b>Software design notations</b>	<b>Software design strategies and methods</b>
<b>Engineering knowledge Category</b>	Theoretical tools and quantitative data are not sufficient- Designers also needs			Design Patterns (Micro architectural patterns)			

considerations derived from experience.	
They are hard to find them documented	Design Patterns (Micro architectural patterns)
They are also derived from production and operation.	
Knowledge difficult to define	
Its defies codification	
The practical consideration coming from operation is judgment.	Quality analysis and evaluations techniques
Rules of thumb.	-General strategies -Function-oriented structured design -Object oriented design -Data structure centered design -Component based design

**Engineering concepts in the software design from the SWEBOOK Guide**

Engineering knowledge Category	Software design fundamentals	Key issues in software design	Software structure and architecture	Software design quality analysis and evaluation	Software design notations	Software design strategies and methods
Knowing how			Architectural style (Macro)		-structural descriptions	

	architectural patterns)	(static view) - behavior description (dynamic view) -structural descriptions (static view) - behavior description (dynamic view)
<b>Procedural knowledge</b>		
<b>Design Instrumentalities</b>		-General strategies -Function-oriented structured design -Object oriented design -Data structure centered design -Component based design
<b>Way of thinking</b>		
<b>Judgmental skills</b>		Quality analysis and evaluations techniques

**3. Software Construction KA- SWEBOK 2004**

**Engineering concepts in the software construction from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	<b>Managing construction</b>	<b>Practical consideration</b>
	Software construction fundamentals	Managing construction	
<b>General design</b>	-Minimizing complexity -Standards in construction	Construction planning	

<b>Operational principles</b>	
Concepts in people minds	Construction planning
Integration	
<b>Fundamental design concepts</b>	
Normal configuration	Integration
Normal technology design	-Construction design
Construction models	

**Engineering concepts in the software construction from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Software construction fundamentals</b>	<b>Managing construction</b>	<b>Practical consideration</b>
<b>Corresponding Criteria</b>			
Specific requirement of an operational principle			Construction design
General qualitative goals			
Specific quantitative goals lay out in concrete technical terms			
The design problem must be "well defined"		Standards in construction	
Unknown criteria or partially understood		Anticipating change	

<b>Assignments of values to appropriate criteria</b>	Anticipating change
<b>This task takes place at the project definition level</b>	

**Engineering concepts in the software construction from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Software construction fundamentals</b>	<b>Managing construction</b>	<b>Practical consideration</b>
<b>Corresponding Criteria</b>			
Mathematical methods and theories for making design calculation	Construction measurement		
Intellectual concepts for thinking about design.	Standards in construction	Construction models	-Construction design -Reuse
Precise and codifiable		Construction planning	

**Engineering concepts in the software construction from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Software construction fundamentals</b>	<b>Managing construction</b>	<b>Practical consideration</b>
<b>Corresponding Criteria</b>			
Specify manufacturing process for production			
Display the detail for			



	the device
	Data essential for design
	Obtained empirically
<b>Quantitative data</b>	Calculated theoretically
	Represented in tables or graphs
	Descriptive knowledge
	Prescriptive knowledge
	Precise and codifiable

**Engineering concepts in the software construction from the SWEBOK Guide**

<b>Engineering knowledge Category</b>	<b>Corresponding Criteria</b>	Software construction fundamentals	Managing construction	Practical consideration
	Theoretical tools and quantitative data are not sufficient- Designers also needs considerations derived from experience. They are hard to find them documented They are also derived from production and operation. Knowledge difficult to define			
<b>Practical considerations</b>				-Construction quality

<p><b>Its defies codification</b></p>	<p>Constructing for verification</p>	<p>- Construction testing</p>
<p>The practical consideration coming from operation is judgment.</p>		
<p>Rules of thumb.</p>		

**Engineering concepts in the software construction from the SWEBOK Guide**

<p><b>Engineering knowledge Category</b></p>	<p><b>Corresponding Criteria</b></p>	<p><b>Software construction fundamentals</b></p>	<p><b>Managing construction</b></p>	<p><b>Practical consideration</b></p>
<p>Knowing how</p>	<p>Constructing for verification</p>	<p>Construction planning</p>		<p>-Construction languages -Coding - Construction testing - Construction quality</p>
<p>Procedural knowledge</p>	<p>Standards in construction</p>	<p>Construction models</p>		
<p>Way of thinking</p>				
<p>Judgmental skills</p>				<p>-Construction quality</p>

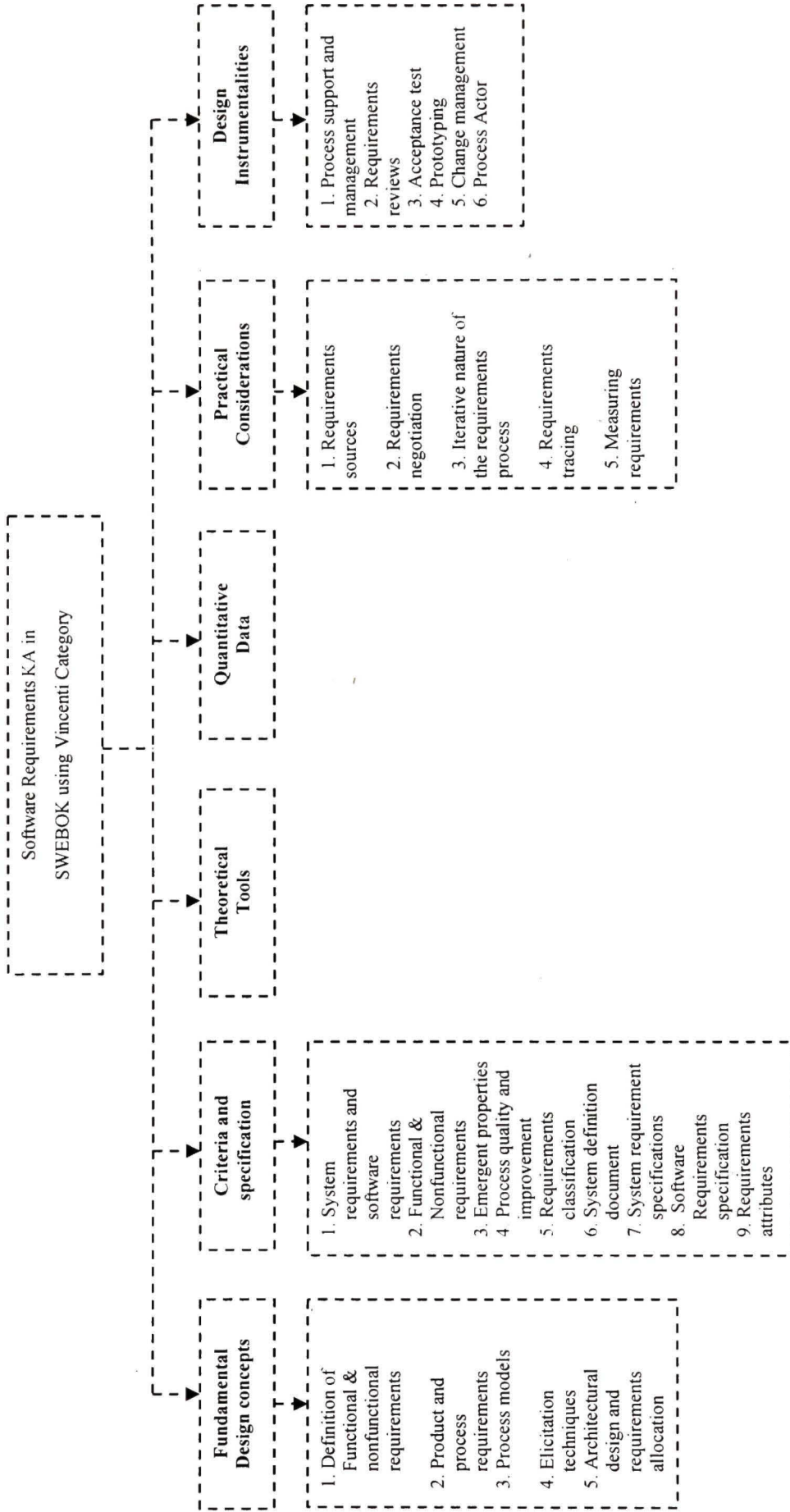
**Design Instrumentalities**

## ANNEX II

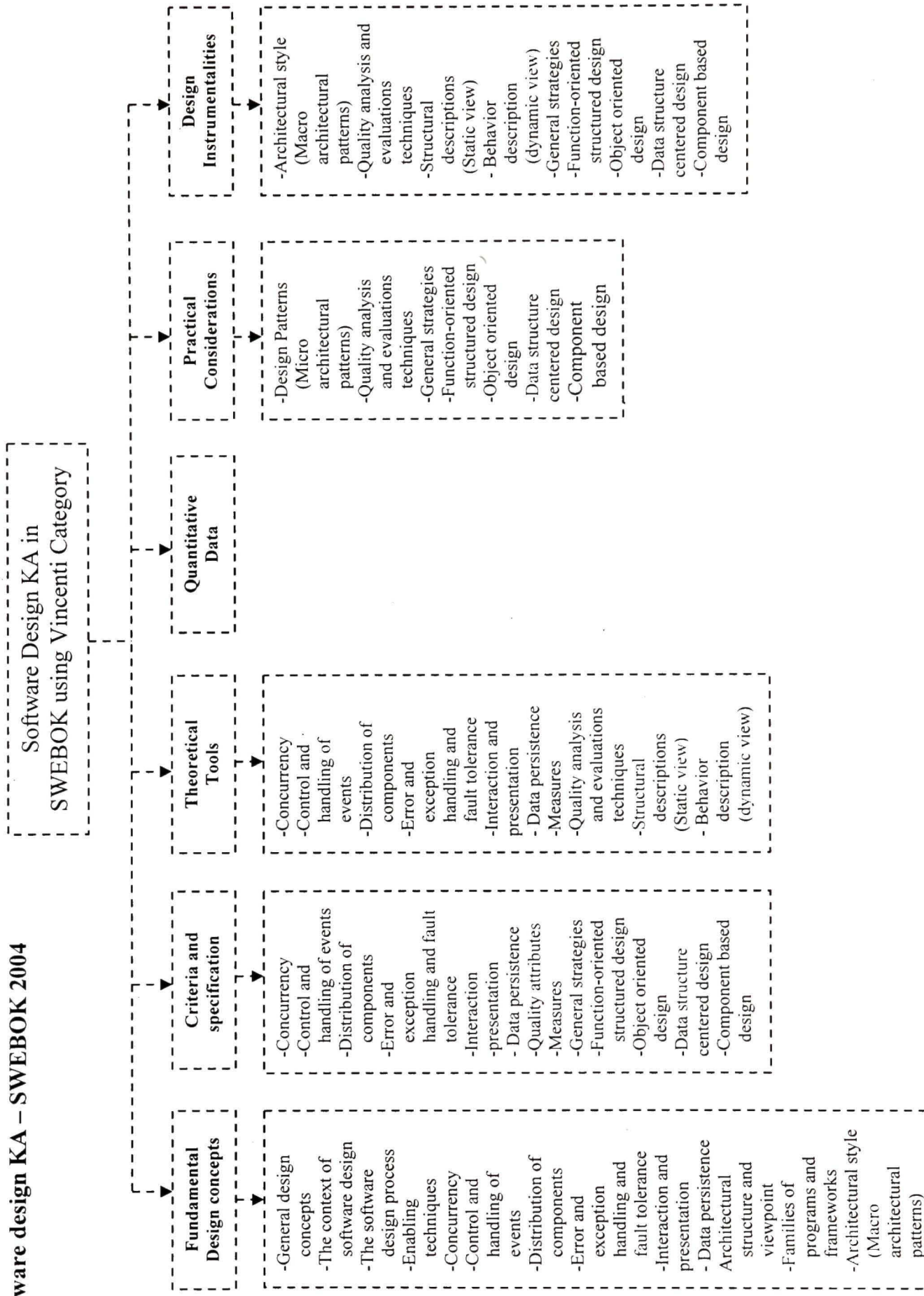
### PRESENTATION OF THE KNOWLEDGE CONTAINED IN THE 3 KAS OF THE SWEBOK GUIDE FOR THE ("SOFTWARE REQUIREMENTS", "SOFTWARE DESIGN" AND "SOFTWARE CONSTRUCTION") KA BY CATEGORIES OF VINCENTI'S ENGINEERING KNOWLEDGE.

The Annex presents the new breakdown of the 3KAs "Software Requirements", "Software design" and "Software construction" KAs based on the categories of engineering knowledge

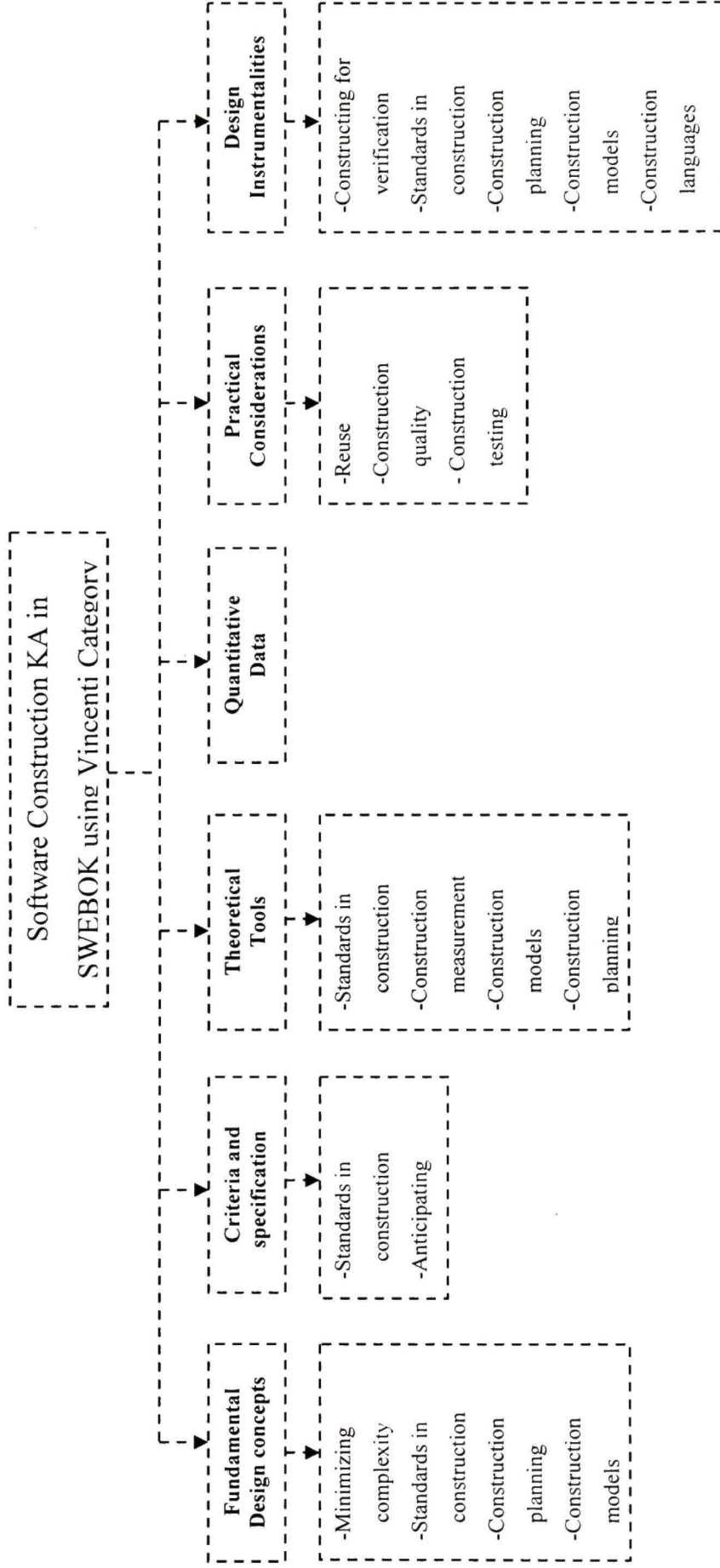
#### 1. Software Requirements KA – SWEBOK 2004



2. Software design KA – SWEBOK 2004



### 3. Software Construction KA– SWEBOK 2004



## ANNEX III

### .MAPPING RESULTS BETWEEN THE ENGINEERING FUNDAMENTAL PRINCIPLES AND THE ENGINEERING CRITERIA

This Annex presents: C-1, C-2, C-3 and C-4

C-1: Result of mapping the candidate FPs to Vincenti engineering criteria

		Problem	Criteria	Techniques	Quality	Testing	Measurement	C#7. Assessment
1	Align incentives for developer and customer	I	I					
2	Apply and use quantitative measurements in decision making				I		D	
3	Build software so that it needs a short user manual		I	I	I	I		I
4	Build with and for reuse		D	I	I	I		I
5	Define software artifacts rigorously	I	I	I	I	I	I	I
6	Design for maintenance	I	I	I	I	I		
7	Determine requirements now	D	I	I	I	I		
8	Don't overstrain your hardware		I					
9	Don't try to retrofit quality	I	I	I	D	D	I	I
10	Don't write your own test plans			I		I		I
11	Establish a software process that provides flexibility		I	I	D	I		
12	Fix requirements specification error now	I	I	I	I			
13	Give product to customers early							
14	Grow systems incrementally	I	I	I	D	I		
15	Implement a disciplined approach and improve it continuously		I	I	D	I		
16	Invest in the understanding of the problem	D	I	I	I	I		
17	Involve the customer	I			I			I
18	Keep design under intellectual control	I	I		I		I	I
19	Maintain clear accountability for results							I
20	Produce software in a stepwise fashion	I	I	I	I	I	I	I
21	Quality is the top priority; long term productivity is a natural consequence of high quality					D	D	
22	Rotate (high performer) people through product assurance				I			
23	Since change is inherent to software, plan for it and manage it	I	I	I	D	I		
24	Since tradeoffs are inherent to software engineering, make them explicit and document it	D	I	I	I	I		
25	Strive to have a peer, rather than a customer, find a defect			I	I			I
26	Tailor cost estimation methods			I	I	I		
27	To improve design, study previous solutions to similar problems		I		I			D
28	Use better and fewer people				I			
29	Use documentation standards			I				
30	Write programs for people first		I					
31	Know software engineering's techniques before using development tools			I				I
32	Select tests based on the likelihood that they will find faults	I		I				I
33	Choose a programming language to assure maintainability			I				I
34	In face of unstructured code, rethink the module and redesign it from scratch.	I	I	I	I	I		

## C-2: Result of mapping the candidate FPs to the IEEE-ACM engineering criteria

		Decision making	Measurements	Disciplined process	Engineer's roles	Use of Tools	Development & validation	Reuse design
1	Align incentives for developer and customer			I				
2	Apply and use quantitative measurements in decision making	D	D					
3	Build software so that it needs a short user manual	I						
4	Build with and for reuse			I				D
5	Define software artifacts rigorously			D				
6	Design for maintenance	I		I	D			I
7	Determine requirements now	I		I				
8	Don't overstrain your hardware	I	I					
9	Don't try to retrofit quality	I	I	I		I		
10	Don't write your own test plans			I	D			
11	Establish a software process that provides flexibility			I				I
12	Fix requirements specification error now	D						
13	Give product to customers early							
14	Grow systems incrementally	I		I				
15	Implement a disciplined approach and improve it continuously	I		D				
16	Invest in the understanding of the problem	I		I				
17	Involve the customer			I				
18	Keep design under intellectual control	I		D				
19	Maintain clear accountability for results		I	I	I			
20	Produce software in a stepwise fashion			I				
21	Quality is the top priority; long term productivity is a natural consequence of high quality	D		I	I	I		
22	Rotate (high performer) people through product assurance				D			
23	Since change is inherent to software, plan for it and manage it	I		I				
24	Since tradeoffs are inherent to software engineering, make them explicit and document it	D		I				
25	Strive to have a peer, rather than a customer, find a defect			I	D			
26	Tailor cost estimation methods		D					
27	To improve design, study previous solutions to similar problems			I				D
28	Use better and fewer people							
29	Use documentation standards			I				
30	Write programs for people first							
31	Know software engineering's techniques before using development tools			I		D		
32	Select tests based on the likelihood that they will find faults	I		I	I			
33	Choose a programming language to assure maintainability					I		
34	In face of unstructured code, rethink the module and redesign it from scratch.	I		I				

The color codes are the following:



CF principles with a direct mapping to engineering criteria



	CF principles with an indirect mapping to engineering criteria
	CF principles with no mapping to engineering criteria

C-3: Categorization of the mapping of Vincenti engineering criteria to the Candidate FPs

		<i>C#1. Problem</i>	<i>C#2. Criteria</i>	<i>C#3. Techniques</i>	<i>C#4. Quality</i>	<i>C#5. Schema of quality</i>	<i>C#6. Measurement</i>	<i>C#7. Assessment</i>
2	Apply and use quantitative measurements in decision making				I		D	
4	Build with and for reuse		D	I	I	I		I
7	Determine requirements now	D	I	I	I	I		
9	Don't try to retrofit quality	I	I	I	D	D	I	I
11	Establish a software process that provides flexibility		I	I	D	I		
14	Grow systems incrementally	I	I	I	D	I		
15	Implement a disciplined approach and improve it continuously		I	I	D	I		
16	Invest in the understanding of the problem	D	I	I	I	I		
21	Quality is the top priority; long term productivity is a natural consequence of high quality				D	D		
23	Since change is inherent to software, plan for it and manage it	I	I	I	D	I		
24	Since tradeoffs are inherent to software engineering, make them explicit and document it	D	I	I	I	I		
27	To improve design, study previous solutions to similar problems		I		I			D
1	Align incentives for developer and customer	I	I					
3	Build software so that it needs a short user manual		I		I	I		I
5	Define software artifacts rigorously	I	I	I	I	I	I	I
6	Design for maintenance	I	I	I	I	I		
8	Don't overstrain your hardware		I					
10	Don't write your own test plans			I		I		I
12	Fix requirements specification error now	I	I	I	I			
17	Involve the customer	I			I			I
18	Keep design under intellectual control	I	I		I		I	I
19	Maintain clear accountability for results							I
20	Produce software in a stepwise fashion	I	I	I	I	I	I	I
22	Rotate (high performer) people through product assurance				I			
25	Strive to have a peer, rather than a customer, find a defect			I	I			I
26	Tailor cost estimation methods			I	I	I		
28	Use better and fewer people				I			
29	Use documentation standards			I				
30	Write programs for people first		I					
31	Know software engineering's techniques before using development tools			I				I
32	Select tests based on the likelihood that they will find faults	I		I				I
33	Choose a programming language to assure maintainability			I				I
34	In face of unstructured code, rethink the module and redesign it from scratch.	I	I	I	I	I		
13	Give product to customers early							

C-4: Categorization of the mapping of IEEE-ACM engineering criteria to the Candidate FPs

		<i>C1. Decision making</i>	<i>C2. Measurements</i>	<i>C3. Disciplined process</i>	<i>C4 Engineer's roles</i>	<i>C5 Use of Tools</i>	<i>C6. Development validation</i>	<i>C7. Reuse design</i>
2	Apply and use quantitative measurements in decision making	D	D					
4	Build with and for reuse			I				D
5	Define software artifacts rigorously			D				
6	Design for maintenance	I		I	D			I
10	Don't write your own test plans			I	D			
12	Fix requirements specification error now	D						
15	Implement a disciplined approach and improve it continuously	I		D				
18	Keep design under intellectual control	I		D				
21	Quality is the top priority; long term productivity is a natural consequence of high quality	D		I	I	I		
22	Rotate (high performer) people through product assurance				D			
24	Since tradeoffs are inherent to software engineering, make them explicit and document it	D		I				
25	Strive to have a peer, rather than a customer, find a defect			I	D			
26	Tailor cost estimation methods		D					
27	To improve design, study previous solutions to similar problems			I				D
31	Know software engineering's techniques before using development tools			I		D		
1	Align incentives for developer and customer			I				
3	Build software so that it needs a short user manual	I						
7	Determine requirements now	I		I				
8	Don't overstrain your hardware	I	I					
9	Don't try to retrofit quality	I	I	I		I		
11	Establish a software process that provides flexibility			I				I
14	Grow systems incrementally	I		I				
16	Invest in the understanding of the problem	I		I				
17	Involve the customer			I				
19	Maintain clear accountability for results		I	I	I			
20	Produce software in a stepwise fashion			I				
23	Since change is inherent to software, plan for it and manage it	I		I				
29	Use documentation standards			I				
32	Select tests based on the likelihood that they will find faults	I		I	I			
33	Choose a programming language to assure maintainability					I		
34	In face of unstructured code, rethink the module and redesign it from scratch.	I		I				
13	Give product to customers early							
28	Use better and fewer people							
30	Write programs for people first							

## ANNEX IV.

### DETAILED RESULTS FOR THE IDENTIFICATION OF THE SOFTWARE ENGINEERING PRINCIPLES WITHIN THE CONTENT OF THE

#### SWEBOK GUIDE – ISO TR 19759

This Annex introduces the detailed results related to the knowledge areas mentioned in section 5.3.2 “Results in other KAs”.

##### **1. “Software design” knowledge area**

The following six FP were identified as being present in the “Software design” KA:

- (1) Apply and use quantitative measurements in decision making
- (2) Build with and for reuse
- (3) Grow systems incrementally
- (4) Implement a disciplined approach and improve it continuously
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it (is covered implicitly and is mentioned in the introduction).
- (9) To improve design, study previous solutions to similar problems

The missing ones are the following:

- (5) Invest in the understanding of the problem
- (7) Since change is inherent to software, plan for it and manage it

##### **2. Software Construction knowledge area**

The following five FP were identified as being present in the “Software construction” KA:

- (1) Apply and use quantitative measurements in decision making
- (2) Build with and for reuse
- (3) Grow systems incrementally
- (4) Implement a disciplined approach and improve it continuously
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality
- (7) Since change is inherent to software, plan for it and manage it

The missing ones are the following:

- (5) Invest in the understanding of the problem
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it
- (9) To improve design, study previous solutions to similar problems

### 3. Software Testing knowledge area

The following six FP were identified as being present in the “Software testing” KA:

- (1) Apply and use quantitative measurements in decision making
- (2) Build with and for reuse
- (3) Grow systems incrementally
- (4) Implement a disciplined approach and improve it continuously
- (7) Since change is inherent to software, plan for it and manage it
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it

The missing ones are the following:

- (5) Invest in the understanding of the problem
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality
- (9) To improve design, study previous solutions to similar problems

### 4. Software Maintenance knowledge area

The following five FP were identified as being present in the “Software maintenance” KA:

- (1) Apply and use quantitative measurements in decision making
- (3) Grow systems incrementally
- (4) Implement a disciplined approach and improve it continuously
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality
- (7) Since change is inherent to software, plan for it and manage it

The missing ones are the following:

- (2) Build with and for reuse
- (5) Invest in the understanding of the problem
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it
- (9) To improve design, study previous solutions to similar problems

### 5. Software Configuration Management knowledge area

The following three FP were identified as being present in the Software configuration management KA:

- (1) Apply and use quantitative measurements in decision making
- (4) Implement a disciplined approach and improve it continuously
- (7) Since change is inherent to software, plan for it and manage it

The missing ones are the following:

- (2) Build with and for reuse
- (3) Grow systems incrementally
- (5) Invest in the understanding of the problem
- (6) Quality is the top priority; long term productivity is a natural consequence of high quality
- (8) Since tradeoffs are inherent to software engineering, make them explicit and document it
- (9) To improve design, study previous solutions to similar problems

### **6. Software Engineering Management knowledge area**

The following three FP were identified as being present in the “Software Engineering Management” KA:

Apply and use quantitative measurements in decision making

(4) Implement a disciplined approach and improve it continuously

(6) Quality is the top priority; long term productivity is a natural consequence of high quality

The missing ones are the following:

(2) Build with and for reuse

(3) Grow systems incrementally

(5) Invest in the understanding of the problem

(7) Since change is inherent to software, plan for it and manage it

(8) Since tradeoffs are inherent to software engineering, make them explicit and document it

(9) To improve design, study previous solutions to similar problems

### **7. Software Engineering Process knowledge area**

The following three FP were identified as being present in the “Software Engineering Process” KA:

(1) Apply and use quantitative measurements in decision making

(4) Implement a disciplined approach and improve it continuously

(6) Quality is the top priority; long term productivity is a natural consequence of high quality

The missing ones are the following:

(2) Build with and for reuse

(3) Grow systems incrementally

(5) Invest in the understanding of the problem

(7) Since change is inherent to software, plan for it and manage it

(8) Since tradeoffs are inherent to software engineering, make them explicit and document it

(9) To improve design, study previous solutions to similar problems

## ANNEX V

### PROPOSED DRAFT: OPERATIONAL GUIDELINES OF THE NINE FUNDAMENTAL PRINCIPLES OF SOFTWARE ENGINEERING BASED ON THE SWEBOK GUIDE CONTENT - ISO TR 19759 (2004)

The content of this draft proposes operational guidelines for the main knowledge areas of the SWEBOK Guide such as “Software requirements” (p.1), “Software design” p. 14, “Software construction” p.25, “Software testing” p.38, “Software maintenance” p.51

The content of this Annex for each of the KA is structured as follows:

- Operational concepts.
- Operational scenario.
- Operational capabilities.
- Operational improvement.
- Operational impacts

#### Proposed Operational Guidelines for Software Requirements The Proposed Operational Guidelines (Software Requirements for SWEBOK Guide)

The operational guidelines of the 9 FP in the “Software requirements” KA of SWEBOK Guide.

Software Engineering Principles	SWEBOK Guide Software Requirements Knowledge Area
<b>1. Principle #1 “Apply and Use Quantitative Measurements in Decision Making”</b>	<ul style="list-style-type: none"><li>➤ Quantifiable requirements.</li><li>➤ Measuring requirements.</li></ul>
<b>2. Principle #2 “Build with and for Reuse”</b>	<ul style="list-style-type: none"><li>➤ Conceptual Modeling</li><li>➤ Architectural Design and Requirements Allocation</li><li>➤ Acceptance Test</li></ul>
<b>3. Principle #3 “Grow System Incrementally”</b>	<ul style="list-style-type: none"><li>➤ Requirement Elicitation</li><li>➤ Requirement Analysis</li><li>➤ Requirement Specification</li><li>➤ Requirement Validation</li></ul>
<b>4. Principle #4 “Implement A Disciplined Approach and Improve It Continuously”</b>	<ul style="list-style-type: none"><li>➤ Requirement Elicitation</li><li>➤ Requirement Analysis</li><li>➤ Requirement Specification</li><li>➤ Requirement Validation</li></ul>
<b>5. Principle #5 “Invest in The Understanding</b>	<ul style="list-style-type: none"><li>➤ Requirement Elicitation</li></ul>

<b>of The Problem”</b>	➤ Requirement Analysis
<b>6. Principle #6 “Quality is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”</b>	<ul style="list-style-type: none"> <li>➤ “Process Quality and Improvement” (topic).</li> <li>➤ “Requirements Validation” (sub-area).</li> </ul>
<b>7. Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It”</b>	<ul style="list-style-type: none"> <li>➤ Iterative Nature of the Requirements Process</li> <li>➤ Change management</li> <li>➤ Requirements attributes</li> <li>➤ Requirements tracing</li> </ul>
<b>8. Principle #8 “Since Tradeoffs are Inherent to Software Engineering, Make Them Explicit and Document them”</b>	➤ Requirements Negotiation

By applying principle #1 “Apply and use quantitative measurements in decision making” of the software requirement (Quantifiable and Measuring requirements) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

### **Principle #1 “Apply and Use Quantitative Measurements in Decision Making”**

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Quantifiable requirements.</li> <li>– Measuring requirements.</li> </ul>
<b>Operational Scenario</b>	<p><b>Measurement process</b></p> <ul style="list-style-type: none"> <li>– At the end of the software requirements phase, the ‘software specification document’ is produced: this document contains the functional and nonfunctional requirements.</li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– The functional requirements can be measured right from the requirements phase, using a functional size measurements method (eg. COSMIC – ISO 19761).</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Input to estimate the cost of design, development, test or the cost of maintenance task.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #2 “*Build with and for reuse*” of the software requirement (Conceptual Modeling, Architectural Design and Requirements Allocation, Acceptance Test) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #2 “*Build with and for Reuse*”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Conceptual Modeling</li> <li>– Architectural Design and Requirements Allocation</li> <li>– Acceptance Test</li> </ul>
<b>Operational Scenario</b>	<p><b>Build with and for reuse</b></p> <ul style="list-style-type: none"> <li>– After finishing the task of eliciting the requirements the software engineer start analyzing the requirements by classifying them,</li> <li>– Modeling them using one of the following models: data and control flows, state models, event traces, user interactions, object models, data models and many others.</li> </ul>
<b>Operational Capabilities</b>	<p><b>Conceptual Modeling</b></p> <ul style="list-style-type: none"> <li>– The software engineer can be interested in developing the system by, for example, reusing the conceptual models for the set of requirements.</li> <li>– Components can be reused in requirement allocation and finally “Test cases” can be reused to conduct acceptance test to validate that the software satisfies the requirements.</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– These models, components and test cases can be carefully defined and documented so that they may be reused.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #3 “*Grow system incrementally*” of the software requirement (Requirement Elicitation, Requirement Analysis, Requirement Specification, Requirement Validation) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #3 “*Grow System Incrementally*”**

---

- Requirement Elicitation
-



<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Requirement Analysis</li> <li>– Requirement Specification</li> <li>– Requirement Validation</li> </ul>
<b>Operational Scenario</b>	– The operational guidelines for the principle # 3: “Grow system incrementally’ suggest that software should be build by increment: the implementation of this principle is similar to principle #4: “ <i>Implement a disciplined approach and improve it continuously</i> ” the only difference is in the input.
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Focusing on a small set of requirements (that have high priority).</li> <li>– Increasing the number of requirements slowly in the set of requirements.</li> </ul>
<b>Operational improvements</b>	– Software requirements growth.
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #4 “*Implement a disciplined approach and improve it continuously*” in the software requirement (Requirement Elicitation, Requirement Analysis, Requirement Specification, Requirement Validation) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document, table xx put forward innovative operational concepts as follows:

---

**Principle #4 “*Implement A Disciplined Approach and Improve It Continuously*”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Requirement Elicitation</li> <li>– Requirement Analysis</li> <li>– Requirement Specification</li> <li>– Requirement Validation</li> </ul>
	<b>Requirements Activities</b>
<b>Operational Scenario</b>	<p><b>Activity 1: Requirements elicitation:</b></p> <ul style="list-style-type: none"> <li>– Software engineer starts by defining the sources of requirements by identifying one of the many sources of each requirement.</li> <li>– Start collecting requirements by using different elicitation techniques such as interviews, scenarios, prototypes, facilitated meetings and observations.</li> </ul> <p><b>Activity 2: Requirements Analysis:</b></p>

---

	<ul style="list-style-type: none"> <li>– Software engineer starts by classifying the requirements whether they are functional or non functional.</li> <li>– Develop a conceptual model of the real world problem using one of the following models: data and control flows, state models, event traces, user interactions, object models, data models and many others.</li> </ul> <p><b>Activity 3: Requirements Specification:</b></p> <ul style="list-style-type: none"> <li>– In the software specification phase the software engineer produces a document. For complex systems three kinds of documents are produced: “System Definition”, “System Requirements Specification” and “Software Requirements Specification”.</li> </ul> <p><b>Activity 4: Requirements Validation:</b></p> <ul style="list-style-type: none"> <li>– In the software requirement KA the following artifacts are subject to validation &amp; verification: the system definition document, the system specification document, the software requirements specification document and the baseline specification.</li> </ul>
<p><b>Operational Capabilities</b></p>	<p><b>Activity 1: Requirements elicitation:</b></p> <ul style="list-style-type: none"> <li>– Categorized the source of requirements into goals, domain knowledge, stakeholders, the operational environment and the organizational environment.</li> </ul> <p><b>Activity 2: Requirements Analysis:</b></p> <ul style="list-style-type: none"> <li>– Derived requirements, type of requirements product or process, requirements priority (classified on a fixed point scale: mandatory, highly desirable, desirable and optional), the scope of requirements and finally the estimation of volatility and stability requirements.</li> <li>– Allocate requirements to components,</li> <li>– Negotiate requirements</li> </ul> <p><b>Activity 3: Requirements Specification:</b></p> <ul style="list-style-type: none"> <li>– For simplified complex systems the “Software Requirements Specification” document is produced.</li> </ul> <p><b>Activity 4: Requirements Validation:</b></p> <ul style="list-style-type: none"> <li>– Reviews, prototyping, validating the quality of models, identifying and designing acceptance test to validate that the finished product satisfies the requirements.</li> </ul>
<p><b>Operational improvements</b></p>	<ul style="list-style-type: none"> <li>– Defining the sources of requirements</li> <li>– Using different elicitation techniques</li> <li>– Classifying the requirements</li> <li>– Developing a conceptual model</li> <li>– Simplified complex systems</li> <li>– Validating the product</li> </ul>
<p><b>Operational impacts</b></p>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #5 “Invest in the understanding of the problem” of the software requirement (Requirement Elicitation, Requirement Analysis) for SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #5 “Invest in The Understanding of The Problem”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Requirement Elicitation</li> <li>– Requirement Analysis</li> </ul>
<b>Operational Scenario</b>	<p><b>Requirements Activities</b></p> <p><b>Activity 1: Requirements elicitation:</b></p> <ul style="list-style-type: none"> <li>– Defining the sources of requirements by identifying one of the many sources of each requirement.</li> <li>– Collecting requirements by using different elicitation techniques such as interviews, scenarios, prototypes, facilitated meetings and observations.</li> </ul> <p><b>Activity 2: Requirements Analysis:</b></p> <ul style="list-style-type: none"> <li>– Classifying the requirements whether they are functional or non functional.</li> <li>– Developing a conceptual model of the real world problem using one of the following models: data and control flows, state models, event traces, user interactions, object models, data models and many others.</li> </ul>
<b>Operational Capabilities</b>	<p><b>Activity 1: Requirements elicitation:</b></p> <ul style="list-style-type: none"> <li>– Categorized the source of requirements into goals, domain knowledge, stakeholders, the operational environment and the organizational environment.</li> </ul> <p><b>Activity 2: Requirements Analysis:</b></p> <ul style="list-style-type: none"> <li>– Derived requirements, type of requirements product or process, requirements priority (classified on a fixed point scale: mandatory, highly desirable, desirable and optional), the scope of requirements and finally the estimation of volatility and stability requirements.</li> <li>– Allocate requirements to components,</li> <li>– Negotiate requirements</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– defining the sources of requirements</li> <li>– using different elicitation techniques</li> <li>– classifying the requirements</li> <li>– Developing a conceptual model</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

By applying principle #6 “*Quality is the top priority; long term productivity is a natural consequence of high quality*” of the software requirement (“Process Quality and Improvement” (topic), “Requirements Validation” (sub-area)) for SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #6 “*Quality is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality*”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– “Process Quality and Improvement” (topic).</li> <li>– “Requirements Validation” (sub-area).</li> </ul>
<b>Operational Scenario</b>	<p><b>Process quality and improvement</b></p> <ul style="list-style-type: none"> <li>– In this topic the software engineer evaluate the quality of the requirements process with the help of the quality standards.</li> <li>– Process improvement models are used to orient the improvements of the requirements process activities.</li> </ul> <p><b>Requirements Validation:</b></p> <ul style="list-style-type: none"> <li>– In the software requirement KA the following artifacts are subject to validation &amp; verification: the system definition document, the system specification document, the software requirements specification document and the baseline specification.</li> </ul>
<b>Operational Capabilities</b>	<p><b>Process quality and improvement</b></p> <ul style="list-style-type: none"> <li>– Evaluate the quality of the requirements process</li> <li>– Improvements of the requirements process activities</li> </ul> <p><b>Requirements Validation:</b></p> <ul style="list-style-type: none"> <li>– Reviews, prototyping, validating the quality of models, identifying and designing acceptance test to validate that the finished product satisfies the requirements.</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Validating the product</li> <li>– Validating the quality of models</li> <li>– Quality of the requirements process</li> <li>– Improving process activities of the requirements.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

By applying principle #7 “*Since change is inherent to software, plan for it and manage it*” of the software requirement (Iterative Nature of the Requirements Process, Change management, Requirements attributes and Requirements tracing ) for the SWEBOK

taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Iterative Nature of the Requirements Process</li> <li>– Change management</li> <li>– Requirements attributes</li> <li>– Requirements tracing</li> </ul>
<b>Operational Scenario</b>	<p>Change management necessitates the following tasks in the software requirements process.</p> <ul style="list-style-type: none"> <li>– Identifying the requirements that possibly change,</li> <li>– Define the review, Approve the process,</li> <li>– Perform the change,</li> <li>– Apply requirements tracing,</li> <li>– Apply impact analysis,</li> <li>– Apply software configuration management and</li> <li>– Report change history.</li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Change management</li> <li>– Requirements attributes</li> <li>– Requirements tracing</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Iterative Nature of the Requirements Process</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #8 “Since tradeoffs are inherent to software engineering, make them explicit and document them” of the software requirement (Requirements Negotiation) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #8 “Since Tradeoffs are Inherent to Software Engineering, Make Them Explicit and Document them”**

---

<b>Operational</b>	– Requirements Negotiation
--------------------	----------------------------

---

<b>Concepts</b>	
<b>Operational Scenario</b>	Stakeholders may require incompatible features, between requirements and resources or between functional and non-functional requirements <ul style="list-style-type: none"> <li>– Identify conflict</li> <li>– Consult with stakeholders to negotiate an acceptable compromise.</li> <li>– Trace decision back to customer</li> <li>– Implement the decision</li> </ul>
<b>Operational Capabilities</b>	– Software Requirements Negotiation
<b>Operational improvements</b>	– Solve the software requirements conflict – Improve the decision process
<b>Operational impacts</b>	– Impacts during development – Organizational impacts – User impacts – Developer impacts – Buyer impacts

### **Proposed Operational Guidelines for Software Design**

#### **The Proposed Operational Guidelines (Software Design for SWEBOK Guide)**

The operational guidelines of the 9 FP in the “Software design” KA of SWEBOK Guide.

<b>Software Engineering Principles</b>	<b>SWEBOK Guide Software Design Knowledge Area</b>
<b>1. Principle #1 “Apply and Use Quantitative Measurements in Decision Making”</b>	➤ Measures
<b>2. Principle #2 “Build with and for Reuse”</b>	➤ Design Patterns (micro architectural patterns): ➤ Families of Programs and Frameworks ➤ Component-Based Design:
<b>3. Principle #4 “Implement A Disciplined Approach and Improve It Continuously”</b>	➤ The software design process ➤ Architectural structures and viewpoints ➤ Architectural styles ➤ Design patterns (micro architectural patterns) ➤ Families of programs and frameworks ➤ Structural descriptions (static view)

	<ul style="list-style-type: none"> <li>➤ Behavior descriptions (dynamic view)</li> <li>➤ Software Design Strategies and Methods</li> <li>➤ Enabling techniques and Keys issues in Software Design (sub-area)</li> </ul>
<b>4. Principle #6 “Quality is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”</b>	<ul style="list-style-type: none"> <li>➤ Quality Attributes</li> <li>➤ Quality Analysis and Evaluation Techniques.</li> </ul>
<b>5. Principle #9 “To improve design, study previous solutions to similar problems”</b>	<ul style="list-style-type: none"> <li>➤ Architectural styles</li> <li>➤ Design patterns (micro architectural patterns)</li> </ul>

By applying principle #1 “Apply and use quantitative measurements in decision making” for the software design (Measures) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

### Principle #1 “Apply and Use Quantitative Measurements in Decision Making”

<b>Operational Concepts</b>	– Measures
<b>Operational Scenario</b>	<p><b>Measurement process</b></p> <p>The measurement process can take place during the design phase, two types of measures are referred to in this KA:</p> <ul style="list-style-type: none"> <li>– Function oriented (structured) design measures represented by modules, interfaces, hidden information, concurrency, message passing, invocation of operations and overall program structure in a comprehensive way.</li> <li>– Object oriented design measure represented as a class diagram. Different measures can be computed form both structure chart and class diagram to serve as a foundation for the formal definition of object-oriented design measure.</li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– The definitions of measures shall be precise and unambiguous (this, however, is a goal that is not achieved easily).</li> <li>– The measures shall be based on design artefacts that are typically constructed in the design phase.</li> <li>– The measures shall be suited for automatic measurement as far as possible.</li> </ul>

---

	– Measurement shall be possible even if the design is incomplete or not detailed.
<b>Operational improvements</b>	– Evaluate the size, structure and quality.
<b>Operational impacts</b>	– Impacts during development – Organizational impacts – User impacts – Developer impacts – Buyer impacts

---

By applying principle #2 “*Build with and for reuse*” for the software design (Design Patterns (micro architectural patterns), Families of Programs and Frameworks and Component-Based Design) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

### **Principle #2 “*Build with and for Reuse*”**

---

<b>Operational Concepts</b>	– Design Patterns (micro architectural patterns): – Families of Programs and Frameworks – Component-Based Design:
<b>Operational Scenario</b>	<p><b>Build with and for reuse</b></p> <ul style="list-style-type: none"> <li>– Can be applicable for the following elements: patterns, components, families of programs, and frameworks.</li> <li>– Patterns create re-usable descriptions of how a building's architecture could support the architecture behaviours. The pattern give a resource to ensure they got all the details right.</li> <li>– When the team specifies the response that works best for them (and their users), they can codify it in a pattern. Future teams, need to respond to that desired behaviour, respond similarly, meeting established user expectations while leveraging the previous work.</li> <li>– Many teams, starting their library, often turn to one of the off-the-shelf pattern libraries that are popping up on the scene. While these are often well documented and inexpensive (some are open source and free), they turn out to be less helpful than it would seem. This is because they are generic solutions, not taking the project's specific technological constraints and business requirements into account. The most helpful pattern libraries make these constraints and requirements their focus.</li> <li>– Components specify the design response to the pixel level. Because they</li> </ul>

---



---

often are represented by their code, they also embody the specific interaction behaviour. They contain the brand styling elements, such as the fonts, color, and look.

- Developers use these components to piece together the specifics of the design. Once built, they are ready-made elements that can easily plug into any new screen. This speeds every part of the development process, from early prototypes to final deployment.
- Interaction design frameworks describe entire subsystems of patterns. For example, a login subsystem needs a pattern where users enter an id and password. But it also needs a pattern for password recovery, a pattern for setting up the id initially, a pattern for creating new ids, and a pattern for changing the password.
- Teams create the frameworks by looking at what other designs have been done. They become a checklist, helping the team ensure they've all the right patterns to start their design.
- Frameworks are a high-level of abstraction. They don't talk to the specific design. Instead, that is filled in by the components based on the individual patterns.

---

<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Patterns create re-usable descriptions of how a building's architecture could support the architecture behaviours</li> <li>– Developers use these components to piece together the specifics of the design.</li> <li>– Interaction design frameworks describe entire subsystems of patterns</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– minimal effort</li> <li>– components are close to the final implementation</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– Designer impacts</li> <li>– Developer impacts</li> </ul>

---

By applying principle #4 “*Implement a disciplined approach and improve it continuously*” to the software design (The software design process Architectural Structures and viewpoints, Design patterns (micro architectural patterns), Families of programs and frameworks, Structural descriptions (static view), Behavior descriptions (dynamic view), Software Design Strategies and Methods (sub-area), Enabling techniques, Keys issues in Software Design (sub-area)) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #4 “Implement A Disciplined Approach and Improve It Continuously”**

---

**Operational Concepts**

- The software design process
  - Architectural Structures and viewpoints
  - Design patterns (micro architectural patterns)
  - Families of programs and frameworks
  - Structural descriptions (static view)
  - Behavior descriptions (dynamic view)
  - Software Design Strategies and Methods
  - Enabling techniques and Keys issues in Software Design (sub-area)
- 

**Software design steps**

**Operational Scenario**

Software design is composed of two steps process architectural design and detailed design. Software design produces solutions in form of models The following describes the software design steps:

**Step 1: Architectural design**

Architectural design for the software is decomposed and organized into components using the following different elements:

- Use the general strategies to help guide the design process. For instance: divide-and-conquer and stepwise refinement, top-down vs. bottom-up strategies, data abstraction and information hiding, use of heuristics, use of patterns and pattern languages, iterative and incremental approach.
  - Identify the views necessary to represent the system such as: logical view, physical view, process view and development view.
  - Define the architectural style. It describes the software high level organization.
    - General structure (for example, layers, pipes, and filters, blackboard)
    - Distributed systems (for example, client-server, three-tiers, broker)
    - Interactive systems (for example, Model-View-Controller, Presentation-Abstraction-Control)
    - Adaptable systems (for example, micro-kernel, reflection)
    - Others (for example, batch, interpreters, process control, rule-based).
-

- 
- Use the different methods for modeling the structural and behavioral descriptions of the system such as: Function-Oriented (Structured) Design, Object-Oriented Design, Data-Structure-Centered Design, Component-Based Design (CBD) and other methods.
  - Model the structural description of the system: this represents the static view using different notations. For instance: Architecture description languages, Class and object diagrams, Component diagrams, Class responsibility collaborator cards, Deployment diagrams, Entity-relationship diagrams, Interface description languages, Jackson structure diagrams, Structure charts.
  - Model the behavioral description of the system dynamic view such as: Activity diagrams, Collaboration diagrams, Data flow diagrams, Decision tables and diagrams, Flowcharts and structured flowcharts, Sequence diagrams, State transition and state chart diagrams, Formal specification languages, Pseudo-code and program design languages

## **Step 2: Detailed design**

Detailed design describes the specific behavior of the components already decomposed in the architectural step.

- In this step more low level details are given for the previous architectural design steps.
- The frameworks and design patterns are used to describe details at a lower level (the micro-architecture). For instance - some examples of design patterns:
  - Creational patterns (for example, builder, factory, prototype, and singleton)
  - Structural patterns (for example, adapter, bridge, composite, decorator, façade, flyweight, and proxy)
  - Behavioral patterns (for example, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor).

## **Enabling Techniques and Key issues**

- Enabling techniques and key issues are necessary to implement principle #4 “Implement a disciplined approach and improve it continuously” in the “Software Design”.
  - Various enabling techniques and key issues are presented to evolution of scientific knowledge involves learning by observing, formulating
-

	<p>theories, and performing experiments. Many fields like physics, medicine and manufacturing use experimentation as a means to encapsulate as well as to verify and validate knowledge.</p> <ul style="list-style-type: none"> <li>– Appropriate abstractions from the reality of software development to learn how to improve skills and competencies. For that purpose, process, product and quality models are developed.</li> <li>– Key issues: Concurrency, Control and Handling of Events, Distribution of Components, Error and Exception Handling and Fault Tolerance, Interaction and Presentation and Data Persistence.</li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– General structure (for example, layers, pipes, and filters, blackboard)</li> <li>– Distributed systems (for example, client-server, three-tiers, broker)</li> <li>– Interactive systems (for example, Model-View-Controller, Presentation-Abstraction-Control)</li> <li>– Adaptable systems (for example, micro-kernel, reflection)</li> <li>– Others (for example, batch, interpreters, process control, rule-based).</li> <li>– Creational patterns (for example, builder, factory, prototype, and singleton)</li> <li>– Structural patterns (for example, adapter, bridge, composite, decorator, façade, flyweight, and proxy)</li> <li>– Behavioral patterns (for example, command, interpreter, iterator, mediator, memento, observer, state, strategy, template, visitor).</li> <li>– Enabling techniques: Abstraction, Coupling and cohesion, Decomposition and modularization, Encapsulation/information hiding, Separation of interface and implementation, Sufficiency, completeness and primitiveness.</li> <li>– Key issues: Concurrency, Control and Handling of Events, Distribution of Components, Error and Exception Handling and Fault Tolerance, Interaction and Presentation and Data Persistence.</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Families of programs and frameworks</li> <li>– Structural descriptions (static view)</li> <li>– Behavior descriptions (dynamic view)</li> <li>– Software Design Strategies and Methods</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

By applying principle #6 “*Quality is the top priority; long term productivity is a natural consequence of high quality*” to the software design (Quality Attributes and Quality Analysis and Evaluation Techniques) for SWEBOK taxonomy and aligned with the IEEE Std 1362-

1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #6 “Quality Is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Quality Attributes</li> <li>– Quality Analysis and Evaluation Techniques.</li> </ul>
<b>Operational Scenario</b>	<p><b>Quality Attributes:</b></p> <ul style="list-style-type: none"> <li>– Quality attributes related to the architecture’s intrinsic qualities (conceptual, integrity, correctness, and completeness, buildability).</li> <li>– The “ilities” (maintainability, portability, testability, traceability),</li> <li>– Various “nesses” (correctness, robustness), including “fitness of purpose.”</li> <li>– Those discernable at run-time (performance, security, availability, functionality, usability),</li> <li>– Those not discernable at run-time (modifiability, portability, reusability, integrability, and testability),</li> </ul> <p><b>Quality Analysis and Evaluation Techniques:</b> The following techniques can be applied to analyze and evaluate the quality of software design artifact such as:</p> <ul style="list-style-type: none"> <li>– Architecture reviews</li> <li>– Design reviews, and inspections</li> <li>– Scenario-based techniques</li> <li>– Requirements tracing</li> <li>– A static analysis technique evaluates a design (example, fault-tree analysis or automated cross-checking).</li> <li>– Software design reviews, static analysis and simulation and prototyping are techniques to evaluate a design.</li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Evaluate the Software Design</li> <li>– Software design review</li> <li>– Identify overall quality of the software design</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Quality of the proposed design</li> <li>– Improved of the evaluating technique of the proposed design.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– Developer impacts</li> </ul>

---

By applying principle #9 principle #9 “To improve design, study previous solutions to similar problems” to the software design for SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

### 1. Principle #8 “To improve design, study previous solutions to similar problems”

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>• Architectural styles (macro architectural patterns);</li> <li>• Design patterns (micro architectural patterns).</li> </ul>
<b>Operational Scenario</b>	The implementation of this FP is in part similar to principle #4 “Implement a disciplined approach and improve it continuously”. This can be found in architectural design step using architectural styles and in detailed design step using design patterns.
<b>Operational Capabilities</b>	
<b>Operational improvements</b>	–
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

### Proposed Operational Guidelines for Software Constructions

#### The Proposed Operational Guidelines (Software Construction for SWEBOK Guide)

The operational guidelines of the 9 FP in the “Software Construction” KA of SWEBOK Guide.

Software Engineering Principles	SWEBOK Guide Software Construction Knowledge Area
1. Principle #1 “Apply and Use Quantitative Measurements in Decision Making”	➤ Construction measurement
2. Principle #2 “Build with and for Reuse”	➤ Reuse

3. Principle #3“Grow System Incrementally”	
4. Principle #4 “Implement A Disciplined Approach and Improve It Continuously”	<ul style="list-style-type: none"> <li>➤ Construction design</li> <li>➤ Coding</li> <li>➤ Construction languages</li> <li>➤ Construction testing</li> <li>➤ Integration</li> </ul>
5. Principle #6“Quality is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”	<ul style="list-style-type: none"> <li>➤ Minimizing complexity</li> <li>➤ Standards in construction</li> <li>➤ Constructing for verification</li> <li>➤ Construction quality</li> <li>➤ Coding</li> <li>➤ Construction testing</li> </ul>
6. Principle #7“Since Change is Inherent to Software, Plan for it and Manage It”	<ul style="list-style-type: none"> <li>➤ Anticipating change.</li> </ul>

By applying principle #1“Apply and use quantitative measurements in decision making” for the software construction (Construction measurement) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

### **Principle #1“Apply and Use Quantitative Measurements in Decision Making”**

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Construction measurement</li> </ul>
<b>Operational Scenario</b>	<p>Several construction activities and artefacts can be measured, these measurements can be useful for purposes of managing construction with the following scenario:</p> <ul style="list-style-type: none"> <li>– Process measurement of the construction to improve the construction process</li> <li>– Software product measured to ensure the quality during construction</li> <li>– Measurement technique for the construction process and product.</li> </ul>
<b>Operational Capabilities</b>	<p>Measure the following artifacts :</p> <ul style="list-style-type: none"> <li>– Code developed</li> <li>– Code modified</li> <li>– Code reused</li> <li>– Code destroyed</li> <li>– Code complexity</li> <li>– Scheduling.</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improve code inspection statistics,</li> <li>– Improve effort</li> </ul>

---

	– Improve fault-fix and fault-find rates
<b>Operational impacts</b>	– Impacts during development
	– Organizational impacts
	– User impacts
	– Developer impacts
	– Buyer impacts

---

By applying principle #2 “*Build with and for reuse*” for the software construction (Reuse ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

### **Principle #2 “*Build with and for Reuse*”**

---

	– Reuse
<b>Operational Concepts</b>	
	In the “Software construction” KA during coding and testing activities, the different tasks related to reuse activity are described as follow;
<b>Operational Scenario</b>	– Select reusable units, databases, test procedures, or test data
	– Evaluate code or test reusability
	– Report reuses information on new code, test procedures, or test data.
<b>Operational Capabilities</b>	– Reusable units,
	– Reusable databases,
	– Reusable test procedures, or test data
	– The evaluation of code or test reusability
	– The reporting of reuse information on new code, test procedures, or test data
<b>Operational improvements</b>	– Improve the reusability of test procedure and data
	– Improve the evaluation of the code or test reusability
<b>Operational impacts</b>	– Impacts during development
	– Organizational impacts
	– Designer impacts
	– Developer impacts

---

By applying principle #4 “*Implement a disciplined approach and improve it continuously*” for the software construction (Construction design, Coding, Construction languages, Construction testing and Integration ) for the SWEBOK taxonomy and aligned with the IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

### **Principle #4 “*Implement A Disciplined Approach and Improve It Continuously*”**

---



---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Construction design</li> <li>– Coding</li> <li>– Construction languages</li> <li>– Construction testing</li> <li>– Integration</li> </ul>
-----------------------------	--

---

**Operational Scenario**      **Construction design**  
 In “Software construction” the design activity is similar as in the “Software design” KA but in the former the design is done on a smaller scale.

**Construction languages**

- Construction languages include all forms of communication by which a human can specify an executable problem solution. There are three types of construction languages. Some of these construction languages includes:
  - **Configuration languages:** the text-based configuration files used in both the Windows and Unix operating systems
  - **Toolkit languages:** they are more complex than configuration languages used to build applications by integrating reusable parts.
  - **Programming languages:** these are the most flexible type of construction languages.
  
- There are three general kinds of notation used for programming languages, namely:
  - **Linguistic:** are distinguished in particular by the use of word-like strings of text to represent complex software constructions, and the combination of such word-like strings into patterns that have a sentence-like syntax
  - **Formal:** heart of most forms of system programming, where accuracy, time behavior, and testability are more important than ease of mapping into natural language. Formal constructions also use precisely defined ways of combining symbols that avoid the ambiguity of many natural language constructions.
  
  - **Visual:** visual entities on a display

**Coding activity**

During the coding activity of the software construction there are numerous techniques that may be applied to write a code that is simple and understandable.

- Naming and code source layout
  - Use of classes, named constant etc
  - Control structures and Handle error conditions
-

- 
- Prevent code-level security breaches
  - Source code organization into statements, classes etc
  - Document code
  - Code tuning
  - Resource usage via use of exclusion mechanisms and discipline in accessing serially reusable resources.

### **Testing activity**

The testing activity in the Construction KA involves two types of testing:

- Unit and
- Integration.

### **Integration activity**

To accomplish the integration task whether related to different routines, components, classes, and subsystems that are constructed during the construction activity.

- “Plan the sequence, in which components will be integrated”
- “Create scaffolding to support interim versions of the software”
- “Determine the degree of testing and quality work performed on components before they are integrated”
- “Determine points in the project at which interim versions of the software are tested”.

---

### **Operational Capabilities**

- Configuration languages, Toolkit languages and Programming languages.
- Techniques that may be applied to write a code that is simple and understandable.
- Test units and software integration.

---

### **Operational improvements**

- Improve the construction design on smaller scale
- Improve the coding activity
- Improve the construction languages in terms of ( linguistic, formal methods and visual trends)
- Construction testing
- Integration the construction closely related to software design and software testing.

---

### **Operational impacts**

- Impacts during development
  - Organizational impacts
  - User impacts
  - Developer impacts
  - Buyer impacts
- 

By applying principle #6 “*Quality is the top priority; long term productivity is a natural consequence of high quality*” for the software construction (Minimizing complexity, Standards in construction, Constructing for verification, Construction quality, Coding and

Construction testing) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #6 “Quality Is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Minimizing complexity</li> <li>– Standards in construction</li> <li>– Constructing for verification</li> <li>– Construction quality</li> <li>– Coding</li> <li>– Construction testing</li> </ul>
-----------------------------	---

---

**Minimizing complexity**

This topic is related to “standards in construction”, “coding” and “construction quality” topics. Minimizing complexity is achieved through many possibilities such as using:

- Standards described in “Standards in Construction”
- Specific techniques described in “Coding” topic.
- Quality techniques described in “Construction Quality” topic.

**Operational Scenario**

**Standards in construction**

- Standards directly affect construction issues include Use of external standards.
- Standards may also be created on an organizational basis at the corporate level or for use on specific projects
- Software Construction KA depends on the use of external standards for construction languages, construction tools, technical interfaces, and interactions between Software Construction and other software engineering.

**Constructing for verification**

The verification activity is important in the software construction phase. Specific tasks that support constructing for verification include the following:

- Follow coding standards to support code reviews, unit testing,
  - Organize the code to support automated testing, and restricted use of complex or hard-to-understand language structures.
  - Building software in such a way that faults can be search out readily by the software engineers writing the software as well as during independent testing and operational activities.
  - Techniques that support constructing for verification include
-

---

following coding standards to support code reviews, unit testing, organizing code to support automated testing, and restricted use of complex or hard-to-understand language structures, among others.

### **Construction Quality**

Construction quality activities are performed on code and on artifacts that are related to code. To write a code of a good quality during software construction, several techniques exist, including: Unit testing and integration testing, test-first development (see also the Software Testing KA), code stepping, use of assertions, debugging, technical reviews (see also the Software Quality KA,), static analysis (IEEE1028).

### **Coding activity and Construction Testing activities**

The related details for these two activities are already described in “Implement a disciplined approach and improve it continuously”

---

<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Minimizing software complexity</li> <li>– Using Standards in the construction</li> <li>– Constructing for verification and testing</li> <li>– Construction quality</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improving software quality</li> <li>– Improving verification method and testing.</li> <li>– Improving the code activity.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– Developer impacts</li> </ul>

---

By applying principle #7 “Since Change is Inherent to Software, Plan for it and Manage It” for the software construction (Anticipating change ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

### **“Principle #7“Since Change is Inherent to Software, Plan for it and Manage It”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>– Anticipating change</li> </ul>
<b>Operational Scenario</b>	<ul style="list-style-type: none"> <li>– “Most software will change over time, and the anticipation of change drives many aspects of software construction. Software is unavoidably part of changing external environments, and changes in those outside</li> </ul>

---

---

environments affect software in diverse ways”.

- Anticipating change is supported by many specific techniques:
  - Communication methods (for example, standards for document formats and contents)
  - Programming languages (for example, language standards for languages like Java and C++)
  - Platforms (for example, programmer interface standards for operating system calls)
  - Tools (for example, diagrammatic standards for notations like UML (Unified Modeling Language))

---

<b>Operational Capabilities</b>	– Implement a disciplined approach
	– Improve it continuously in “coding activity”.

---

<b>Operational improvements</b>	– Construction Communication method
	– Improving platform in a construction
	– Improving programming languages.

---

<b>Operational impacts</b>	– Impacts during development
	– Organizational impacts
	– User impacts
	– Developer impacts
	– Buyer impacts

---

### Proposed Operational Guidelines for Software Testing

#### The Proposed Operational Guidelines (Software Testing for SWEBOK Guide)

The operational guidelines of the 9 FP in the “Software testing” KA of SWEBOK Guide.

Software Engineering Principles	SWEBOK Guide Software Testing Knowledge Area
1. Principle #1 “Apply and Use Quantitative Measurements in Decision Making”	<ul style="list-style-type: none"> <li>– Evaluation of the program under test</li> <li>– Evaluation of the tests performed</li> <li>– Cost/effort estimation and other process measures</li> </ul>
2. Principle #2 “Build with and for Reuse”	– Practical considerations
3. Principle #3 “Grow System Incrementally”	
4. Principle #4 “Implement A Disciplined Approach and Improve It Continuously”	<ul style="list-style-type: none"> <li>– The target of the test</li> <li>– Objectives of testing</li> <li>– Test activities</li> <li>– Test techniques.</li> </ul>
5. Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It”	– Practical considerations

- 
6. Principle #8 “Since Tradeoffs are Inherent to Software Engineering, Make Them Explicit and Document them” – Objectives of testing
- 

By applying principle #1 “Apply and use quantitative measurements in decision making” for the software testing (Evaluation of the program under test, Evaluation of the tests performed and Cost/effort estimation and other process measures) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document

---

**Principle #1 “Apply and Use Quantitative Measurements in Decision Making”**

---

- Operational Concepts**
- Evaluation of the program under test
  - Evaluation of the tests performed
  - Cost/effort estimation and other process measures
- 
- The test-related measures evaluate the program under test based on the observed test outputs, as well as the evaluation of the thoroughness of the test set.
  - **Evaluation of the program under test**

To evaluate a program under test, the following test related measures could be collected:

**Operational Scenario**

**1- Program measurements to aid in planning and designing testing**

To guide testing apply the following measures such as:

- Program size (for example, source lines of code or function points)
  - Program structure (like complexity, frequency with which modules call each other).
  - **Definition of fault types, classification, and statistics**  
To make testing more effective:
    - Define faults types
    - Count the relative fault frequency.  
*More information can be found in the Software Quality KA, topic Defects characterization.*
  - **Measure fault density:** a program under test can be assessed by:
    - Counting the discovered faults
-

- 
- Classifying the discovered faults by their types.
  - Measuring the fault density (the ratio between the number of faults found and the size of the program) for each fault class.

- **Life test, reliability evaluation**

To decide when to stop test:

- Evaluate a product by using a statistical estimate of software reliability (see sub-topic 2.2.5),

- **Reliability growth models**

*“Reliability growth models provide a prediction of reliability based on the failures observed under reliability achievement and evaluation (see sub-topic 2.2.5)”.*

These models are divided into:

- Failure-count
- Time-between failure models.

– **Evaluation of the tests performed**

To evaluate the test performed the following test related measures could be done:

- **Coverage/thoroughness measures**

- Evaluate the thoroughness of the executed tests by choosing the test cases that exercise a set of elements identified in the program or in the specifications.
- Measure dynamically the ratio between covered elements and their total number.

Example:

- 1- Measure the percentage of covered branches in the program flowgraph,
- 2- Measure the functional requirements exercised among those listed in the specifications document.

- **Fault seeding**

Before test insert fault into the program.

Some related measures include:

- The number of artificial faults discovered,
- The number of testing effectiveness ,
- Estimation of the remaining number of genuine faults.

- **Mutation score**

To measure the effectiveness of the executed test set:

- Measure the ratio of killed mutants to the total number of generated mutants.

- **Termination**

To decide when to stop test the following thoroughness measures

---

---

can help, such as:

- Achieved code coverage
- Functional completeness,
- Estimates of fault density
- Estimate operational reliability,
- Cost
- Risks

– **Cost/effort estimation and other process measures**

Measures relative to the control and the improvement of the test process for management purposes include:

- Measure the resources spent on testing,
- Measure the relative fault-finding effectiveness of the various test phases,
- These tests measures cover the following elements:
  1. Number of test cases specified,
  2. Number of test cases executed,
  3. Number of test cases passed,
  4. Number of test cases failed,
- Evaluate test process effectiveness by evaluating:
  1. Test phase reports
  2. Root cause analysis.

---

<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Evaluation of the test criteria</li> <li>– Evaluation of the software</li> <li>– Estimation of the cost</li> <li>– Estimation of the effort</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improve the evaluation testing model</li> <li>– Improve the criteria to evaluate the software</li> <li>– Improve the estimating process for the cost and productivity</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

By applying principle #2 “*Build with and for reuse*” of the software testing (Practical considerations ) for the SWEBOK taxonomy and aligned with the IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.



---

<b>Operational Concepts</b>	– Practical considerations
<b>Operational Scenario</b>	<b>Build with and for reuse:</b> reuse the test material used to test the software. This test material should also be documented so that it can be reused such as: test cases.
<b>Operational Capabilities</b>	– Test the software – Test material
<b>Operational improvements</b>	– Improve the criteria for reusing the test
<b>Operational impacts</b>	– Impacts during development – Organizational impacts – Designer impacts – Developer impacts

---

By applying principle #4 “*Implement a disciplined approach and improve it continuously*” of the software testing (The target of the test, Objectives of testing, Test activities and Test techniques ) for the SWEBOK taxonomy and aligned with the IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

#### **Principle #4 “*Implement A Disciplined Approach and Improve It Continuously*”**

---

<b>Operational Concepts</b>	– The target of the test – Objectives of testing – Test activities – Test techniques.
<b>Operational Scenario</b>	– The test process is composed of several activities from planning to defect tracking. The details related to those activities are illustrated in the test activities as follow: <ul style="list-style-type: none"> <li>• <b>Plan the testing activities</b> <ol style="list-style-type: none"> <li>1. The different steps for one baseline of the software include:</li> <li>2. Coordinate personnel,</li> <li>3. Manage available test facilities and equipment (which may include magnetic media, test plans and procedures),</li> <li>4. Plan for possible undesirable outcomes.</li> <li>5. Estimate the time and effort needed for more than one baseline project.</li> </ol> </li> </ul>

---

---

- **Generate test-cases**

To generate test cases the following should be done:

1. Define the target of the test - see test levels section
2. Define the objective of the test - see test levels section
3. Identify the techniques that are used for testing - see test techniques section
4. Put the control of test cases under the software configuration management
5. For each test case include the expected results.

- **Define the environment test development**

1. Check the compatibility for the testing environment with the software engineering tools.
2. Test environment should facilitate development and control of test cases, logging and recovery of expected results, scripts, and other testing materials.

- **Execute the tests**

During the execution of tests everything done should be:

1. Performed and documented clearly enough that another person could replicate the results.
2. Performed in accordance with documented procedures using a clearly defined version of the software under test.

- **Evaluate the test results**

The results of test are determined by success or failure. When a failure is identified before it can be removed:

1. Analyze and debug to isolate, identify, and describe a failure.
2. Evaluate the test result with a formal review board if they are important.

- **Report problems related to testing activities/ Test log**

Below a list of various information that can be reported into a test log or a problem-reporting system

1. When a test was conducted,
2. Who performed the test,
3. What software configuration was the basis for testing,
4. And other relevant identification information.
5. Record incorrect test results in a problem-reporting system,
6. Document anomalies not classified as faults

- Test reports are also an input to the change management requests process (see the Software Configuration Management KA, subarea 3: Software Configuration Control).
-

---

- **Track defect for later analysis**

Analyze the defects to determine:

1. When they were introduced into the software,
2. What kind of error caused them to be created (poorly defined requirements, incorrect variable declaration, memory leak, programming syntax error, for example),
3. When they could have been first observed in the software.

Defect-tracking information is used to determine:

1. What aspects of software engineering need improvement
2. How effective previous analyses and testing have been.

- **Test Levels**

The test level defines the target of the test and the objectives of the test. The target of the test is divided into three levels; unit, integration and system testing. Figure 6.19 illustrates the model related to the test levels.

- **Test Techniques**

To test software various techniques are defined. Some of these techniques includes specification based techniques, code based techniques and techniques based on the nature of the application.

---

<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– The target of the test</li> <li>– Objectives of testing</li> <li>– Test activities</li> <li>– Test techniques</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improve the target of the test</li> <li>– Improve Objectives of testing</li> <li>– Improve Test activities</li> <li>– Improve Test techniques</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

By applying principle #7 “Since Change is Inherent to Software, Plan for it and Manage It” for the software testing (Practical considerations ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

<b>Operational Concepts</b>	– Practical considerations
<b>Operational Scenario</b>	“Test materials must be under the control of software configuration management, so that changes to software requirements or design can be reflected in changes to the scope of the tests conducted”
<b>Operational Capabilities</b>	– control of software configuration management – software requirements or design – scope of the tests conducted
<b>Operational improvements</b>	– improve the software management – improve the software configuration control activities
<b>Operational impacts</b>	– Impacts during development – Organizational impacts – User impacts – Developer impacts – Buyer impacts

---

By applying Principle #8 “Since Tradeoffs are Inherent to Software Engineering, Make Them Explicit and Document them” of the software testing ( ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #8 “Since Tradeoffs are Inherent to Software Engineering, Make Them Explicit and Document them”**

---

<b>Operational Concepts</b>	– Objectives of testing
<b>Operational Scenario</b>	– <b>Regression testing:</b> In regression testing a trade-off must be made between: <ul style="list-style-type: none"> <li>• The assurance given by regression testing every time a change is made</li> <li>• The resources required to do that.</li> </ul> <p>Also as mentioned in the Introduction for this KA, a trade off must be made to choose a set of test cases between:</p> <ul style="list-style-type: none"> <li>• The limited resources and schedules</li> <li>• Unlimited test requirements.</li> </ul>
<b>Operational Capabilities</b>	– test requirements – resources requirements

---

<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>- Improve resource requirements</li> <li>- Improve test requirements</li> <li>- Improve the objective of the testing frequently</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>- Impacts during development</li> <li>- Organizational impacts</li> <li>- Developer impacts</li> </ul>

**Proposed Operational Guidelines for Software Maintenance**

**The Proposed Operational Guidelines  
(Software Maintenance for SWEBOK Guide)**

The operational guidelines of the 9 FP in the “Software maintenance” KA of SWEBOK Guide.

<b>Software Engineering Principles</b>	<b>SWEBOK Guide Software Maintenance Knowledge Area</b>
1. Principle #1 “Apply and Use Quantitative Measurements in Decision Making”	<ul style="list-style-type: none"> <li>- Maintenance Cost Estimation.</li> <li>- Software Maintenance Measurement</li> </ul>
2. Principle #4 “Implement A Disciplined Approach and Improve It Continuously”	<ul style="list-style-type: none"> <li>- Maintenance processes</li> <li>- Maintenance activities</li> </ul>
3. Principle #6 “Quality is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”	<ul style="list-style-type: none"> <li>- Maintenance Activities</li> </ul>
4. Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It”	<ul style="list-style-type: none"> <li>- Impact analysis</li> <li>- Software configuration management</li> </ul>

By applying principle #1 “Apply and use quantitative measurements in decision making” for the software Maintenance (Maintenance Cost Estimation and Software Maintenance Measurement ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

**Principle #1 “Apply and Use Quantitative Measurements in Decision Making”**

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>- Maintenance Cost Estimation.</li> <li>- Software Maintenance Measurement</li> </ul>
<b>Operational</b>	<ul style="list-style-type: none"> <li>- <b>Maintenance Cost Estimation</b> The maintenance cost estimation could be done for planning purposes. To estimate resources for software maintenance according to ISO/IEC14764 apply the</li> </ul>

<b>Scenario</b>	<p>following approaches:</p> <p>1-Parametric models</p> <p>2-Experience</p> <ul style="list-style-type: none"> <li>➤ Expert judgment (for example Delphi technique)</li> <li>➤ Analogies</li> <li>➤ A work breakdown structure</li> <li>➤ Combine empirical data and experience.</li> </ul> <p>3-Combine both approaches</p>
	<ul style="list-style-type: none"> <li>• <b>Software Maintenance Measurement</b> <ul style="list-style-type: none"> <li>• Measures common to all endeavors: The software engineering Institute (SEI) has identified the following measures that will be useful for the maintainer: size, effort, schedule and quality.</li> <li>• Internal benchmarking techniques: The maintainers determine which of the following specific measures: analyzability, changeability, stability and testability fit for the organization.</li> </ul> </li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>- Maintenance Cost</li> <li>- Software Measurement of the Maintenance</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>- Improve criteria for estimating cost and productivity of the maintenance.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>- Impacts during development</li> <li>- Organizational impacts</li> <li>- User impacts</li> <li>- Developer impacts</li> <li>- Buyer impacts</li> </ul>

By applying principle #4 “*Implement a disciplined approach and improve it continuously*” of the software Maintenance ( ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #4 “*Implement A Disciplined Approach and Improve It Continuously*”**

---

	<ul style="list-style-type: none"> <li>- Maintenance processes</li> <li>- Maintenance activities</li> </ul>
<b>Operational Concepts</b>	
<b>Operational Scenario</b>	<ul style="list-style-type: none"> <li>- <b>Maintenance processes:</b> The Maintenance Process subarea provides references and standards used to implement the software maintenance process.</li> </ul>

- 
- Standard for Software Maintenance (IEEE1219)
  - ISO/IEC 14764 [ISO14764-99]

– **Maintenance Activities:**

Maintenance activities are composed of the same activities that are in the software development for instance: analysis, design, coding, testing and documentation. There are some activities that are unique to software maintenance and other supporting activities.

**1-Unique activities :** The unique activities for “Software maintenance” are described as follow:

- Transition: Transfer software from the developer to the maintainer
- Modification request acceptance/rejection
- Modification request and problem report help desk
- Impact analysis
- Software support
- Service level agreements (SLAs) and specialized (domain-specific): Maintenance contracts which are the responsibility of the maintainers

**2-Supporting activities:** Below a list of activities that support maintenance, such as:

- Software maintenance planning,
- Software configuration management,
- Software quality.

**2.1 Maintenance planning activity:** There are four perspectives to consider for maintenance activities as follow:

**The individual (request level)**

- Planning is carried out during the impact analysis.
  - **The release/version planning activity (software level)**
    - Collect the dates of availability of individual requests
    - Agree with users on the content of subsequent releases/versions
    - Identify potential conflicts and develop alternatives
    - Assess the risk of a given release and develop a back-out plan in case problems should arise
    - Inform all the stakeholders
-

- 
- **Maintenance planning (transition level)**
    - Estimates resources
    - Include those resources in the developers' project planning budgets.
    - Decide to develop a new system
    - Consider quality objectives (IEEE1061-98).
    - Develop a concept document,
    - Develop a maintenance plan.
  - Prepare the concept document for maintenance [ISO14764-99:s7.2] that addresses:
    - The scope of the software maintenance
    - Adaptation of the software maintenance process
    - Identification of the software maintenance organization
    - An estimate of software maintenance costs
  - Prepare the maintenance plan during software development, and specify:
    - How users will request software modifications
    - How users will report problems.
  - **Business planning (organizational level)**
    - Conduct business planning activities (budgetary, financial, and human resources).

**2.2 Software configuration management:** Software configuration management procedures should: Verify, validate and audit every step essential to identify, authorize, implement and release the software product.

---

<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Process of the Maintenance</li> <li>– Activities of the Maintenance</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improve the maintenance processes</li> <li>– Improve maintenance activities.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

By applying principle #6 “*Quality is the top priority; long term productivity is a natural consequence of high quality*” for the software Maintenance ( ) for the SWEBOK taxonomy



and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**Principle #6 “Quality Is The Top Priority; Long Term Productivity Is A Natural Consequence of High Quality”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>- Maintenance Activities</li> </ul>
<b>Operational Scenario</b>	<ul style="list-style-type: none"> <li>- Software Quality represents one of the supporting activities of “Software maintenance” To achieve the appropriate level of quality the different tasks related to software quality need to be completed as follow:             <ul style="list-style-type: none"> <li>• Plan quality</li> <li>• Plan processes implemented to support the maintenance process.</li> <li>• Select the activities and techniques for Software Quality Assurance (SQA), V&amp;V, reviews, and audits</li> <li>• A recommendation: The maintainer should adapt the software development processes, techniques and deliverables, for instance:                 <ul style="list-style-type: none"> <li>➤ Testing documentation</li> <li>➤ Test results.</li> </ul> </li> </ul> </li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>- Software quality</li> <li>- Maintenance of the activities steps.</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>- Improve software quality for maintenance requirements</li> <li>- Improve the maintenance of the activities steps.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>- Impacts during development</li> <li>- Organizational impacts</li> <li>- Developer impacts</li> </ul>

By applying principle #7 “Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It” for the software Maintenance ( ) for the SWEBOK taxonomy and aligned with IEEE Std 1362-1998 concepts of operational document, the current situation of the SWEBOK guide require to improve the concepts of the operational document.

---

**“Principle #7 “Since Change is Inherent to Software, Plan for it and Manage It”**

---

<b>Operational Concepts</b>	<ul style="list-style-type: none"> <li>- Impact analysis</li> <li>- Software configuration management</li> </ul>
<b>Operational Scenario</b>	<ul style="list-style-type: none"> <li>- Here are presented a few steps on how to perform impact analysis:             <ul style="list-style-type: none"> <li>• Determine the risk of making a change</li> </ul> </li> </ul>

---

	<ul style="list-style-type: none"> <li>• Analyze a change request, modification request (MR) or a problem report (PR)</li> <li>• Translate a change request into software terms.</li> <li>• Perform impact analysis after a change request enters the software configuration management process.</li> </ul>
	<ul style="list-style-type: none"> <li>– <b>Software configuration management</b> <ul style="list-style-type: none"> <li>• Control the changes made to a software product.</li> <li>• Establish the control by implementing and enforcing an approved software configuration management process.</li> </ul> </li> </ul>
<b>Operational Capabilities</b>	<ul style="list-style-type: none"> <li>– Impact analysis</li> <li>– Software configuration management</li> </ul>
<b>Operational improvements</b>	<ul style="list-style-type: none"> <li>– Improve the analysis impacts</li> <li>– Improve the configuration management cycle.</li> </ul>
<b>Operational impacts</b>	<ul style="list-style-type: none"> <li>– Impacts during development</li> <li>– Organizational impacts</li> <li>– User impacts</li> <li>– Developer impacts</li> <li>– Buyer impacts</li> </ul>

---

## ANNEX VI

### MAPPING SWEBOK KA WITH VINCENTI SIX CATEGORIES AND THE NINE FUNDAMENTAL PRINCIPLES (9 FP'S).

This Annex presents the mapping between Vincenti six categories of engineering knowledge, the nine fundamental principles and the following 3 KAs in the SWEBOK Guide

- Requirements
- Design
- Construction

**F-1:** Mapping scope definitions of the “Software requirements” KA in SWEBOK taxonomy with Vincenti six categories and the nine fundamental principles ( 9 FP’s).

Software requirements subareas	Software requirements topic	Scope	Engineering FP	Vincenti’s six Categories
<b>Software requirements fundamentals</b>	<b>Definition of a software requirement</b>	Is concerned with problems to be addressed by software, it is a property which must be exhibited by software developed or adapted to solve a particular problem.		# 1
	<b>Product and process requirements</b>	Product parameters are requirements on software to be developed and A process parameter is essentially a constraint on the development of the software		#1
	<b>Functional &amp; nonfunctional requirement</b>	Describe the functions that the software is to execute and Non functional requirements are a constraints or quality requirements.		#2
	<b>Emergent properties</b>	requirements which cannot be addressed by a single component, but which depend for their satisfaction on how all the software components interoperate		#2
	<b>Quantifiable requirements</b>	Is to state as clearly and as unambiguously as possible quantitatively the software requirements.	#1	
	<b>System requirements and software requirements</b>	System requirements are the requirements for the system as a whole. Software requirements are derived from system requirements.		#2
	<b>Requirements Process</b>	<b>Process models</b>	a process initiated at the beginning of a project and continuing to be refined throughout the life cycle, also is concerned with how the activities of elicitation, analysis, specification, and validation are configured for different types of projects and constraints	#4
<b>Process actors</b>		There are often many people involved besides the requirements specialist, each of whom has a stake in the software.	#4	#6
<b>Process support and management</b>		To make the link between the process activities and the Issues of cost, human resources, training, and tools.	#4	#6

	<b>Process quality and improvement</b>	Requirements process coverage by process improvement standards and models, Requirements process measures and benchmarking and Improvement planning and implementation	#4, #6	#2
<b>Requirements elicitation</b>	<b>Requirements sources</b>	Is concerned with where software requirements come from and how the software engineer can collect them. For example Goals, Domain knowledge, Stakeholders, The operational environment and The organizational environment	#3, #4, #5	#5
	<b>Elicitations techniques</b>	It comes after requirement sources the requirement technique includes Interviews, Scenarios, Prototypes, Facilitated meetings and Observation.	#3, #4, #5	#1
<b>Requirements analysis</b>	<b>Requirements classification</b>	Classified the requirements whether Functional and non functional requirements and Whether the requirement is on the product or the process, and emergent prosperities.	#3, #4, #5	#2
	<b>Conceptual modeling</b>	Conceptual models comprise models of entities from the problem domain configured to reflect their real-world relationships and dependencies such as The nature of the problem, The expertise of the software engineer, process requirements of the customer and availability of methods and tools	#2, #3, #4, #5	#3
	<b>Architectural design and requirements allocation</b>	Architectural design is the point at which the requirements process overlaps with software or systems design and illustrates how impossible it is to cleanly decouple the two tasks. Allocation is important to permit detailed analysis of requirements	#2, #3, #4	#1
	<b>Requirement negotiation</b>	Conflict resolution.” This concerns resolving problems with requirements where conflicts occur between two stakeholders requiring mutually incompatible features, between requirements and resources, or between functional and non-functional requirements,	#3, #4, #5, #8	#5
<b>Requirements specification</b>	<b>System definition document</b>	known as the user requirements document or concept of operations, includes representatives of the system users/customers and conceptual models designed to illustrate the system context, usage scenarios and the principal domain entities, as well as data, information, and workflows	#3, #4	#2
	<b>Systems requirement specification</b>	often separate the description of system requirements from the description of software requirements	#3, #4	#2
	<b>Software requirement specification</b>	Permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Software requirements specification provides an informed basis for transferring a software product to new users or new machines.	#3, #4	#2

		Finally, it can provide a basis for software enhancement also often written in natural language		
<b>Requirements validation</b>	<b>Requirement reviews</b>	Validation is by inspection or reviews of the requirements document(s). A group of reviewers is assigned a brief to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice.	#3, #4, #6	#6
	<b>Prototyping</b>	Prototyping is commonly a means for validating the software engineer's interpretation of the software requirements, as well as for eliciting new requirements	#3, #4, #6	#6
	<b>Model validation</b>	to validate the quality of the models developed during analysis	#3, #4, #6	
	<b>Acceptance test</b>	to validate the finished product satisfies	#2,#3,#4, #6	#6
<b>Practical consideration</b>	<b>Iterative nature of the requirement process</b>	Requirements typically iterate towards a level of quality and detail which is sufficient to permit design and procurement decisions to be made.	#7	#5
	<b>Change management</b>	the procedures that need to be in place, and the analysis that should be applied to proposed changes	#7	#6
	<b>Requirement attributes</b>	Consist not only of a specification of what is required, but also of ancillary information which helps manage and interpret the requirements. This should include the various classification dimensions of the requirement	#7	#2
	<b>Requirements tracing</b>	is concerned with recovering the source of requirements and predicting the effects of requirements,	#7	#5
	<b>Measuring requirement</b>	is typically useful to have some concept of the "volume" of the requirements for a particular software product to evaluating the "size" of a change in requirements and estimating the cost of a development or maintenance task	#1	#5

**F-2:** Mapping scope definitions of the "Software design" KA in SWEBOK taxonomy with Vincenti six categories and the nine fundamental principles ( 9 FP's).

<b>Software design subareas</b>	<b>Software design topic</b>	<b>Scope</b>	<b>Engineering candidate FP</b>	<b>Vincenti's six Categories</b>
	<b>General design concepts</b>	A form of problem solving. A number of other notions and concepts are also of interest in understanding design in its general sense: goals, constraints, alternatives, representations, and solutions.		#1
	<b>The context of software design</b>	the role of software design throughout the life cycle		#1
<b>Software</b>	<b>The software design process</b>	generally considered a two-step process: <b>Architectural design</b> Architectural design describes how software is decomposed and organized into	#4	#1

<b>Design fundamentals</b>	components software architecture	<b>Detailed design</b> Detailed design describes the specific behaviour of these components.	#4	#1
	<b>Enabling techniques</b>	Software design principles, also called <i>enabling techniques</i> . The enabling techniques are the following: <i>Abstraction</i> , Coupling and cohesion, Decomposition and modularization, Encapsulation/information hiding, Separation of interface and implementation, Sufficiency, completeness and primitiveness	#4	#1
<b>Keys issues in Software Design</b>	<b>Concurrency</b>	How to decompose the software into processes, tasks, and threads and deal with related efficiency, atomicity, synchronization, and scheduling issues	#4	#1, #2, #3
	<b>Control and handling of events</b>	How to organize data and control flow, how to handle reactive and temporal events through various mechanisms such as implicit invocation and call-backs.	#4	#1, #2, #3
	<b>Distribution of components</b>	How to distribute the software across the hardware, how the components communicate, how middleware can be used to deal with heterogeneous software	#2, #4	#1, #2, #3
	<b>Error and exception handling and fault tolerance</b>	How to prevent and tolerate faults and deal with exceptional conditions.	#4	#1, #2, #3
	<b>Interaction and presentation</b>	How to structure and organize the interactions with users and the presentation of information (for example, separation of presentation and business logic using the Model-View-Controller approach)	#4	#1, #2, #3
	<b>Data persistence</b>	How long-lived data are to be handled	#4	#1, #2, #3
	<b>Software Structure and Architecture</b>	<b>Architectural Structures and viewpoints</b>	"A view represents a partial aspect of a software architecture that shows specific properties of a software system"	#4
<b>Architectural styles (macro architectural patterns)</b>		Number of major architectural styles. <b>General structure</b> (for example, layers, pipes, and filters, blackboard) <b>Distributed systems</b> (for example, client-server, three tiers, broker) <b>Interactive systems</b> (for example, Model-View-Controller, Presentation-Abstraction-Control) <b>Adaptable systems</b> (for example, micro-kernel, reflection) <b>Others</b> (for example, batch, interpreters, process control, rule-based). design patterns can be used to describe	#2, #4, #9	#1, #6

	<b>Design patterns (macro architectural patterns)</b>	<p>details at a lower, more local level their microarchitecture</p> <p><b>Creational patterns</b> (for example, builder, factory, prototype, and singleton)</p> <p><b>Structural patterns</b> (for example, adapter, bridge, composite, decorator, façade, flyweight, and proxy)</p> <p><b>Behavioural patterns</b> (for example, command, interpreter, iterate, mediator, memento, observer, state, strategy, template, visitor)</p>	#2, #4, #9	#5	
	<b>Families of programs and frameworks</b>	Families allow the reuse of software designs and components	#4	#1	
	<b>Software Design Quality Analysis and Evaluation</b>	<b>Quality attributes</b>	<p>“ilities”- (maintainability, portability, testability, traceability)</p> <p>“nesses” (correctness, robustness), including “fitness of purpose.”</p> <p>at run-time (performance, security, availability, functionality, usability), those not discernable at run-time (modifiability, portability, reusability, integrability, and testability), and those related to the architecture’s intrinsic qualities (conceptual integrity, correctness, and completeness, buildability)</p>	#6	#2
		<b>Quality analysis and evaluation techniques</b>	<p>Software design reviews</p> <p>Static analysis</p> <p>Simulation and prototyping</p>	#6	#3, #5, #6
<b>Measures</b>		<p>Measures can be used to assess or to quantitatively estimate various aspects of a software design’s size, structure, or quality</p> <p>Function-oriented (structured) design measures</p> <p>Object-oriented design measures</p>	#1	#2	
<b>Software Design Notations</b>	<b>Structural descriptions (static view)</b>	describe and represent the structural aspects of a software design such as (Architecture description languages, Class and object diagrams, Component diagrams, Class responsibility collaborator cards, Deployment diagrams, Entity-relationship diagrams, etc)	#4	#3, #6	
	<b>Behavior descriptions (dynamic view)</b>	used to describe the dynamic behavior of software and components such as (Activity diagrams, Collaboration diagrams, Data flow diagrams, Decision tables and diagrams, Flowcharts and structured flowcharts, State transition and state chart diagrams, etc)	#4	#3, #6	
	<b>General strategies</b>	<p>top-down vs. bottom-up strategies</p> <p>data abstraction and information hiding</p> <p>use of heuristics</p> <p>use of patterns and pattern languages</p> <p>use of an iterative and incremental approach</p>	#4	#2, #5, #6	
	<b>Function-</b>	Used after structured analysis, thus			

<b>Software Design Strategies and Methods</b>	<b>oriented (structured design)</b>	producing, among other things, data flow diagrams and associated process descriptions	#4	#2, #5, #6
	<b>Object-oriented design</b>	Software design methods based on objects have been proposed. The field has evolved from the early object based design of the mid-1980s (noun = object; verb = method; adjective = attribute)	#4	#2, #5, #6
	<b>Data-structured centered design</b>	The software engineer first describes the input and output data structures (using Jackson's structure diagrams, for instance)	#4	#2, #5, #6
	<b>Component-based design</b>	A software component is an independent unit, having well defined interfaces and dependencies that can be composed and deployed independently. Component-based design addresses issues related to providing, developing, and integrating such components in order to improve reuse.	#2, #4	#2, #5, #6
	<b>Other methods</b>	Other interesting but less mainstream approaches also exist: formal and rigorous methods	#4	#2, #5, #6

**F-3:** Mapping scope definitions of the “Software construction” KA in SWEBOK taxonomy with Vincenti six categories and the nine fundamental principles ( 9 FP’s).

Software construction subareas	Software construction topic	Scope	Engineering candidate FP	Vincenti's six Categories
<b>Software Construction Fundamentals</b>	<b>Minimizing complexity</b>	Reduced complexity is achieved through emphasizing the creation of code that is simple and readable rather than clever. Minimizing complexity is accomplished through making use of standards,	#6	#1
	<b>Anticipating change</b>	Anticipating change is supported by many specific techniques: Communication methods (for example, standards for document formats and contents) Programming languages (for example, language standards for languages like Java and C++) Platforms (for example, programmer interface standards for operating system calls) Tools (for example, diagrammatic standards for notations like UML (Unified Modeling Language))	#7	#2
	<b>Constructing for verification</b>	Constructing for verification means building software in such a way that faults can be ferreted out readily by the software engineers writing the software, as well as during independent testing	#6	#5, #6



		operational activities. Specific techniques that support constructing for verification include		
		following coding standards to support code reviews, unit testing, organizing code to support automated testing, and restricted use of complex or hard-to-understand language structures, among others.		
	<b>Standards in construction</b>	<p>Include</p> <p>Use of external standards: for construction languages, construction tools, technical interfaces, and interactions between Software Construction and other KAs.</p> <p>Use of internal standards: support coordination of group activities, minimizing complexity, anticipating change, and constructing for verification.</p>	#6	#1, #2, #3, #6
<b>Managing Construction</b>	<b>Construction models</b>	<p>Some models are more linear from the construction point of view such as the waterfall and staged-delivery life cycle models.</p> <p>Other models are more iterative, such as evolutionary prototyping, Extreme Programming, and Scrum.</p>	#4	#1, #3, #6
	<b>Construction planning</b>	<p>Affects the project's ability to reduce complexity, anticipate change, and construct for verification. Each</p> <p>defines the order in which components are created and integrated, the software quality management processes, the allocation of task assignments to specific software engineers</p>	#4	#1, #3, #6
	<b>Construction measurement</b>	including code developed, code modified, code reused, code destroyed, code complexity, code inspection statistics, fault-fix and fault-find rates, effort, and scheduling	#1	#3
<b>Practical Considerations</b>	<b>Construction design</b>	building a physical structure must make small-scale modifications to account for unanticipated gaps in the builder's plans.	#4	#1, #3
	<b>Construction languages</b>	<p>Include all forms of communication by which a human can specify an executable problem solution to a computer.</p> <p>The simplest type of construction language is a configuration language,</p> <p>Toolkit languages are used to build applications out of toolkits (integrated sets of application-specific reusable parts), and are more complex than configuration languages.</p> <p>Programming languages are the most flexible type of construction languages. They also contain the least amount of information about specific application areas and development processes, and so require the most training and skill to use effectively.</p>	#4	#6

<b>Coding</b>	<p>The following considerations apply to the software construction coding activity:</p> <ul style="list-style-type: none"> <li>- source code,</li> <li>- Use of classes, enumerated types, variables, named constants, and other similar entities</li> <li>- Use of control structures</li> <li>- Handling of error conditions</li> </ul>	#4	#6
	<ul style="list-style-type: none"> <li>- Prevention of code-level security breaches</li> <li>- Resource usage via use of exclusion mechanisms and discipline in accessing serially reusable resources</li> <li>- Source code organization (into statements, routines, classes, packages, or other structures)</li> <li>- Code documentation</li> <li>- Code tuning</li> </ul>		
<b>Construction testing</b>	<p>two forms of testing, which are often performed by the software engineer who wrote the code:</p> <ul style="list-style-type: none"> <li>-Unit testing</li> <li>-Integration testing</li> </ul> <p>The purpose of construction testing is to reduce the gap between the time at which faults are inserted into the code and the time those faults are detected.</p>	#4, #6	#6
<b>Reuse</b>	<p>The tasks related to reuse in software construction during coding and testing are:</p> <ul style="list-style-type: none"> <li>-The selection of the reusable units, databases, test procedures, or test data</li> <li>-The evaluation of code or test reusability</li> <li>-The reporting of reuse information on new code, test procedures, or test data</li> </ul>	#2	#5
<b>Construction quality</b>	<p>The primary techniques used for construction include:</p> <ul style="list-style-type: none"> <li>-Unit testing and integration testing</li> <li>-Test-first development</li> <li>-Code stepping</li> <li>-Use of assertions</li> <li>-Debugging</li> <li>-Technical reviews</li> </ul>	#6	#5, #6
<b>Integration</b>	<p>include planning the sequence in which components will be integrated, creating scaffolding to support interim versions of the software, determining the degree of testing and quality work performed on components before they are integrated, and determining points in the project at which interim versions of the software are tested.</p>	#4	#1

## ANNEX VII

### WORKSHOP IN INTERNATIONAL CONFERENCE ON ENGINEERING EDUCATION – ICEE 2007-03-20 COIMBRA (PORTUGAL) 2007

The Engineering Foundations of Software Engineering  
Workshop Program – Sunday Sept. 2, 2007  
Draft – May 1 2007

International Conference on Engineering Education – ICEE 2007-03-20  
Coimbra (Portugal) 2007 – <http://icee2007.dei.uc.pt/program.htm>

Topics	Speaker- Discussion Leader	Ref. material	Discussio n Topics
<b>RELATED WORK</b>			
<b>1</b>	Delphi Studies on Fundamental Principles of Software Engineering	Robert Dupuis	JSS paper
<b>2</b>	The literature on software engineering principles + identification of criteria for selecting candidate fundamental principles (from an inventory of 313 principles proposed in the literature to a core set of 34 candidate fundamental principles)	Normand Séguin	In 'Revue Génie logiciel' (to be translated)  <b>-Criteria -Outcome -Coverage</b>
<b>WORK IN PROGRESS</b>			
<b>A- Fundamentals of Software Engineering</b>			
<b>3</b>	Vincenti engineering knowledge types and their mapping to software engineering concepts	Alain Abran	Upgrade paper
<b>4</b>	The core set of fundamental principles selected using Vincenti and ACM-IEEE 2000 curriculum criteria	Kenza Meridji	Draft paper
<b>5</b>	Overview of the Software Engineering Body of Knowledge – SWEBOK	Robert Dupuis	SWEBOK Guide
<b>6</b>	Coverage of SE fundamental engineering principles in the SWEBOK & software engineering education	Kenza Meridji	Slides
<b>7</b>	1st Group Discussion	To be determined	How to cover fundamental principles in the teaching of software engineering?
<b>8</b>	Software engineering ontology for project management		Fran Ruiz-Bertol Javier Dolado
<b>9</b>	2 <sup>nd</sup> Group Discussion	To be determined	Coverage verification for teaching software

project management			
<b>B- IEEE-Computer Society - Professional Practice Committee (PPC) project: Principles of Practice</b>			
10	Context and Needs of PPC	Robert Dupuis	IEEE-CS PPC related slides
11	3 <sup>rd</sup> Group Discussion for software engineering teaching on principles of practice	To be determined	<ul style="list-style-type: none"> <li>- Identification of gaps</li> <li>- Identification of sources to cover gaps</li> </ul>
12	4 <sup>th</sup> Group discussion on related process issues for principles of practices	To be determined	<ul style="list-style-type: none"> <li>- How to recognize obsolescence?</li> <li>- Update mechanisms?</li> <li>- How to describe levels of abstraction?</li> <li>- How to build consensus (wiki, etc.)?</li> <li>- Next steps?</li> </ul>

### Result of workshop on engineering foundations of software engineering in the international conference on engineering education ICEE 2007

#### First result on the mapping of the candidate FP to Vincenti engineering criteria

Fundamental principles	Vincenti engineering Criteria with corresponding direct (D) or indirect (I) mapping
5: Define software artifacts rigorously	Problem: D, Criteria:D
20: Produce software in a stepwise fashion	Testing: D
25: Strive to have a peer, rather than a customer find a defect	Problem: D/I
Fundamental principles	Reason for rejection
5: Define software artifacts rigorously	1: Is not related to problem directly for the same reason as in 3 Is not related to criteria directly. This criteria is related to the identification of concepts.
20; Produce software in a stepwise fashion	2: Testing has no think to do with the actual FP
25: Strive to have a peer, rather than a customer find a defect	3: Is not related to problem, "recognition of problem" which is recognized at the domain

	level
<b>Fundamental principles</b>	<b>IEEE &amp; ACM engineering criteria with corresponding direct (D) or indirect (I) mapping</b>
14: Grow systems incrementally	Disciplined process: (D) instead of (I)
28: Use better and fewer people	Engineer's role : I
<b>Fundamental principle</b>	<b>Reason for rejection</b>
14: Grow systems incrementally	1: Is not related to discipline process directly. It can be done in any order.
28: Use better and fewer people	2: Engineer's can have many roles

## **Second result on the mapping of the candidate FP to IEEE & ACM engineering criteria**

### **List of fundamental principles of software engineering**

The participants agreed on the list of 9FP.

### **Hierarchy of candidate FP**

The participants agreed on the hierarchy of candidate FP.

## BIBLIOGRAPHY

- Abran A. et al. (2004). The search for software engineering principles: An overview of results. PRinciples of Software Engineering, Buenos Aires (Argentina).
- Basili V. et al. (1986). Experimentation in Software Engineering. IEEE Transactions on Software Engineering.
- Baskerville R. et al. (2003). internet-speed software development different? IEEE Computer Society.
- Boehm B.W. (1983). "Seven Basics Principles of Software Engineering." Journal of Systems and Software 3(no 1): 366-371.
- Bourque P. and Dupuis R. (1997). Fundamental Principles of Software Engineering ,. in Third International Symposium and Forum on Software Engineering Standards ,, Walnut Creek, CA ,.
- Bourque P. et al. (2002). "Fundamental principles of software engineering – a journey." Journal of Systems and Software 62: 59-70.
- Buschmann F. et al. (1996). Pattern Oriented Software Architecture. England. England.
- Davis A.M. (1994). "Fifteen principles of software engineering." Software, IEEE 11(6): 94-96, 101.
- Davis A.M. (1995). 201 Principles of Software Development. New-York, McGraw-Hill.
- Dupuis R. et al. (1997). Principes Fondamentaux du génie logiciel : Une étude Delphi dans le génie logiciel et ses applications. . In Dixièmes journées internationales «Le génie logiciel et ses applications» (GL97). Paris (France): 3-5. .
- Dupuis R. et al. (1999). Progress Report on the Fundamental Principles of Software Engineering. ISESS'99 , , IEEE Computer Society ,. in 4th International Software Engineering Standards Symposium. Curitiba, Brazil.
- Ghezzi C. et al. (2003). Fundamentals of Software Engineering. New-Jersey, Prentice Hall.
- Guay B. (2004). Comparative analysis between the SWEBOK Guide and the fundamental principles of software engineering. Software engineering. Montreal, Ecole de technologie supérieure.
- IEEE1028-97 IEEE Standard for Software Reviews, IEEE.

- IEEE1219-98 "IEEE Std 1219-1998, IEEE Standard for Software Maintenance, IEEE, 1998."
- IEEE 610.12-1990 (2001). IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers R. D. Pierre Bourque, Alain Abran, James W. Moore, and Leonard Tripp SWEBOK Guide.: 84
- IEEE and ACM (2004). Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. A. V. o. t. C. C. Series.
- IEEE STD 1362-1998 IEEE Guide for Information Technology System Definition Concept of Operations (ConOps) Document, IEEE computer society.
- ISO9126-01 "ISO/IEC 9126-1:2001, Software Engineering-Product Quality-Part 1: Quality Model, ISO and IEC, 2001."
- ISO14764-99 ISO/IEC 14764-1999, Software Engineering-Software Maintenance, ISO and IEC, 1999. ISO/IEC 14764-1999,.
- ISO-12207 (1995 ). "Information Technology Software Life Cycle Processes."
- ISO-TR\_19759 (2004). SWEBOK Guide:Software Engineering Body of Knowledge
- Jabir and Moore J.W. (1998). A Search for Fundamental Principles of Software Engineering - Report of a Workshop conducted at the Forum on Software Engineering Standards Issues, Computer Standards and Interfaces, vol. 19, pp. 155-160, 1998. Montréal, Quebec, Canada, 21-25 October 1996,.
- Kotonya G. and Sommerville I. (2000). Requirements Engineering: Processes and Techniques, John Wiley & Sons.
- IEEE-Std 610.12 (1990). "IEEE Standard Glossary of Software Engineering Terminology."
- Robert F. et al. (2002). A technical review of the software construction knowledge area in the SWEBOK Guide. STEP.
- Séguin, N. and A. Abran (2007). "Inventaire des principes du génie logiciel." Revue Génie Logiciel: 45-51.
- Séguin N. (2006). Inventaire, Analyse et Consolidation des Principes Fondamentaux du Génie logiciel Montréal, Université du Québec à Montréal
- Vincenti W. G. (1990). What engineers know and how they know it. Baltimore, London, The Johns Hopkins University Press.
- Wang Y. (2007). Software Engineering Foundations: A Software Science Perspective.

Wieggers K.E. (1996). Creating a software Engineering culture, New-York, Dorset House Publishing.