

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
Malik QASAMEH

AUDITING FOR ISO 9001 REQUIREMENTS IN THE CONTEXT OF AGILE
SOFTWARE PROCESSES

MONTREAL, SEPTEMBER 1, 2012

© Copyright 2012 Malik QASAMEH

© Copyright reserved

It is forbidden to reproduce, save or share the content of this document either in whole or in parts. The reader who wishes to print or save this document on any media must first get the permission of the author.

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain Abran, Ph.D, Thesis Supervisor
Software Engineering and Information Technology Department, École de technologie supérieure

Mr. Nicolas Constantin, Ph.D, President of the Board of Examiners
Electrical Engineering Department, École de technologie supérieure

Mrs. Ghizlane El Boussaidi, Ph.D, Examiner
Software Engineering and Information Technology Department, École de technologie supérieure

Mr. Hamid Mcheick, Ph.D, External Examiner
Computer Science and Software Engineering Department, Université du Québec à Chicoutimi

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND THE PUBLIC

17 AUGUST, 2012

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENT

I would like to seize this opportunity to express my gratitude to all who have helped me in my graduate studies. First and foremost, I would like to thank my advisor, Professor Alain Abran, for his incredible support in all stages of my doctoral study. His kindness, encouragement, wise advice, knowledge, and professionalism have made working with him a great pleasure and honor. His professionalism and positive attitude towards research passed on to me a sound technical knowledge to the area of software engineering that will accompany me wherever I go. He nicely devoted so much of his time through face to face meetings and very fast email feedbacks to help me succeed in my graduate studies. I could not have asked for a better adviser to direct me through my doctoral studies. Also, I would like to thank everyone in the Software Engineering Research Laboratory (GÉLOG) and the Department of Software Engineering and IT for their support.

I would like to acknowledge the Compass Group at Montréal and the internal awards committee in ETS for their financial support and graduate fellowships.

I am deeply grateful to my committee members Prof. Nicolas Constantin, Prof. Ghizlane El Boussaidi, and Prof. Hamid Mcheick for their time and effort in reviewing this work.

A great love to my parents, my father (Prof. Ghazi) the source of strength and wisdom in my life, my mother (Ghazieh) whom I need to spend my entire life to honor and thank. You have shown me the joy of intellectual pursuit ever since I was a child. Thanks to my sisters (Dr. Dania and Ghaida) and to my brothers (Dr. Motaz, Dr. Mohammad, Raji and Salem) for their moral support. Last but not least, my deep appreciation to all my friends (especially Dr. Mohammad Ameen Qasaimeh, Dr. Khalid Al Makhadmeh and Eng. Derar Rizk), roommates and ex-roommates in Montréal.

AUDITING FOR ISO 9001 REQUIREMENTS IN THE CONTEXT OF AGILE SOFTWARE PROCESSES

Malik QASAMEH

ABSTRACT

ISO 9001 demands of (software) organizations that a rigorous demonstration of their software processes be implemented and a set of guidelines followed at various levels of abstraction. What these organizations need to show, in other words, is that their software processes have been designed and implemented in a way that allows for a level of configuration and operation that complies with ISO 9001 requirements.

For software organizations needing ISO 9001 certification, it is important that they establish a software process life cycle that can manage the requirements imposed by this certification standard. However, software organizations that develop their software products using the agile software processes, such as Extreme Programming (agile-XP), face a number of challenges in their effort to demonstrate that their process activities conform to ISO 9001 requirements, major ones being: product construction, traceability, and measurement. Agile software organizations must provide evidence of ISO 9001 conformity, and they need to develop their own procedures, tools, and methodologies to do so.

As yet, there is no consensus on how to audit the agile software organization to ensure that their software processes have been designed and implemented in conformity with ISO 9001 requirements. Moreover, it is challenging to ensure that such lightweight documentation methodologies meet these requirements for certification purposes.

The motivation of this research is to help software organizations that use agile software processes in their effort to meet the ISO 9001 certification requirements. This research project is also aimed at helping IS auditors extract auditing evidence that demonstrates conformity to the ISO 9001 requirements that must be met by agile software organizations. Extreme programming (agile-XP) has been selected for improvement as a candidate agile process. This selection was based on the literature indicating a higher adoption of agile-XP over other agile software processes.

The goal of this research project is to improve the ability of the agile-XP process to meet the auditing requirements of ISO 9001. The goal of the research also focuses on helping agile software organizations in their effort to become ISO 9001 certified.

The main objective of this research project is to design an auditing model that covers the measurement and traceability requirements of ISO 9001. The auditing model should provide IS auditors with auditing evidence that the software projects developed with the agile-XP process have fulfilled the requirements of ISO 9001. The objective also proposes several sub

processes to enhance the early planning activities of agile-XP according to ISO 9001 requirements.

To achieve these objectives, the main phases of the research methodology are: Investigation of the capability of agile-XP to achieve the requirements of ISO 9001 software process certification; modification of the early phases of agile-XP (i.e. release planning phase) using CMMI-DEV; and design of an auditing model for ISO 9001 traceability and measurement requirements.

The main outcome of this research study, which is an auditing model that is aligned with the principles of agile-XP and focuses on ISO 9001 traceability and measurement requirements to provide the IS auditors with a methodological approach for the auditing process. The auditing model has been assessed based on case studies selected from the literature.

Keywords: Agile Software Process Improvements, Certification Process, Auditing, Evaluation Theory, ISO 9001, ISO 90003, Software Process Improvement, Engineering Design.

AUDIT DES REQUIS ISO 9001 DANS LE CONTEXTE DES PROCESSUS LOGICIELS AGILES

Malik QASIMEH

RÉSUMÉ

ISO 9001 exige des organisations une démonstration rigoureuse de la mise en œuvre de leurs processus logiciels et un suivi d'un ensemble de lignes directrices à différents niveaux d'abstraction. En d'autres termes, ce que ces organisations ont besoin de démontrer c'est que leurs processus logiciels ont été conçus et mis en œuvre d'une manière qui permette un niveau de configuration et de fonctionnement qui est conforme à la norme ISO 9001.

Pour les organisations de logiciels qui ont besoin de la certification ISO 9001, il est important d'établir un processus de cycle de vie logiciel qui permet de gérer les exigences imposées par la présente norme de certification. Toutefois, les organisations de logiciels qui développent leurs produits logiciels en utilisant les processus logiciels agiles comme l'Extreme Programming (XP-agile) font face à un certain nombre de défis dans leurs efforts pour démontrer que leurs activités de processus sont conformes aux exigences d'ISO 9001. Les plus importantes exigences ISO 9001 sont liées à la construction du produit logiciel, la traçabilité et la mesure. Les processus dits Agile doivent fournir la preuve de la conformité ISO 9001, et ils doivent développer leurs propres procédures, outils et méthodologies pour ce faire.

Pour l'instant, il n'y a pas de consensus sur la façon de vérifier les organisations de type agile pour s'assurer que leurs processus logiciels ont été conçus et mis en œuvre en conformité avec les exigences ISO 9001. Il est donc difficile de prendre en compte des méthodologies de documentation du processus légers (par exemple logiciel agile agile-XP) pour démontrer que les exigences ISO 9001 ont été rencontrées.

La motivation de cette recherche est d'aider les organisations de logiciels qui suivent des processus logiciels agiles à répondre aux exigences de certification ISO 9001. Ce projet de recherche vise également à aider les vérificateurs logiciels pour extraire des preuves d'audit qui démontrent la conformité aux exigences ISO 9001 des organisations de logiciels agiles. Extreme programming (XP-agile) a été choisi pour être amélioré comme un processus agile candidat. La sélection est basée sur la littérature indiquant une plus grande adoption de agile-XP parmi les autres processus de développement logiciel agile.

Le but de ce projet de recherche est d'améliorer le processus Agile-XP pour rencontrer les exigences de vérification de la norme ISO 9001. L'objectif de la recherche vise également à aider les organisations de logiciels agiles dans leurs efforts pour devenir certifié ISO 9001.

Un premier objectif de ce projet de recherche est de concevoir un modèle d'audit qui couvre l'exigence de mesure et de traçabilité de l'ISO 9001. Le modèle de vérification devrait fournir aux auditeurs des preuves d'audit que les projets de logiciels développés avec agilité-XP processus ont rempli les exigences de la norme ISO 9001. Le second objectif vise à proposer plusieurs sous-processus pour améliorer les activités de planification au début de agile-XP et selon les exigences ISO 9001.

Pour atteindre ces objectifs, les principales phases de la méthodologie de recherche sont: Etude de la capacité de agile-XP pour atteindre les exigences de la certification ISO 9001 des processus logiciels; Modification de la phase précoce de agile-XP (i.e. la phase de planification) à l'aide du modèle CMMI-DEV; conception d'un modèle d'audit ISO 9001 pour rencontrer les exigences de traçabilité et de mesure.

Le principal résultat de cette étude est un modèle d'audit qui est aligné avec les principes de souplesse-XP et se concentre sur la norme ISO 9001, et en particulier sur la traçabilité et les exigences de mesure de fournir des auditeurs est une approche méthodologique pour le processus de vérification. Le résultat de cette recherche a été évalué sur la base de neuf études de cas identifiées dans la littérature.

Mots-clés: Agile amélioration des processus logiciels, processus de certification, Audit, théorie de l'évaluation, ISO 9001, ISO 90003, l'amélioration des processus logiciels, de conception technique.

TABLE OF CONTENTS

	Pages
INTRODUCTION	1
CHAPTER 1 AGILE SOFTWARE PROCESS IN THE LITERATURE REVIEW	7
1.1 Introduction.....	7
1.2 Software process and software life cycle model.....	8
1.2.1 ISO-IEEE viewpoint	8
1.3 Agile software development	11
1.4 Related work on agile software processes and ISO 9001	12
1.4.1 Vitoria (2004).....	12
1.4.2 Vriens (2003)	12
1.4.3 Wright (2003).....	12
1.4.4 Maurer <i>et al.</i> (2002)	13
1.5 Agile software process in systematic reviews	13
1.5.1 Jalali and Wohlin (2011).....	13
1.5.2 Diaz <i>et al.</i> (2011)	20
1.6 Statistical evidence on agile software adoption	26
1.6.1 Industry based evidences	26
1.6.2 Academic based evidences	28
1.7 Agile software processes.....	30
1.7.1 Extreme Programming (XP)	30
1.7.2 Scrum	32
1.7.3 Feature-Driven Development (FDD).....	33
1.7.4 Adaptive Software Development (ASD)	35
1.7.5 Crystal Methodologies.....	36
1.8 Comparison of agile processes based on software design and projects requirements	38
1.9 Experiments classification of agile literature.....	41
1.10 Summary	46
CHAPTER 2 ISO 9001 AND AUDITING PRINCIPLES	49
2.1 Introduction.....	49
2.2 Adoption of ISO 9001 in software organizations	51
2.3 Evolution of auditing practices	56
2.4 Auditing for certification	57
2.5 ISO 9001	60
2.6 Summary	62
CHAPTER 3 RESEARCH GOAL, OBJECTIVES, AND METHODOLOGY	65
3.1 Introduction.....	65
3.2 Research motivation.....	65
3.3 Research goal	65
3.4 Research objectives.....	66

3.5	Research inputs	66
3.6	Research users.....	67
3.7	Overview of the research methodology	67
3.8	Detailed research methodology.....	69
CHAPTER 4 ANALYSIS OF AGILE-XP FROM ISO 9001 PERSPECTIVE		77
4.1	Introduction.....	77
4.2	Analysis scope and design	78
4.2.1	Analysis scope	78
4.2.2	Design process for the analysis.....	79
4.3	Mapping results.....	82
4.3.1	Planning of product realization	82
4.3.2	Requirements phase	85
4.3.2.1	ISO Requirements during software requirement gathering activity	85
4.3.3	Construction phase.....	88
4.3.4	Design and development verification and validation.....	91
4.4	Summary.....	95
CHAPTER 5 EXTENDING AGILE-XP USER STORIES TO MEET ISO 9001 FORMALITY REQUIREMENTS		97
5.1	Introduction.....	97
5.2	Terminology.....	98
5.2.1	System.....	98
5.2.2	System feature and system function from the XP viewpoint.....	98
5.3	Design for user stories extension	99
5.4	Proposed sub processes.....	103
5.4.1	Identify the source of the user story.....	103
5.4.2	Categories of non functional requirements	110
5.4.3	Identify the user story relationships.....	113
5.4.4	Prioritizing the user stories	116
5.5	Extended user story for XP	125
5.6	Summary and Discussion.....	127
CHAPTER 6 AN AUDIT MODEL FOR ISO 9001 TRACEABILITY REQUIREMENTS IN AGILE-XP ENVIRONMENTS		131
6.1	Introduction.....	131
6.2	Analysis of traceability requirements in ISO 9001	132
6.2.1	Support for change management	133
6.2.2	Cost management.....	134
6.2.3	Process improvements	135
6.3	Design process	137
6.3.1	Evaluation fundamentals.....	137
6.4	Design of the Auditing Model	141
6.4.1	Scope delimitation	141
6.4.2	Design of the audit criteria and yardsticks.....	141

6.5	Summary	147
CHAPTER 7 EXTENDING THE AUDITING MODEL BY COVERING THE ISO 9001 MEASUREMENT REQUIREMENTS		
7.1	Introduction.....	149
7.2	Analysis of measurement requirements in ISO 9001	150
7.2.1	Analysis of section 8.2.1: customer satisfaction.....	150
7.2.2	Analysis of section 8.2.3 of ISO 9001: measurement of processes.....	152
7.2.3	Analysis of ISO 9001 section 8.2.4: measurement of products.....	156
7.2.4	ISO 9001 section 8.2.2: Internal Auditing.....	157
7.4	Existing agile measurement and estimation techniques.....	161
7.5	Design process	165
7.5.1	Engineering design process.....	165
7.5.2	Design formulation	166
7.5.3	Subdivision of design component.....	171
7.6.1	A: Auditing criteria for measurement plans.....	176
7.6.2	B: Auditing criteria for measurement development.....	180
7.6.3	C: Auditing criteria for measurement management.....	185
7.6	Summary	187
CHAPTER 8 CASE STUDIES.....		
8.1	Introduction.....	189
8.2	Classification of auditing evidences	191
8.3	Case studies: agile traceability audit.....	192
8.3.1	Context and scope.....	192
8.3.2	Traceability audit of Case A _{TR} (Espinoza, Garbajosa)	196
8.3.3	Auditing for the five case studies based on yardstick _{TR} #6 (Traceability item relationships).....	204
8.4	Case studies: agile measurement audit	208
8.4.1	Context and scope.....	208
8.4.2	Preliminary considerations.....	212
8.4.3	Measurement audit of Case B _{MR} (Mahnic, Zabkar).....	213
8.4.4	Auditing of measurement case studies for yardstick _{MR} #5 (Measurement data and models)	221
8.4.3	Auditing summary of agile measurement case studies	222
8.5	Summary	227
CONCLUSION.....		229
BIBLIOGRAPHY		237

LIST OF TABLES

	Pages
Table 1.1 Agile distribution in Global Software Engineering (GSE) -----	16
Table 1.2 Agile distribution of research type in the area of Global Software Engineering (GSE).....	16
Table 1.3 Location & Agile practices in Agile-Distribution settings-----	18
Table 1.4 IBM rational methods group survey -----	27
Table 1.5 Comments on the agile surveys -----	29
Table 1.6 Comparison of agile processes based on the project and design requirements-	40
Table 1.7 Summarization and classification of the experiments -----	43
Table 4.1 ISO 9001 planning phase mapping -----	85
Table 4.2 ISO 9001 Requirement gathering and validation mapping -----	88
Table 4.3 Classification of Review Methods -----	89
Table 4.4 ISO 9001 Construction phase mapping -----	91
Table 4.5 ISO 9001 Construction phase mapping -----	94
Table 5.1 Derived XP sub processes-----	102
Table 5.2 ISO 9126 quality characteristics -----	111
Table 5.3 Examples of non functional capability categories-----	113
Table 5.4 AHP scale -----	121
Table 5.5 Pairwise matrix for the selected criteria -----	122
Table 5.6 Pairwise matrix for the cost criterion -----	122
Table 5.7 Pairwise matrix for time criterion-----	123
Table 5.8 Pairwise matrix for the risk criterion-----	123
Table 6.1 ISO 9001 obligations and CMMI KPAs corresponding to process-----	136
Table 7.1 Agile estimation techniques and their weaknesses from measurement and auditing perspectives -----	164
Table 7.2 Examples of publications work based on ISO 15939-----	173
Table 7.3 Questions based context analysis for auditing evidences extraction -----	178
Table 7.4 Examples of base measures and derived measures in the CMMI model-----	182

Table 8.1	Selected case studies for the agile (XP) traceability audit-----	195
Table 8.2	Traceability audit of case A _{TR} -----	198
Table 8.3	Traceability audit of case B _{TR} -----	200
Table 8.4	Traceability audit of case C _{TR} -----	201
Table 8.5	Traceability audit of case D _{TR} -----	202
Table 8.6	Traceability audit of case E _{TR} -----	203
Table 8.7	Audit based on yardstick _{TR} #6 -----	205
Table 8.8	Summary of evidences in the selected case studies -----	207
Table 8.9	Selected case studies for the audit of agile (XP) measurement-----	210
Table 8.10	Measurement audit of case B _{MR} -----	216
Table 8.11	Measurement audit of case A _{MR} -----	218
Table 8.12	Measurement audit of Case C _{MR} -----	219
Table 8.13	Measurement audit of Case D _{MR} -----	220
Table 8.14	Audit based on Yardstick _{MR} #5 -----	222
Table 8.15	Existence of evidence in the selected case studies -----	226

LIST OF FIGURES

	Pages
Figure 1.1 ISO 12207:2008 processes (ISO/IEC 12207:2008 -Reprinted with permission from IEEE, Copyright 2008 by IEEE, 3 Park Avenue, New York, NY 10016-5997 USA, All rights reserved) -----	9
Figure 1.2 SWEBOK ‘software engineering process’ knowledge area -----	10
Figure 1.3 The extent of use of different agile process reported in -----	27
Figure 1.4 Extreme programming - XP -----	30
Figure 1.5 Scrum process -----	32
Figure 1.6 Steps of FDD -----	34
Figure 1.7 ASD process activities-----	35
Figure 1.8 Crystal methodologies-----	37
Figure 2.1 ISO 9126 quality approach-----	50
Figure 2.2 Maturity distributions of Danish organizations without ISO 9001 certification, Hass, Johansen <i>et al.</i> (1998) -----	52
Figure 2.3 Maturity distributions of Danish organizations with ISO 9001 certification, --	52
Figure 2.4 Generic auditing model-----	59
Figure 2.5 ISO 9001 Time line -----	61
Figure 3.1 Detailed research methodology -----	75
Figure 4.1 ISO 9001, clause 7 -----	79
Figure 4.2 ISO 12207 focused processes-----	80
Figure 4.3 Mapping process & phases-----	81
Figure 4.4 Validation and verification process -----	91
Figure 5.1 Relationship between system features and system functions in XP -----	99
Figure 5.2 Methodology for deriving the XP sub-processes-----	101
Figure 5.3 User story sources – the various types of contributors -----	105
Figure 5.4 Government side contributors -----	109
Figure 5.5 Logical dependency -----	114
Figure 5.6 Data dependency-----	114

Figure 5.7	Temporal dependency -----	115
Figure 5.8	Resource dependency -----	115
Figure 5.9	Procedure for prioritizing the user stories in XP using the AHP-----	118
Figure 5.10	AHP diagram for user story selection -----	120
Figure 5.11	A priority hierarchy-----	124
Figure 5.12	Calculation of user stories priority -----	125
Figure 5.13	Extended user story -----	126
Figure 6.1	Components of an evaluation procedure Lopez (2000)-----	138
Figure 6.2	Design process for the audit model-----	140
Figure 6.3	The structure of the proposed auditing model -----	142
Figure 7.1	Relations of ISO 9001section 8.2.1” Customer satisfaction” to other sections in ISO 9001-----	151
Figure 7.2	Relations of ISO 9001 section 8.2.3 “Measurement of processes” to other sections in ISO 9001 -----	154
Figure 7.3	Relation of ISO 9001section 8.5.2 “corrective action” to the ISO standards -	156
Figure 7.4	Relation of ISO 9001 section 8.2.3 “Measurement of processes” to other sections in ISO 9001 -----	157
Figure 7.5	Relations of ISO 9001 section 8.2.2 “Internal auditing” to other sections in ISO 9001 -----	159
Figure 7.6	An example of a burn down chart -----	163
Figure 7.7	Vincenti's levels of engineering design -----	166
Figure 7.8	The elicitation of auditing audit criteria -----	169
Figure 7.9	The structure of the measurement auditing model -----	170
Figure 7.10	Auditing model design levels -----	175

LIST OF ABBREVIATIONS

AHP	Analytic Hierarchy Process
APLE	Agile Product Line Engineering
ASD	Adaptive Software Development
ATAM	An Evaluation Theory Perspective of the Architecture Tradeoff Analysis Method
BPMN	Business Process Modeling Notation
CMMI	Capability Maturity Model Integration
COBIT	Control Objectives for Information and related Technology
COCOMO	Constructive Cost Model
COSMIC	Common Software Measurement International Consortium
CPI	Cost Performance Index.
DE	Domain Engineering
DSDM	Dynamic Systems Development Method
ECSS	European Cooperation on Space Standardization
EEML	Extended Enterprise Modeling Language
FDA	Food and Drug Administration
FDD	Feature Driven Development
FSM	Software Functional Size Measurement
GQM	Goal Question Metric
HIPAA	Health Insurance Portability and Accountability Act
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers

IS	Information System
ISA	International Standard of Auditing
ISACA	Information Systems Audit and Control Association
ISBSG	International Software Benchmarking Standard Group
ISO	International Organization for Standardization
KAs	knowledge Areas
KPA	Key Process Area
KPIA	Key Process Indicator Areas
MPS-BR	Model for Brazilian Software Process Improvement
QFD	Quality Function Deployment
QMS	Quality Management System
RD	Requirement Development
REQM	Requirement Management
RM	Risk management
SCM	Software Configuration Management
SEI	Software Engineering Institute
SLCM	Software Life Cycle Model
SOMF	Service-Oriented Modeling Framework
SOX	Sarbanes-Oxley
SPLE	Software Product Line Engineering
SPM	Software Process Model
SWEBOK	Software Engineering Body of Knowledge
TQS	Thai Quality Software

UML	Unified Modeling Language
US	User Story
USP	User Story Point
VSEs	Very Small Entities
WSEAS	World Scientific and Engineering Academy and Society
XP	Extreme Programming

INTRODUCTION

Customers expect from their software vendors a product that can perform the desired functionality while maintaining the required quality attributes and characteristics, such as maintainability, reliability, usability, and efficiency. However, developing quality software is not an easy task and requires that the software development organizations continuously improve their processes and strategies. An important standard designed to guide organizations to develop a quality product is ISO 9001. This research work focuses on improving the audit ability of agile software processes (e.g. agile-XP), in particular from the ISO 9001 perspective. The problem statement underlying this thesis and the structure of the thesis are detailed next.

Problem Statement

The origins of ISO 9001 can be traced back to the manufacturing sector; however, this quality standard is now being applied to many other types of organizations, including health care. Since the standard is neither industry nor product specific, it may be used by any organization which needs to provide a high quality product or service. The development of software has also become an important endeavor in ISO member countries, and the ISO has developed and released a set of software engineering guidelines to serve as a roadmap to enable software development organizations to become ISO 9001 certified. These guidelines are contained in the ISO 90003 publication, and organizations that need to be ISO 9001 certified can use it when audited to show evidence that they have implemented the ISO 9001 requirements. However, ISO 90003 does not recommend any tool, methodology, or software process to software organizations that can be implemented to support the certification requirements.

The ISO is not itself a certification body. A certification body is a third-party agency that is responsible for auditing an organization's activities to ensure their ability to comply with specific standards such as ISO 9001. Certification bodies usually charge for their services. If no major non conformity is found, the certification body will issue a certificate based on a

specified auditing scope (e.g. product construction). Small- to mid-sized software organizations (such as agile software organizations) tend to find the certification process with ISO 9001 costly. Such costs include consulting fees, training programs, auditing costs, and registration.

ISO 9001 impacts the entire range of software life cycle activities, including software planning, software requirement gathering and analysis, software construction activities, the software life cycle traceability process, and the measurement process. To address all these activities, software organizations that need to become ISO 9001 certified find themselves in a position where they need to develop myriad tools and technique to demonstrate that their software processes are in conformity with the quality standard. A common methodology is to have in place a certification team (i.e. software analysts), which is responsible for understanding which ISO 9001 clauses impact the organization's business processes, including software process activities. This team must also assess the development team, to demonstrate that the software products are being developed according to ISO 9001 requirements. In essence, this means providing documented evidence that clarifies how and when a particular design decision has been implemented. The collection of evidence constitutes a very important foundation on which the IS auditors base their audit results and conclusion.

Prior to our undertaking the research reported in this thesis, there was no model or framework designed to help agile software organizations in their effort to become ISO 9001 certified. Consequently, it was challenging for them to provide the IS auditors with audit evidence demonstrating they were meeting the ISO 9001 requirements.

This document reports on the research carried out to enhance the audit ability of software organizations that have adopted a lightweight software life cycle model, such as agile-XP, and decide to obtain ISO 9001 certification. The enhancement has been achieved by modifying the early phase of agile-XP to accommodate important information related to ISO

9001. The enhancement process also focuses on the development of an auditing model for process traceability and process measurement.

Thesis organization

This thesis contains ten chapters (including the introduction and the conclusion). In this current chapter, we outline the structure of the thesis, as follows.

Chapter 1 presents an analysis of the literature related to agile software process deployment and improvement, and focuses on identifying evidence concerning the adoption of agile software processes in the context of software organizations. This chapter also provides a comparison of agile software processes, based on key requirements for software development, as to understand their strength and weaknesses.

Chapter 2 presents an overview of auditing practices and the ISO 9001 certification process in the context of software organizations, and focuses on identifying the potential advantages for software organizations in becoming ISO 9001 certified. This chapter also presents an analysis of several studies and surveys that report on the implementation of ISO 9001 in software organizations.

Chapter 3 presents the definition of our research project, including the research motivation, goal, and objectives, as well as the users of the research results. This chapter also presents our detailed methodology, which is designed to tackle the research objectives, including the research phases and research inputs.

Chapter 4 identifies the main ISO 9001 requirements that have a direct impact on the software process life cycle model, which is mainly based on ISO 12207 terminologies. This chapter also presents an analysis of the strength and weaknesses of agile-XP in terms of meeting ISO 9001 requirements. Our conclusions with respect to the capabilities of agile-XP are then presented.

Chapter 5 provides a modification of the structure of traditional user stories, in order to supply the ISO 9001 auditor of agile-XP with sufficient evidence that the data they require

have been collected, and to enable traceability for the requirements throughout the earliest phases of agile-XP (i.e. the release planning phase). This modification is applied following the design of four sub processes (activities) aligned with agile-XP release planning phase. These sub processes are: 1) identification of the user story resources; 2) identification of a non functional requirements category; 3) identification of user story relationships; and 4) identification of user story priorities.

Chapter 6 presents an analysis of the ISO 9001 traceability requirements with the objective of developing an auditing model aligned with agile-XP. This model focuses on identifying a set of auditing criteria that supports the ISO 9001 traceability requirements. The model uses evaluation theory principles as a framework on which to build the proposed auditing model. This chapter also presents the engineering design principles used as a basis for building the proposed auditing model.

Chapter 7 presents an extension for the auditing model to support the ISO 9001 measurement requirements. This chapter also presents an analysis with respect to ISO 9001 measurement requirements with the objective of developing an auditing model aligned with agile-XP. The extension consists of three major categories of auditing criteria, which focus on the evidence that can be extracted to demonstrate process conformity with the ISO 9001 measurement requirements.

Chapter 8 presents the case studies selected for an auditing process involving five traceability approaches and four measurement approaches in the area of agile software processes. This chapter also extracts the auditing evidence from the selected agile approaches to assess their conformity to ISO 9001 traceability and measurement requirements.

The concluding chapter summarizes the results of this thesis, as well as its contributions and limitations, and suggestions for future work.

CHAPTER 1

AGILE SOFTWARE PROCESS IN THE LITERATURE REVIEW

1.1 Introduction

Software development practices have evolved significantly since the term software engineering was popularized by F. L. Bauer during the NATO Software Engineering Conference in 1968. There exist today a large number of software process life cycle models that have been introduced and studied to a great extent, but up to date none has proven to be the golden life cycle model and each model has its own advantages and disadvantages.

Agile software processes challenges the traditional way of software development and project management. In rapidly changing environments, changing requirements and tight schedule constraints require software developers to understand the main features of agile software processes. The objective of this chapter is to identify the adoption level of different agile software processes in both academia and industry. Additionally, several agile software processes will be presented and compared to provide an understanding of the main similarities and differences between the selected agile software processes.

This chapter presents a survey and analysis of the related literature and is organized as follows:

- Section 1.2 provides an overview of the common software processes in ISO 12207 and SWEBOK;
- Section 1.3 provides an overview of the agile software process;
- Section 1.4 discusses the related work on agile software processes and ISO 9001;
- Section 1.5 discusses the systematic reviews in agile software process;
- Section 1.6 provides statistical evidences on agile software adoption' and compare different practices of agile software processes;

- Section 1.7 provides summarization for the practices of agile software processes;
- Section 1.8 compares the agile software processes;
- Section 1.9 provides classification for the twenty experiment in the area of agile software processes;
- A summary is presented in section 1.10.

1.2 Software process and software life cycle model

1.2.1 ISO-IEEE viewpoint

ISO-IEEE12207:2008 is an international standard that establishes a common framework for software life cycle processes. It defines a process as “A set of interrelated activities, which transform inputs into outputs”, and a life cycle model as a “framework of processes and activities concerned with the life cycle that may be organized into stages, which also acts as a common reference for communication and understanding of the process terminologies”.

From the ISO 12207 perspective, *“the life of a system or a software product can be modelled by a life cycle model consisting of stages. Models may be used to represent the entire life from concept to disposal or to represent the portion of the life corresponding to the current project. The life cycle model is comprised of a sequence of stages that may overlap and/or iterate, as appropriate for the project's scope, magnitude, complexity, changing needs and opportunities. Each stage is described with a statement of purpose and outcomes. The life cycle processes and activities are selected and employed in a stage to fulfil the purpose and outcomes of that stage”*.

The processes of ISO 12207:2008 have been classified into two categories: system context processes and software specific processes. The system context processes are: agreement processes, project processes, technical processes, and organizational project-enabling processes. The software specific processes are: software implementation processes, software support processes and software reuse processes. Each process has its sub processes, as shown

in figure 1.1. The main objective of ISO12207:2008 is to provide the software development team with common definitions and terminologies throughout the software life cycle.

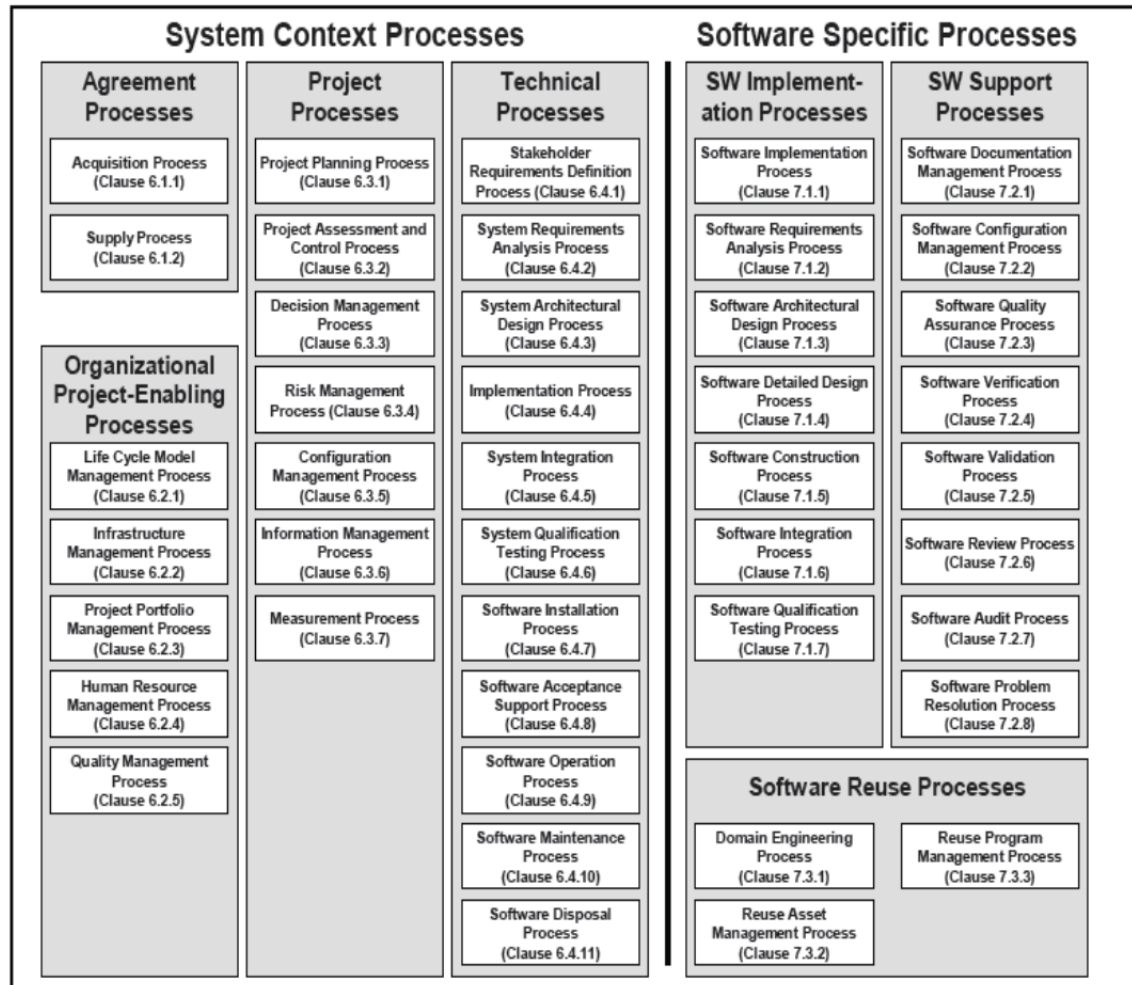


Figure 1.1 ISO 12207:2008 processes (ISO/IEC 12207:2008 -Reprinted with permission from IEEE, Copyright 2008 by IEEE, 3 Park Avenue, New York, NY 10016-5997 USA, All rights reserved)

The SWEBOK Guide - ISO TR 19759 Abran, Moore *et al.* (2004) defines ten knowledge areas (KAs) within the field of software engineering: requirements, design, construction, testing, maintenance, configuration management, engineering management, engineering process, engineering tools and methods, and quality. All software organizations are expected to implement the best practices listed in all those KAs, or in a subset of them. The knowledge area on 'software engineering process' lists and describes the following topics: process

implementation and change, process definition, process assessment and process and product measurement- See figure 1.2.

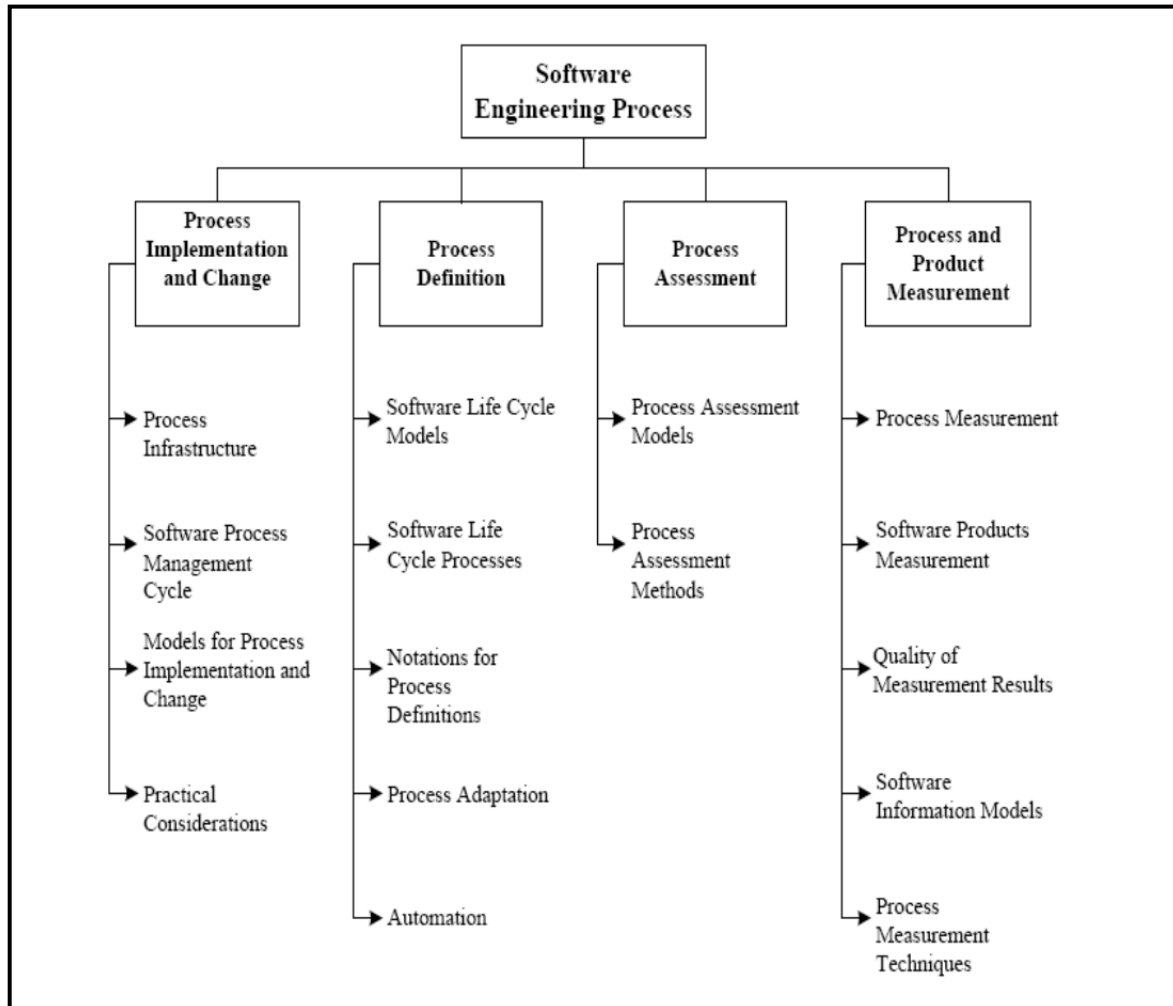


Figure 1.2 SWEBOK ‘software engineering process’ knowledge area
Abran, Moore *et al.* (2004)

IEEE-610-1991 defines the software life cycle model (SLCM) as a “set of operations that is repeated regularly in the same sequence, possibly with variations in each repetition”.

In the field of software engineering, there are many software life cycle models that are available in the literature; however, each model has its own characteristics, advantages and disadvantages.

Software life cycle models have been classified into two main categories. One is called plan-driven development or traditional software development models and the other is called lightweight development models or agile software development. The next sections will focus on the details of agile software development.

1.3 Agile software development

A study has been conducted by Baskerville, Levine *et al.* (2002) to identify the factors that influence the development practices in nine development organizations. This study has identified three main factors:

- Demand for rush to market;
- Operating in a different type of market environment;
- Lack of experience developing such products.

The study of Baskerville, Levine *et al.* (2002) has concluded that the software organizations should enhance their software processes by integrating the following practices:

- Customers should be intensely involved in development and guiding the construction and design of the product.
- Prototyping should be heavily used in understanding requirements.
- The application should be evolving through frequent releases.

Agile processes have been proposed to overcome the inflexibility issues of traditional processes. They have been developed by practitioners based on their experience working on several software development projects Cao (2006); Nerur, Mahapatra *et al.* (2005). Although existing agile processes can differ in the way they approach software development, they all share one key characteristic which consists of favouring close collaboration between software development and business teams via face-to-face communication, as opposed to putting an emphasis on written documentation (Highsmith, 2000) (Nerur, Mahapatra *et al.*, 2005).

1.4 Related work on agile software processes and ISO 9001

A number of authors have studied the relationships between agile software processes and ISO 9001 requirements. The focuses of these authors differ based on the selected ISO 9001 provision or set of requirements investigated. This section is a summary of the related research work.

1.4.1 Vitoria (2004)

Vitoria (2004) studies the ISO 9001 and TickIT standard and analyzes how it has been used in two case studies with agile projects. Vitoria reports for these two projects that 33% of ISO 9001 requirements could not be applied in an XP project, 24% could be partially applied, 20% could be applied in full, while 23% were not relevant to the scope of the projects.

1.4.2 Vriens (2003)

Vriens (2003) discusses CMM, ISO 9001 and their relationships to XP and Scrum: he observes that most of the ISO 9001 requirements are independent of development methods used and are covered by the existing processes. This author reports on his experience of getting certified for both CMM Level 2 and ISO 9001:2000 on a time scale of 2 years by using agile methodologies.

1.4.3 Wright (2003)

Wright (2003) describes a successful certification story for an XP organization. This author describes how the organization managed the large team through the practice of XP and highlights the tools used to support the project team to handle the ISO 9001 requirements: this author focus is only on some selected ISO 9001 requirements and he highlights their corresponding XP support activities.

1.4.4 Maurer *et al.* (2002)

In a study evaluating the success of XP principles in Web development, Maurer and Martel (2002) report an average increase of 66% in new lines of code produced, a 302% increase in the number of new methods developed and a 283% increase in the number of classes implemented.

1.5 Agile software process in systematic reviews

This section summarizes the findings of two recent systematic reviews in the area of agile software processes. The importance of those studies is highlighted next:

- These recent studies have been published in 2011.
- These studies have identified and analyzed the findings of many other research papers (i.e. each study summarizes the findings of more than 10 papers in the area of agile software process).
- The studies follow a disciplined approach based on Kitchenham (2007) to identify, analyze and interpret all the available evidences related to research goals.
- Instead of focusing on the agile software process success in developing small and medium software projects, these studies reveal the status of agile software process in developing large software (e.g. Global project development and Software Product Line Engineering) where multiple collaborations between the project stakeholders are needed and efficient software practices are required.

1.5.1 Jalali and Wohlin (2011)

1.5.1.1 Goal of this study

In this study, Jalali and Wohlin (2011) carry out a Systematic Review (SR) to investigate the application of agile methods and practices in Global Software Engineering (GSE) in existing research.

The authors conducted peer-reviewed research produced between the years 1999 and 2009. The authors note that the majority of existing research report industrial experience on

adjustments of agile practices for application in GSE. The authors claim the results are supported with examples of usage of agile software processes in different areas of GSE. The authors suggest that a need exists to develop a thorough, extensive framework for application of agile software processes in GSE.

1.5.1.2 Methodology of this study

The authors developed the research questions, search terms and keywords. In parallel, the authors selected a list of key papers as a reference for validation of results and search terms. The research questions were:

Research question 1: What has been reported in the target literature about agile software process and practices in GSE?

Research question 2: Which Agile practices in which GSE settings, and under which circumstances, have been successfully applied?

The authors defined two sets of search keywords, one for agile software processes, and one for distributed development. Search strings were combinations of search keywords using OR & AND operators. The search was conducted for agile software processes of SCRUM, XP, and lean software development. The set of GSE search keywords were different synonyms and spellings of GSE, distributed development, global teams and outsource.

The data sources included databases ACM Portal, IEEE Xplore, Compendex, and Scopus. The research considered literature done in English language in the interval 1999-2009. The search was limited to the title, abstract and keywords sections of papers.

The papers were categorized into relevant, irrelevant and maybe relevant categories. For more objective decision making, the two authors/researchers made their decisions separately. Papers with at least one 'irrelevant' vote, and one 'maybe relevant' vote were excluded. Papers with two 'maybe relevant' votes were included for further investigation. Only unique papers were considered. Finally, 81 studies were selected out of initially 534 papers.

The authors used MS Excel in data extraction and collection. The authors classified the papers according to the research type into the following categories:

- Evaluation research: Practical implementation and analysis of the outcome.
- Validation research: Lab studies of novel techniques.
- Solution proposal: Elaborate discussion of a solution to a problem.
- Philosophical papers: Focus is on the concepts of a framework.
- Experience papers: Personal experience on practical implementation of a solution.
- Opinion papers: Reflection of personal opinion.

Further analysis was based on the full text of each paper. Classifications of papers were carried out, with respect to research methodology, empirical background, findings, participants and context.

1.5.1.3 Findings of this study

Table 1.1 presents the number of papers found in each data source per each year. The results suggest a higher interest on Agile GSE in last 5 years.

Table 1.2 presents the number of papers in each research type over the years of the study. The majority of the literature is on experience reports. The authors note the need for more philosophical, evaluation and validation studies to establish a more solid basis for the area of agile software processes and GSE.

Table 1.1 Agile distribution in Global Software Engineering (GSE)
Jalali and Wohlin (2001)

Year	2001	2002	2003	2004	2005	2006	2007	2008	2009
		2							
ACM				2	2	3			2
IEEE			1	2	1	2	6	15	9
Compendex		1		1	2	4	4	2	2
Inspec			1	5	1	3	2	1	
AIS							1	2	
Scopus							1		4
Total		1	2	10	6	12	14	20	17

Table 1.2 Agile distribution of research type in the area of Global Software Engineering (GSE)

Research Type	2002	2003	2004	2005	2006	2007	2008	2009
Evaluation			1		3	3	3	4
Validation			1		1	1	2	
Solution	1	1	1		3	1		1
Philosophical								2
Experience	1		5	4	2	9	14	7
Opinion			1	2	3		1	3

Successful applications

In total, the authors reported 53 successful examples out of the 81 papers. The most used Agile-distribution combinations were found to be Agile-offshore, XP-distributed teams and Agile-distributed teams.

Countries involved: Asian countries like India and Malaysia were found to be popular outsourcing destinations. Collaboration between USA and India constitutes a large portion of the reviewed literature. Distributed development within USA has also been found popular.

Research methods: 88% of successful examples were case studies, while only 2% were based on an experiment. The research method could not be identified for 6% of the papers.

Contribution and Analysis:

- 70% of the papers were problem reports and lesson learned.
- 11% present recommendations for implementing the agile software processes in a global context.
- In 6% of papers, the authors present the best applied agile practices in their organizations.
- 4% of papers investigated industrial case studies and their analysis results.
- Implementations of tools to help Agile-distributed development were presented in 4% of papers.
- Finally, 4% focused on comparisons between performances of Agile and collocated development.

Details of successful cases: in answer to research question 2, in most of successful cases teams were distributed globally, working for long periods of time (>7 months) on small to med size projects (≤ 20 to ≤ 50 people). Results are summarized in Table 1.3.

Table 1.3 Location & Agile practices in Agile-Distribution settings

Agile software process Distribution settings	Most successful Agile practices reported	Location Location- Outsource	Authors' Notes
XP-Offshore	"Retrospectives"	USA-India	
XP- Outsource	"Continuous Integration", "unit/integration testing", "simple design"	USA-China	
XP-Distributed team	"SCRUM/ iterations" "Stand-up Meetings"	USA	Insufficient details
XP-Virtual team	"stand up meetings", "automated testing", "pair programming", "onsite/proxy customer" "enough documentation"		Only 1 paper. Countries not specified
Scrum-Offshore	"Sprint/ iterations", "Retrospectives", "Sprint review/demo"	USA-China USA-India	
Scrum-Outsource	"pair programming", "one team/sit together", "Scrum of Scrum" "Continuous/automated builds"	USA	
SCRUM-SCRUM-Distributed team	"Stand-up meetings" "backlog".	USA	
Agile-Offshore	"Spring planning"	USA	
Agile- Outsource	"Continuous Integration"	Denmark, Russia	
Agile-Distributed team	"Stand-up meetings" "Sprint/iterations"	India	
Agile- Open source	"Stand-up meetings" "pair programming" "sprint/iterations" "test-driven development" "unit/integration testing"	Italy, Norway	

The authors devised a ranking system for measuring success rates and location rates. If a specific agile practice is reported in N projects, each success-or specified location- counts for 1nth. For example, if 1 success and 1 failure are reported for an Agile-GSE combination, the success rate is 0.5.

Successful Agile practices: The authors highlight that "standup Scrum meetings", "Sprint/iterations", "continuous integration" and "sprint planning" are the most used agile practices .

Efficient Agile method distribution type combination: The authors identify Extreme programming- globally distributed team as the most used agile software process in the context of GSE. The authors list all identified combinations as follows, ranked by their frequency of usage:

1. XP–Distributed team: 9
2. Agile–Offshore: 7.5
3. Scrum–Distributed team: 7
4. Scrum–Offshore: 6.5
5. Agile–Distributed team: 6
6. Scrum–Outsource: 4.5
7. XP–Offshore: 3
8. XP–Outsource: 3
9. Agile–Open source: 2
10. Agile–Outsource: 1.5
11. Agile–Virtual team: 1
12. XP–Unclear: 1
13. XP–Virtual team: 1
14. Pair programming–Distributed team: 1

The authors highlight the insufficiency in some research methods, analysis of challenges and obstacles in Agile-GSE combinations, and observed repetitions in reviewed literature. Based on these observations, the authors suggest additional evaluation & validation research, and that joint analysis of challenges in Agile-GSE combinations between academia and industry are needed. The authors also suggest establishing a universal database of reports by practitioners as an aid for this purpose.

1.5.2 Diaz *et al.* (2011)

1.5.2.1 Goal of this study

Diaz, Pérez *et al.* (2011) carried out a systematic review of existing research on the integration of Software Product Line Engineering (SPLE) and Agile Software processes in what is known as Agile Product Line Engineering (APLE).

Agile software processes and practices are flexible in response to unpredictable changes, rather than being limited by rigid plans. Agile software processes exploit change for the competitive advantage of customers. However, scalability of agile software processes, especially in large software line projects, poses challenges in its application.

The development of SPLE consists of two phases: Domain Engineering (DE), and Application Engineering (AE). DE consists of creating reusable assets. DE determines the scope of the SPLE and handles commonalities and variability. AE consists of developing products through systematic reuse of core-assets by knowledge of commonalities and variability points.

To carry a systematic literature review on APLE the authors identified important challenges on integration of the SPLE model with agile software processes. The authors found only 39 studies directly related to APLE.

1.5.1.2 Methodology of this study

The authors point out that the methodology execution is nonlinear, involves iteration, feedback and refinement. The authors defined these phases in their methodology:

Planning the review

The review objective was to identify current APPLE approaches and experiences. The authors developed a review protocol which defines search questions, search strategy, evaluation criteria and methods of data extraction and evidence synthesis.

Research Questions:

- RQ1: What are the reasons, and when is it advantageous combining SPLE and agile software process?
- RQ2: How do the principles of SPLE and agile software process match?
- RQ3: How is APPLE positioned with respect to business strategic objectives?
- RQ4: Which current approaches combine SPLE and agile software process satisfying AE activities?
- RQ5: Which current approaches combine SPLE and agile software process satisfying DE activities?
- RQ6: What are the challenges and gaps in current APPLE during DE activities?
- RQ7: Which current APPLE approaches satisfy both DE and AE activities?
- RQ8: Are there successful industrial experiences putting APPLE into practice?

Search Strategy

The authors defined the search space and search strings. The search space included electronic databases (ACM Digital library, IEEE Xplore, SpringerLink, EI Compendex, Inspec, ISI Web of Knowledge, ScienceDirect), and conference proceedings (SPLC (Software Product Line Conference), XP (Extreme Programming), Agile Conference, APPLE (Workshop on Agile Software Product Line Engineering)). The search was conducted to extract a collection of primary studies. The review material included primary studies, references in primary studies, and other possibly relevant works of their authors, obtained through direct contact by e-mail and DBLP searches.

For the search of electronic databases, the authors defined two sets of keywords: a set included keywords of agile software process and the other set included keywords of SPLE. Acronyms, synonyms and abbreviations of the keywords were included in the search. The search strings were OR & AND combinations of these keywords.

Evaluation Criteria

The authors included scientific material (papers, experience reports, summaries of workshops, panels and poster sessions) written in English, produced until June 2010. The authors excluded studies that include Agility as a synonym for flexibility; and studies with ‘poor arguments’, which they defined as based on general opinion or poor arguments.

For assessment of studies quality, the authors adopt the quality criteria defined for the Critical Appraisal Skill Program (CASP) and the criteria proposed by Dyba and Dingsøy (2008). The authors pinpoint the main issues covered by the criteria: rigor, credibility, and relevance.

Data extraction

The authors stated they developed forms to store key concepts of the objectives, findings and conclusion of each study, in answer to their research questions. In their synthesis of evidence, the authors organized the key concepts and findings, for comparisons across studies.

The search produced over 536 primary studies. Secondary searches added 32 citations increasing the number to 568. The authors excluded duplicate studies reducing the number to 370 studies. Applying the evaluation criteria, authors shortlisted 52 studies, only 39 of which passed the authors’ quality assessment.

1.5.2.3 Findings of this study

In this step, the authors provide answers to their research questions.

RQ1: What are the reasons, and when is it advantageous combining SPLE and agile software process?

The main reasons concluded by the authors from several studies are:

1. To cut down long term investment in the DE phase: upfront long term investment provides flexibility in SPLE; however, it is resource-consuming and risky since these long term investment products might be outdated.
2. To deal with volatile business situations: when market stability decreases, SPLE design and long term lack flexibility. Agile software process has been found as a promising alternative.
3. To deal with lack of required knowledge about DE: as SPL developers might lack the required knowledge for DE and prediction of future changes, agile software process small-scale, iterative approach, is not dependent on long term predictions of market conditions, offers a solution and reduces risks.

RQ2: How do the principles of SPLE and agile software process match?

Various studies compared SPLE and agile software process, and found that both pursue common goals but using different strategies. While SPLE emphasizes change prediction and architecture definition, agile software process emphasizes incremental development and close iterations with customers. Agile software process promotes minimal investment in upfront design and architecture. The authors found studies which present similarities and differences between the two approaches; and some studies map and tailor both approaches together. Focusing on the principles, the authors suggest the possibility of SPLE-Agile integration, to analyze the most significant among commonalities in a family of products to meet changing customer requirements.

RQ3: How is APLE positioned with respect to business strategic objectives?

The authors found a number of studies that tackle this question. While some studies recommended using SPLE in strategic level and agile software process for technical level, some studies call for adaptation of agile software process on the strategic level for more flexibility.

RQ4: Which current approaches combine SPLE and agile software process satisfying AE activities?

The authors found papers that explain the importance of release matrix and Configuration Management (CM) in integration of SPLE and Agile. The authors review a number of studies of different approaches, and contributions in identifying challenges.

RQ5: Which current approaches combine SPLE and agile software process satisfying DE activities?

The authors pointed out the implementation of mechanisms to combine SPLE and agile software process in order to satisfy DE activities such as: mechanisms for supporting effort estimation, traceability, and synchronization between platform and product teams (DE and AE teams, respectively).

RQ6: What are the challenges and gaps in current APLE during DE activities?

The authors pointed out some attempts to address applicability of APLE in DE. The authors provide explanation of these attempts, but also notice the lack of findings, conclusions and clear explanations of models and implementations.

RQ7: Which current APLE approaches satisfy both DE and AE through Agile principles?

The authors pointed out the challenge inherent in combining SPL & Agile in both AE and DE. The authors notice that a single interesting study discusses agile organization applying SPLE, while all other studies introduce Agile into SPL for flexibility. The study presents a bottom-up approach and iterative design of SPL. The authors notice that further work is needed in this regard.

RQ8: Are there successful industrial experiences putting APLE into practice?

The authors present detailed reviews of some empirical studies which introduced examples of SPL-Agile integration in enterprises and organizations. The authors highlight the findings, conclusions and recommendations of these papers.

From their review, the authors highlighted the variety of reasons for combination of SPL and Agile, and the need of further work in APLE practice. The authors identified advantages of APLE application as described next.

- Agile software process may be used when SPL developers lack the knowledge in DE. Iterative APLE can be used in a rapidly changing product lifecycle.
 - Agile software process improves feedback between design phases of SPLE.
- The authors categorized about 30 studies on application of APLE, depending on the area (DE or AE), and challenges faced/solved. The authors categorized contributions and approaches as follows:
1. Five studies address the challenges of agile software process and SPL combination, and provide industrial case studies in applying XP and SCRUM in SPL.
 2. Three studies address support mechanisms for agile software process such as requirements change management, features modelling, product-line scoping and requirements engineering. The authors pointed out a lack of studies on traceability management in process activities such as requirements and design.
 3. Two studies address the challenges of SPLE architecture to support agile software process incremental design, but do not provide case studies.
 4. Five studies present successful industrial case studies, tool support and specific activities for the entire APLE process.

The findings suggest that further research is required in agile-SPLE integration and framework developments. Results highlight the feasibility of APLE in AE. Twelve papers

address APLE application in AE and various activities. Four papers identify challenges in synchronization between platform and product teams, and definition of traceability among SPL artifacts.

The authors pointed out a lack of evidence on applicability of agile software process in DE activities. 15 papers provide solutions in DE activities. The authors highlight the existing challenges in APLE architecture and traceability. The authors also suggested quality specification and mechanisms for trade-offs between SPL upfront long-term design and agile software process.

1.6 Statistical evidence on agile software adoption

This section presents a summary of the recent industrial studies and research papers that investigated the deployment of agile software process in software organizations.

1.6.1 Industry based evidences

An online survey conducted by MethodsAnd-Tools.com in 2005 to provide information about the rate of adoption of agile development indicates about 40% of the 232 participants organizations had adopted agile software processes and another 20% were evaluating them in pilot projects to evaluate their capability for future adaptation.

The same survey conducted on February 2008 with 502 participants, comparing the 2008 and 2005 results, indicates that the level of the agile movement had increased and only 13% of the organizations were not aware of any agile software practices. Full deployment numbers had doubled in 2008 to reach 17%, compared to 8% in 2005 and the total rate of various adoption levels is 56% compared to 41% in 2005. The conclusion drawn is that the importance of the agile approach is growing in the software development organizations and many software organizations are moving to deploy agile software process instead of traditional software model.

IBM Rational Methods Group conducted a survey on March 2006 to analyze the status of different agile software processes such as XP, SCRUM, FDD and DSDM. The survey made

online was based on a previous survey made by Shine Technologies. The survey is not limited to one geographical location to collect evidences from global and local software organizations to analyze their adoption as well as their understanding to agile software practices. There were 4232 participants, divided based on the size of the software organizations as of table 1.4.

Table 1.4 IBM rational methods group survey

Size	# Respondent	Percentage
1-10 people	1353	32%
11 to 50	877	21%
51 to 100	422	10%
101 to 200	332	8%
201 to 500	310	7%
501 to 1000	232	5%
1000 to 2000	142	3%
2000+	564	13%

Figure 1.3: presented the answers of the participants to a multiple choice question: “which agile process your organization is using”?

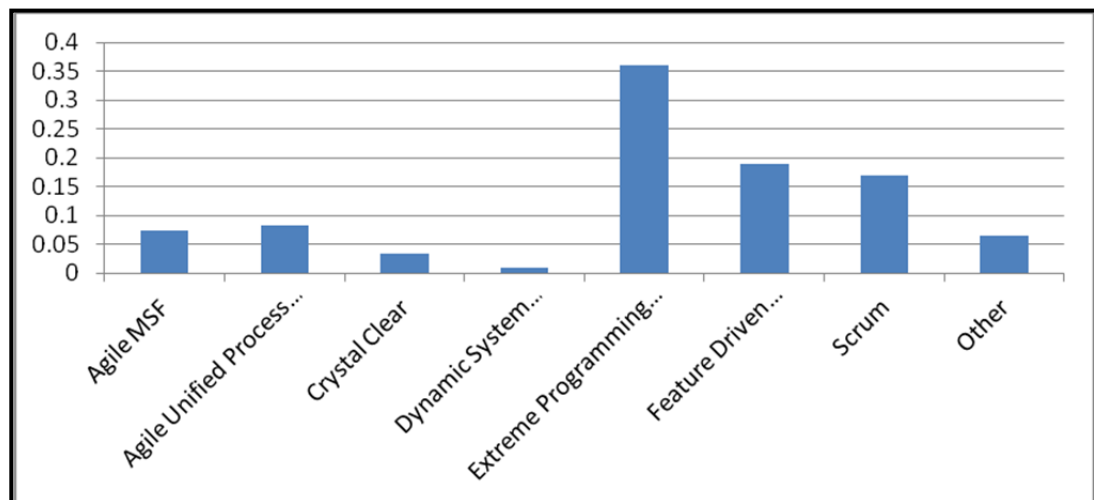


Figure 1.3 The extent of use of different agile process reported in Ambler (2006)

Figure 1.3 shows that the XP was the most widely used at almost 40%. The IBM Rational Methods Group report states that “it is no surprise that XP and Scrum are popular options”.

1.6.2 Academic based evidences

Vijayasarathy and Turk (2008) reports on an online survey to find the percentage of adoption of agile software process and to investigate the factors that influence their adoption as well as to determine the agile software processes that are commonly in use. Data were collected from Yahoo discussion groups that focus on agile software process and the feedback represents a sample from 17 different countries (USA, Canada, India, United Kingdom, Australia, Colombia, Mexico and New Zealand, etc.). There were 198 participating software professionals with an average of 15.5 (median = 15.0) years of experience with software development and 3.9 (median = 3.0) years of agile experience. The survey findings can be summarized as follows:

- 90% of the participants of this survey had a basic understanding of agile development practices and 81% were either using or planning to use agile methods in their organizations.
- XP is reported to be used the most extensively, ranking 5.4 on a 7-point scale. Followed by Scrum and Agile Modeling with rankings of 3.5 and 3.4, respectively. AUP (the Agile Unified Process) came in last with a ranking of 1.9.

Schindler (2008) conducted a study to analyze the responses of a total set of 400 software development organizations. The organizations were classified based on team size (micro, small, medium and large). The distribution of the participating organizations is: 19.0% micro, 28.6% small, 11.9% medium and 40.5% large organizations. The survey started on 29th of July and ended on 25th of August 2008 and was conducted via telephone. The main findings of this study can be summarized as follows:

- A majority of the interview participants (77%) claimed to have a basic understanding about agile software development methods.

- When participants were asked to name the agile process used by them, Extreme Programming (XP) was mentioned by 46% and Scrum by 32.8%. The sample indicates that XP was used by 53.2% of the developers and by 38.9% of the project managers. Scrum was mentioned by 29.8% of the developers and 33.3% of the project managers.
- 44.3% of the total participants, (42.6%) developers and (52.8%) managers, mentioned that their organization was trying to adopt agile software development.

All the mentioned industrial surveys and papers concur that some software organizations have successfully deployed agile software process such as XP or some practices of agile process such as pair programming, agile modeling, test driven development, etc. The above studies have different viewpoints i.e. industrial viewpoints or academic viewpoints. Table 1.5 shows the main strengths and weaknesses identified in each of these surveys.

Table 1.5 Comments on the agile surveys

Survey author	Comments
Methods And-Tools (2005, 2008)	Little information is provided regarding the methodology used for collecting and analysing the obtained data. The survey is biased to sample of software organizations that already have interest in agile software development. Less information is given regarding the essence and the quality of sample that has been conducted.
IBM Rational Methods Group (2006)	Data has been collected using online multiple choice questions. Little information is provided regarding the expertise of the participants. Little information is provided regarding the size and the complexity of the projects that have used the agile software processes.
Vijayasathya (2008)	The data was collected using yahoo discussion groups that focused on agile software processes. The survey is biased to a sample and participants who most likely have an expertise in agile software processes.
Schindler (2008)	The data represent the response of one geographic area i.e. Austria

1.7 Agile software processes

In the next sub sections, a description of agile processes is provided with the discussion focused on Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Adaptive System development (ASD) and Crystal methodologies.

1.7.1 Extreme Programming (XP)

Extreme Programming (XP) is one of the first proposed agile processes. The XP process was proposed in 1996; XP is an incremental approach that mainly focuses on the most important parts of the product, as defined by the client (Abrahamsson, Salo *et al*, 2002).

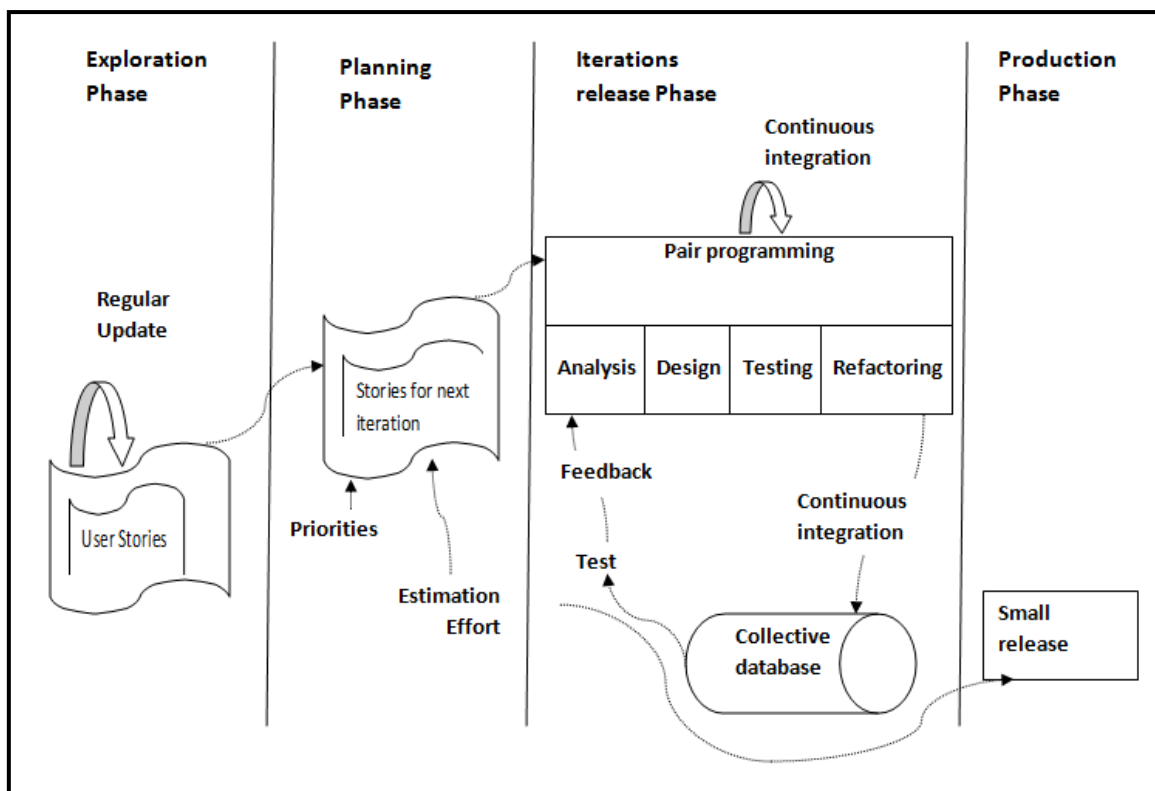


Figure 1.4 Extreme programming - XP

Figure 1.4 illustrates the set of practices which in total creates the XP process. The XP process starts in the exploration phase where the development team starts collecting the

requirements and writing them down in cards called the user stories. In this exploration phase, the team studies the feasibility of the whole project, selects the suitable architectural design, and selects tools and languages that will be used during the project. This exploration phase is followed by a planning phase, sometimes referred to as planning the game. During this planning phase, estimation of the project schedule is performed, and a customer evaluates the written stories and prioritizes them for the coming releases. The XP tracker and the coach then estimate the time and effort needed to complete the first release (Abrahamsson, Salo *et al*, 2002).

Analysis is done at the beginning of each iteration by generating an iteration plan. Customer prioritizes the chosen user stories for the current release and the upcoming iterations based on business understanding. After that, the programmers break the user stories down into a number of tasks and estimate the required time and resources for each task. During each iteration, the programmers reprioritize the user stories considering technical factors, and create the design which should be as simple as possible for the current iteration.

The programming in XP is done in pairs; it is more a hybrid design, programming, testing rather than pure programming, during which one programmer is writing the code and the other is reviewing the code consistency. Refactoring then is performed to ensure that the code is robust and optimized as much as possible (Abrahamsson, Salo *et al*, 2002). Since the code in XP is a collective ownership, refactoring can be done by the same pairs or by a different pairs with an objective to simplify and improve the code internal structure without changing the code functionality.

XP is a test driven development method that relies on a repetition of a very short development cycle. XP requires writing a test case for each story to define an improvement to the implementation that can finally pass the test case.

XP requires two other types of testing; integration testing which is done at the end of each iteration, by integrating the result of the last iteration with the previous ones, and then

ensuring that the overall system is bug free. Finally the acceptance test is done by the customer to ensure the correctness of the stories and the whole system (Abrahamsson, Salo *et al*, 2002).

Using XP, the resulting implementation is owned by all team members. This collective ownership of the artifacts of the system allows the programmers to make modifications to parts of the code that have been created by others. The main advantage of this practice is to speed up the development process such that when programmers detect a fault in the code they have the right to fix it. A coding standard is used to make sure that the development team uses the same design and coding conventions (Abrahamsson, Salo *et al*, 2002).

1.7.2 Scrum

Scrum is an iterative incremental framework for managing software system development and each system feature is delivered in 30 day sprints. It inherits many of the features of the traditional iterative and incremental approaches.

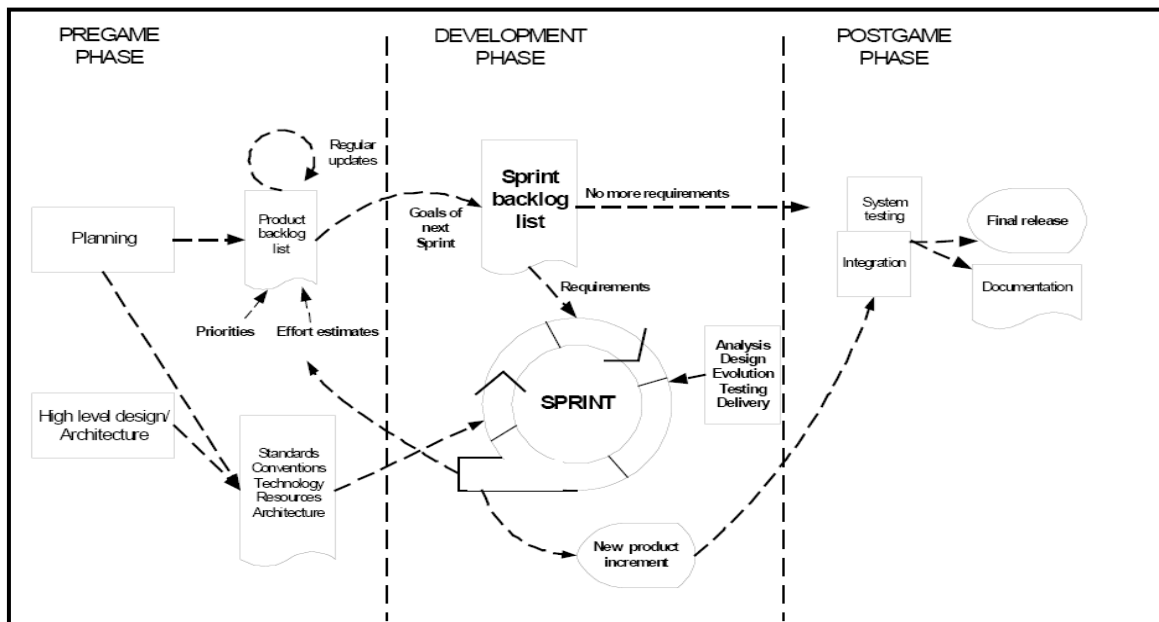


Figure 1.5 Scrum process
Cao (2006)

Figure 1.5 illustrates the Scrum process. The Scrum process starts with a planning phase, during which a backlog list is developed to define the functionality of one or more releases of the system along with the risks associated with each release. The appropriate risk controls are also determined. The product backlog lists contain the total work of the project to be done. After that, a sprint planning meeting takes place. It usually starts every 15 to 30 days after the planning phase. During this meeting, customers, users, managers, and developers discuss the objectives of the next sprint release and the sprint backlog lists to be completed.

One of the practices that is required by Scrum (and that many agile teams are adopting) is the short daily meeting. During this meeting the team discusses the progress each team member has made since the last meeting and the impediments or problems that have been identified. Scrum recommends other interested stakeholders to attend the meeting, but prevents them from speaking in those meetings. This is done to keep the meeting short. The project is led by a Scrum master who is often a project manager and whose job is to remove all impediments the team identified during Scrum meetings.

One of the important aspects of Scrum is continuous integration of project artifacts and test code coverage. Also the Scrum methodology is based on regular review sessions after completion of each sprint to discuss the project progress (tasks finished, impediments and product backlog) with the project manager and the customer. Those sprint review sessions are used to provide progress feedback to various stakeholders involved in the project Abrahamsson, Salo *et al.* (2002).

1.7.3 Feature-Driven Development (FDD)

The Feature-Driven Development (FDD) approach focuses on the software features of the system as the main driver of the development process. It differs significantly from the other agile processes by putting a strong emphasis on planning and upfront design. Those designs became primary driver for the rest of project life-cycle.

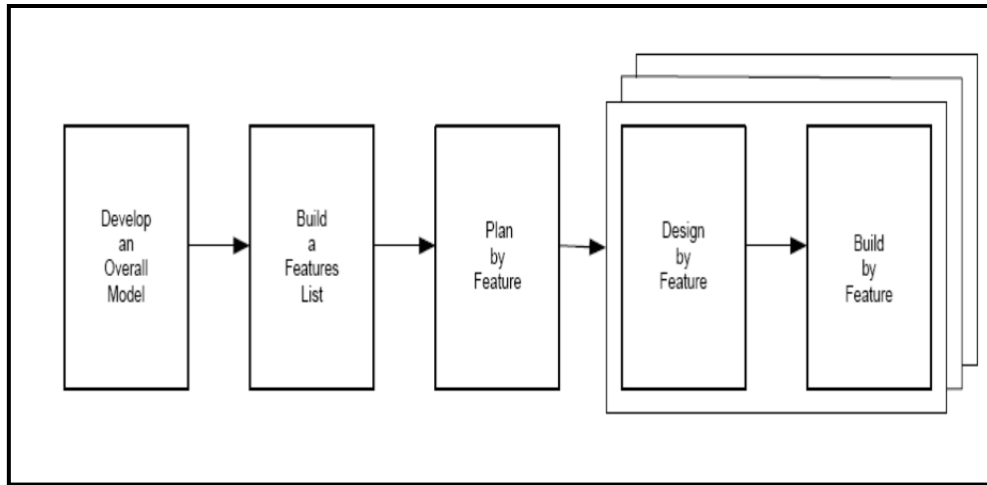


Figure 1. 6 Steps of FDD
(Cao 2006)

Figure 1.6 illustrates the FDD lifecycle. The first step of FDD is to develop a model of the application based on stakeholder's assumptions, requirements and desired quality characteristics. The next step is to create a feature list. Each feature should be small enough to be implemented fast (from few hours up-to a 2 weeks).

Each team is typically working in parallel and once the feature is complete, tested and verified, the team is disbanded. In contrast to Extreme Programming where the whole team owns all the features of the project, FDD assigns a feature to a "feature owner" who acts as team leader and is responsible for the code that implements the feature.

The FDD process utilizes rigorous inspection guidelines in order to find defects in the system. It also enforces coding standards. It also encourages regular builds on a daily or weekly basis in order to add newly designed features to the baseline system. Since features are developed in parallel, it is important to have a configuration management system that allows proper integration of the changes made to the system.

One of the unique aspects of FDD is how it manages and tracks feature completion process and the status of project. Project progress is measured on the number of designed, implemented, verified and tested features and each feature is measured based on its score.

The score is computed by assigning a completion status, ranging from 0 (yet to be implemented) to 1 (feature complete, verified, tested and integrated).

1.7.4 Adaptive Software Development (ASD)

Adaptive software development (ASD) grew out of rapid-prototyping and is defined as “a complex adaptive process that involves interaction between agents (stockholders), environment (organization) and the product (software)” (Abrahamsson, Salo *et al*, 2002).

ASD is based on continuous adaptation of the process – a methodology that accepts the continuous change as a norm. The ASD process consists of three lifecycle phases: Speculate, Collaborate and Learn. ASD is a dynamic lifecycle that ensures teams are constantly learning, changing and adapting to the emergent state of the project. The Speculate cycle is a planning phase where the team decides what items they should work on. The objective of collaboration cycle is towards transfer of knowledge between software developers. The Learning phase is carried out after each iteration in order to improve the developer’s expertise as well as to enhance the quality of the work.

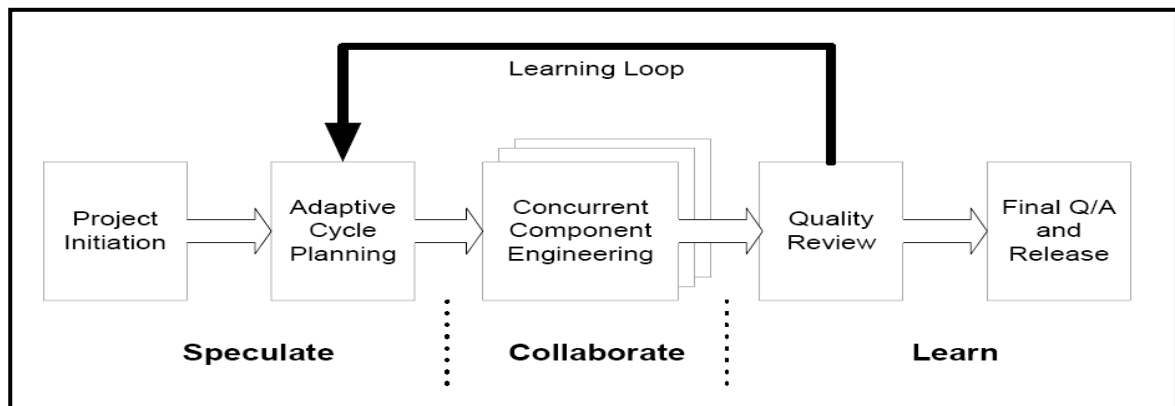


Figure 1.7 ASD process activities
Cao (2006)

Figure 1.7 depicts the main phases of the ASD process. ASD starts with a project initiation phase. During this phase the project mission statement is established, which is defined to

guide the overall process. It must be clear and well organized. The project normally starts with unclear requirements but after each adaptive loop the overall mission becomes clearer.

ASD is a feature-oriented approach rather than task-oriented. The main focus is always on the features of the systems rather than the tasks needed to implement these features. During the concurrent component engineering phase, the developers may work in parallel to implement one or more features at the same time. One of the most important aspects of ASD is the quality review phase where the customers, developers and managers meet to discuss and assess the overall quality of the work performed. The review phase session, known as the joint application development session (JAD), is important for demonstrating the functionality of the system developed as well as to keep the project within the boundaries of the mission statement. Finally, a quality assurance and release phase is held at the end of the project to fix all problems regarding the quality of the work performed.

1.7.5 Crystal Methodologies

The Crystal methodologies are a set of processes that can be applied to different projects depending on the size and the complexity of a project. The framework in Figure 1.8 includes the factors that influence the selection of a particular methodology. The X-axis indicates staff size while the Y-axis represents the system criticality. The more critical the project, the more rigorous and formal processes are required. Crystal methodologies define four levels of critically:

- Life (L): A system failure is critical and may cause loss of life.
- Essential money (E): A system failure may cause loss of money.
- Discretionary money (D): A system failure may cause loss of money but can be fixed by referring to the system's user manual.
- Comfort (C): A system failure may cause a loss of customer comfort.

Crystal methodologies put an emphasis on a set of policy standards that govern the way the project is managed. These standards are common among all crystal methodologies and

include incremental delivery of releases, progress tracking, direct user involvement (Abrahamsson, Salo *et al*, 2002).

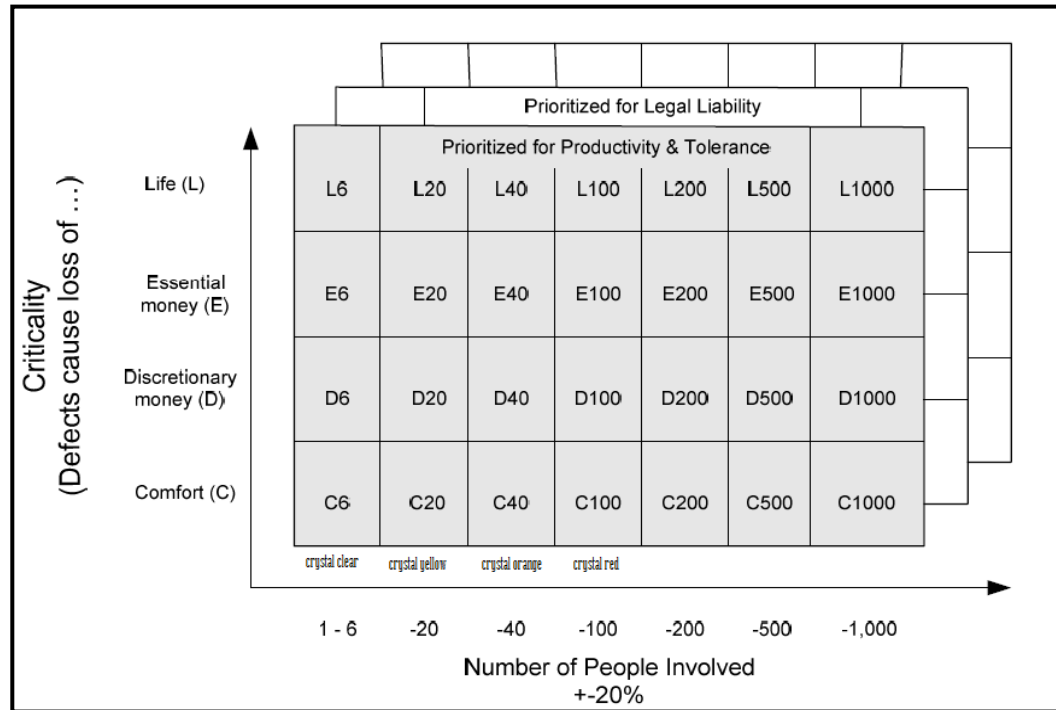


Figure 1.8 Crystal methodologies
Cao (2006)

Different processes are assigned a different color that represents the heaviness of the process. Currently two crystal methodologies have been defined: Crystal clear and Crystal orange. Crystal clear is designed for small projects with a maximum of six developers as shown in figure 1.8. It can be used for different levels of criticality. For example, the D6 category indicates the use of the crystal clear with a critical level of discretionary money.

The developers should be located in a shared space to improve the communication between them. Developers can use any tool to improve the overall work: in other words, Crystal clear keeps the choices open for developers to choose the appropriate tools. The documentation in Crystal clear is very light. The requirements are expressed using UML use cases. The first

incremental cycle must not exceed three months and a workshop meeting is usually held after each delivery.

Crystal orange is targeted for a project with a maximum of 40 developers. The project duration is usually between one to two years. The Crystal orange methodology is suitable for a project of category D40 and may extend to E40 if necessary. Due to lack of rigorous verification techniques, the crystal orange methodology is not appropriate for life critical projects. Similar to crystal clear, developers are encouraged to work in a shared space. Crystal orange requires more documentation than Crystal clear. For example, the requirements should be expressed in a natural language and the design documents are expressed using formal specifications such as state chart diagrams. The first incremental delivery must not exceed four months and more formal testing methods are encouraged in Crystal orange.

1.8 Comparison of agile processes based on software design and projects requirements

This section discusses how different agile software processes address common software project requirements – See table 1.6.

1. Customer Involvement

Customer involvement is a key practice in all agile processes, as shown in table 1.6. Agile processes consider customers as an integral part of the development process. For example, XP, Crystal methodologies, and Scrum require on-site visits to customer's venues to allow end users to verify and prioritize the requirements during the requirements phase. The involvement of customers is also reflected during acceptance testing, where most agile processes require these tests to be written and executed by customers.

2. Documentation

The agile processes studied in this report vary in the level and the type of documentation they provide. For example, XP uses user stories to capture the software features that need to be implemented. Scrum's main documentation consists of product and sprint backlog lists. FDD and Crystal methodologies use UML diagrams such as use cases, class diagrams, and object models to document the design. Test cases have also been used by XP and Crystal methodologies as documentation artifacts – See table 1.6.

3. Verification and Validation

Every agile process places strong emphasis on the correctness and quality of the software artifacts. But methodologies differ in how much verification is required and what validation and verification activities to perform. This allows teams to choose a methodology that would satisfy customer's quality requirements. If the application is used in safety critical environment such as medical, industry safety monitoring or military, then more extensive testing would be performed.

Automated verification and validation with unit tests is used in agile methodologies to check that the software product meets requirements and specifications and that it fulfills its intended purpose. Some methodologies like XP are test driven such that each release should pass test cases that are developed to improve the release functionality. In addition to unit testing, regression testing is used in Scrum.

Other quality review techniques are also used; for example FDD requires design and code inspections. Scrum requires a sprint review in the end of each iteration, and ASD recommends to do code quality reviews - See table 1.6.

4. Team Management

Team management is important for organizing the team from many perspectives, such as team size, team communication and the use of standardized procedures (e.g., design conventions), etc. Team size is one of the important factors that may affect the selection of the development process. Although agile processes emphasize a face-to-face communication instead of formal documentation, the number of developers considered is a serious obstacle to the effectiveness of the communication. Except Crystal orange, all other agile processes suggest at most 20 persons per team - See table 1.6.

Table 1.6 Comparison of agile processes based on the project and design requirements

Agile Process	Customer Involvement	Documentation	Verification and Validation	Team Management
XP	User stories written by customer who is part of the project team	User stories Test cases Acceptance test	Test driven - development Unit testing Integration Testing Acceptance Testing	1 – 10 teams 5 -9 people per team Coding standards is mandatory
Scrum	Backlog written by a customer who is part of the project team Review meeting with customer presence	Product backlog list Sprint backlog test	Sprint review Unit testing Integration testing Regression Testing	1 – 5 teams 16 -20 people per team
FDD	Customer only reviews the feature list.(Does not require a customer on the project site)	Overall model design User cases and class diagrams List of features	Design Inspection Unit Testing Code Inspection	1 – 10 teams 2 -4 people per team
ASD	Customer presence at frequent quality review meetings Quality review phase	Project data sheet Project outline	Quality review	1 – 20 teams 10 - 40 people per team
Crystal Clear	Direct user involvement Short release	Test cases User model Object model	Automated unit test Automated regression test	1 –10 teams per project (2-4 per team)
Crystal Orange	Direct user involvement Short release	Object models User manual Test cases Feature description	Automated regression test Formal testing External testing	Coding standards is mandatory 1 –20 teams per project (10-40per team)

Team communication is considered as the second factor in team management. Agile processes tend to be people-oriented processes by allowing team members to take appropriate decisions when required without being restricted by any procedure or technique.

The use of code standard guidelines has been proposed in XP and Crystal methodologies to facilitate exchange of information among team members. This facilitate that these processes favor collective ownership of the system artifacts. In other words, any member can modify the code or design of someone else. In such cases, standard coding guidelines facilitate the collaborative work.

1.9 Experiments classification of agile literature

This section provides an experiment classification for twenty papers published in the area of agile software process development and improvement from 2001 to 2009. The classification is based on the approach of Zelkowitz and Wallace (1997). In 1997 Zelkowitz mentioned that “Computer science is a relatively new field, with most academic departments formed during the late 1960s and 1970s. A strong experimental model of the field has not developed; at least as computer science folklore explains it”.

The research of agile software process improvement and development has started after the “Agile Manifesto” introduced in 2001. This section provides classification for some of the experiments in the area of agile process. Zelkowitz and Wallace (1997) classified the research experiment models in the area of software engineering into three main categories.

- **Observational method:** Researchers collect the relevant data as a project develops. There is relatively little control over the development process. This method consists of multiple models: project monitoring, case study, assertion, and field study.
- **Historical method:** Researchers collects data from projects that have already been completed. The data already exists; it is only necessary to analyze what has already been collected. This method consists of multiple models: literature search, legacy data, lessons learned, and static analysis.

- Controlled method: Provides multiple instances of an observation in order to provide statistical validity of the results. This method consists of multiple models: replicated synthetic environment, dynamic analysis, and simulation.

Table 1.7 summarizes and classifies the experiments of the selected papers based on the categories of Zelkowitz and Wallace (1997) and the following comments can be made:

The research model will be classified as assertion if the researcher provides no experiment to support his/her conclusions or the researcher provide only preliminary test before a more formal validation for the research approach. An assertion experiment found in 4 papers out of 20. The assertion model found on papers #1, #2, #5 and #9.

For example in paper number #5 Nerur, Mahapatra *et al.* (2005) the authors discussed the possible challenges that software organizations could find in their process to migrating to implement the agile software development. The author provide no experiments to support his findings and conclusion: rather the discussion is based only on the author experience in the area of agile software processes. The research model will be classified as a case study if the experiment specifies the monitoring and the data collection technique over the time of the project development. The main characteristic of this experiment model is that that the project is to be undertaken whether data is to be collected or not. "With a relatively minimal addition to the costs to the project, valuable information can be obtained on the various attributes characterizing its development" Zelkowitz and Wallace (1997). Case study experiment is found in 7 papers out of 20. The case study model is found in paper #6, #7, #10, #15, #16, #17, #20. For example the authors of paper #6 Canfora, Cimitile *et al.* (2005) provide an experiment model to empirically study on the productivity of the pair programming in XP environment. The authors provide design for the experiment, data collection and analysis technique. The experiment was executed with the collaboration of students at master of technologies of software who require taking a programming course as a graduation requirement.

Table 1.7 Summarization and classification of the experiments

NO.	Authors	Publication year	Paper title	Publisher	Experiment type
1	Turk <i>et al.</i>	2004	Assumptions underlying agile software-development processes	Journal Of Database Management	Assertion
2	Conboy and Fitzgerald	2004	Toward a conceptual framework of agile methods: A study of agility in different disciplines	Proceedings of the 2004 acm Workshop on Interdisciplinary Software Engineering Research	Assertion
3	Wang <i>et al.</i>	2009	Where agile research goes: starting from a 7-year retrospective (report on agile research workshop at XP 2009)	ACM Software Engineering Notes	Literature Search
4	Chandra <i>et al.</i>	2009	Identifying some important success factors in adopting agile software development practices	Journal of Systems and Software	Field study
5	Nerur <i>et al.</i>	2005	Challenges of migrating to agile methodologies	Communications of the ACM	Assertion
6	Canfora <i>et al.</i>	2005	Empirical study on the productivity of the pair programming	Lecture Notes in Computer Science, Springer	Case study
7	Cockburn and Williams	2001	The costs and benefits of pair programming	Extreme Programming Examined, Boston, MA: Addison Wesley	Case study
8	Lindstrom and Jeffries	2004	Extreme programming and agile software development methodologies	Information Systems Management	No experiment
9	Maurer and Martel	2002	Extreme programming: rapid development for web-based application,	IEEE Internet Computing	Assertion

Table 1.7 Summarization and classification of the experiments (Continued)

NO.	Authors	Publication year	Paper title	Publisher	Experiment type
10	Boehm and Turner	2003	Using risk to balance agile and plan-driven methods	IEEE Computer Society	Case study
11	VIjayasarathy and Turk	2008	Agile Software development: A Survey of early adopters	Journal of Information Technology Management	Field study
12	Salo and Abrahamsson	2008	Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum	Institution of Engineering and Technology Journal	Field study
13	Schindler	2008	Agile software development methods and practices in austrian IT-industry results of an empirical study	International Conferences on Computational Intelligence for Modelling, Control and Automation	Field study
14	Wright	2003	Achieving ISO 9001 certification for an XP company	Lecture Notes in Computer Science	Project monitoring
15	Vriens	2003	Certifying for CMM level 2 and ISO9001 with XP@Scrum	Proceedings of the Conference on Agile Development	Case study
16	Alegria, and Bastarrica	2006	Implementing CMMI using a combination of agile methods	CLEI Electronic journal,	Case study
17	Stelzer <i>et al.</i>	1996	Software process improvement via ISO 9000? results of two surveys among European software houses	Proceedings of the 29th Hawaii International Conference on System Sciences	Case study
18	Dybå andr Dingsøyr	2008	Empirical studies of agile software development: A systematic review	Information and Software Technology	Replicated Experiment.
19	Chow and Cao	2008	A survey study of critical success factors in agile software projects, Journal of Systems and Software	Journal of Systems and Software	Field study
20	Rumpe and Schröder	2002	Quantitative survey on extreme programming projects	International Conference on Extreme Programming and Flexible Processes in Software Engineering	Case study

The students have different scientific background (engineering, mathematics, and physics). The course provides the basic education in computer engineering (operating systems, programming languages, network, database, and software engineering) and the students attend theoretical classes and lab sessions, they develop a large and complex project in connection with software organizations.

The main different between a case study model and a project monitoring model is that data collection is focused on a specific goal for the project. A certain attribute is studied (e.g., reliability, cost) and data is collected to assess that attribute. This model is found on paper #14.

The main characteristic of the experiment model of field study is that an outside group will monitor the collection of relevant data. The data that can be collected is limited, but designed to achieve specific goals. Zelkowitz and Wallace (1997) mentioned that this model can be viewed as a cross between the project monitoring method, where any data is collected and the case study, where specific data is collected. Field study is found in 5 papers out of 20. The field study model has been found in paper #4, #11, #2, #13 and #19.

The main characteristic of replicated experiment model is that the authors can use well known method to design his/her experiment, beside the authors should design inclusion and exclusion criteria for the subject under investigation. Only one paper (i.e. # 18) was based on replicated experiment model. This model has been used in paper # 18 where the authors provide a systematic review for agile software development. The goal was to investigate what is known about the benefits and limitations as well the strength of evidences of agile software processes. The authors design this experiment based on the approach of systematic review in software engineering Kitchenham (2007). The authors have also developed an inclusion and exclusion criteria for the reviewed research data.

1.10 Summary

Agile software processes have been proposed to overcome the inflexibility of traditional software processes (e.g. waterfall process) which put an emphasis on fully elaborated documents as completion criteria for the requirements and design phases. Agile software processes are based on iterative and incremental development, adaptive planning and informal (i.e. face-to-face) communications rather than formal documentation. This chapter has analyzed several research studies and surveys related to the topic of agile software process and its implementation in software organizations. The following comments can summarize the findings of this chapter:

- Agile software processes differ from the traditional software process. The traditional software processes follow liners and strict order of the development activities such that the development team should be able to complete all the development activities for a certain development phase (e.g. the requirements phase) before they can move to the next development phase (e.g. the design phase). Agile software processes have been designed to shorten the software development lifecycle with small working deliveries that are fully functional and can be used before the overall project is complete.
- Many agile software processes have been proposed, such as: Extreme programming (XP), Scrum, Crystal Methods, Adaptive Software Development (ASD), Dynamic System Delivery Model (DSDM), and Feature-Driven Development (FDD). Based on the analysed literature in this chapter it has been found that XP is one of the most deployed and widely used agile software processes.
- Prototyping, iterative development, smaller team members, and direct involvement of the customer are among of the main similarities of the agile software process studied in this chapter. Some differences have been found on the team management of agile software processes. For example, XP consist of 1 to 10 teams and 5 to 9 developers per team, where Scrum consists of 1 to 5 teams and 16 to 20 developers per team.
- Several research papers have studied the improvement and the implementation of agile software processes for complex software project such as global project development and

software product line engineering.. A recommendation for focused research efforts and the development of tools to support the implementation of agile software processes in the context of large and complex software projects has been found in the literature.

- Less research papers have been found on the topic of the implementation of ISO 9001 in the context of agile software organizations. The lack of documentation for the agile processes, traceability analysis and planned activities for validation/verification and measurement process were among of the main drawbacks for agile software process to support the ISO 9001 requirements.

CHAPTER 2

ISO 9001 AND AUDITING PRINCIPLES

2.1 Introduction

ISO 9001 is a Quality Management Standard (QMS) that provides a set of generic quality requirements for the industrial organizations, and it can be applied to the software process life cycle Kevin (2003). Researchers Hass, Johansen *et al.* (1998), Ferreira, Santos *et al.* (2007), Makdee and Praneetpolgrang (2005) and Fuller (2006) have studied the implication of ISO 9001 certification on quality improvement for software organizations by implementing certain requirements throughout all the software development phases, like design, development, production, installation, and maintenance.

The term quality has been defined in IEEE-610.12:1992 as “The degree to which a system, component, or process meets specified requirements”. The same definition is also in ISO 90003:2008. The ISO 9126:2001 has defined the quality as the totality of characteristics of an entity that bear on its ability to satisfy stated needs. ISO 9126:2001 considers software product quality from three different viewpoints: internal quality, external quality and quality in-use.

- Internal quality is “the totality of the characteristics of the software product from an internal view”.
- External quality is “the totality of the characteristics of the software product from an external view”.
- Quality in-use is “the user’s view of the quality of the software product when it is used in a specific environment and a specific context of use”.

The quality of a software product is highly related to the process selected to develop the software product: for example ISO 9126 directly associates the software process quality and the software product quality as shown in figure 2.1. The quality of the software product is directly influenced by the quality of the process used to develop it. Another approach is defined by the Capability Maturity Model (CMM), which is in use in many organizations. The CMM provides a framework for process improvement that consists of "key process areas" influential in various aspects of the development process and resultant software quality.

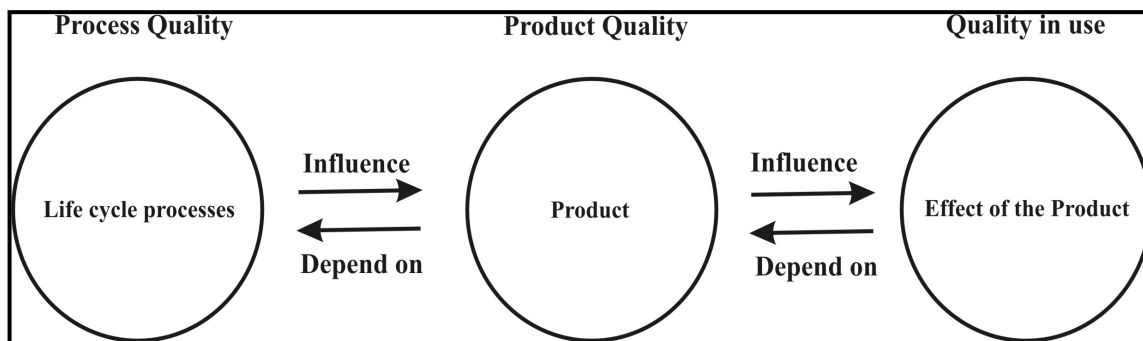


Figure 2.1 ISO 9126 quality approach

It has been indicated in Boehm and Turner (2003) that it is hard to find a general purpose software process in an organization and if the organization fails to identify the appropriate software process model this often results in customer dissatisfaction, flawed software products, projects over budget and overdue project completion.

This chapter is organized as follows:

Section 2.2 presents the results of survey evidences on ISO 9001 and software organizations.

Section 2.3 presents an overview of auditing practices.

Section 2.4 describes the main auditing activities to achieve certification for ISO 9001.

Section 2.5 provides an overview of ISO 9001.

Section 2.6 presents a summary.

2.2 Adoption of ISO 9001 in software organizations

This section focuses on summarizing the papers and industrial surveys that explore the impact of the adoption of ISO 9001 in software organizations. It will assist in gaining an insight about the implications of ISO 9001 certification on the software organizations.

According to a 1996 survey by Bradstreet and Irwin, 10,648 ISO 9001 certificates were issued in North America with ISO 9001-registered organizations in the software category which accounting for only 5.2 percent of the total number of registered organizations in that geographic area. Business services and electronic and electrical equipment were the two industrial sectors holding the most ISO 9001 certificates in the software category in 1996.

The research conducted by Hass, Johansen *et al.* (1998) provides information about the importance of ISO 9001 certification for the software organizations in order to improve the software development processes. The research is based on the BOOTSTRAP methodology and the CMM (capability maturity model). A numerical scale from 1 to 5 was defined based on CMM for determining the maturity level of the organizations processes. Through this methodology, 25 Danish owned software organizations operating in US were assessed during April 1996 to September 1997. The organizations have been divided into one group of 12 organizations with an ISO 9001 certificate and another group of 13 organizations without an ISO 9001 certificate.

The maturity level of the Danish organizations was evaluated in terms of the maturity levels scale based on CMMI - See figures 2.2 and 2.3.

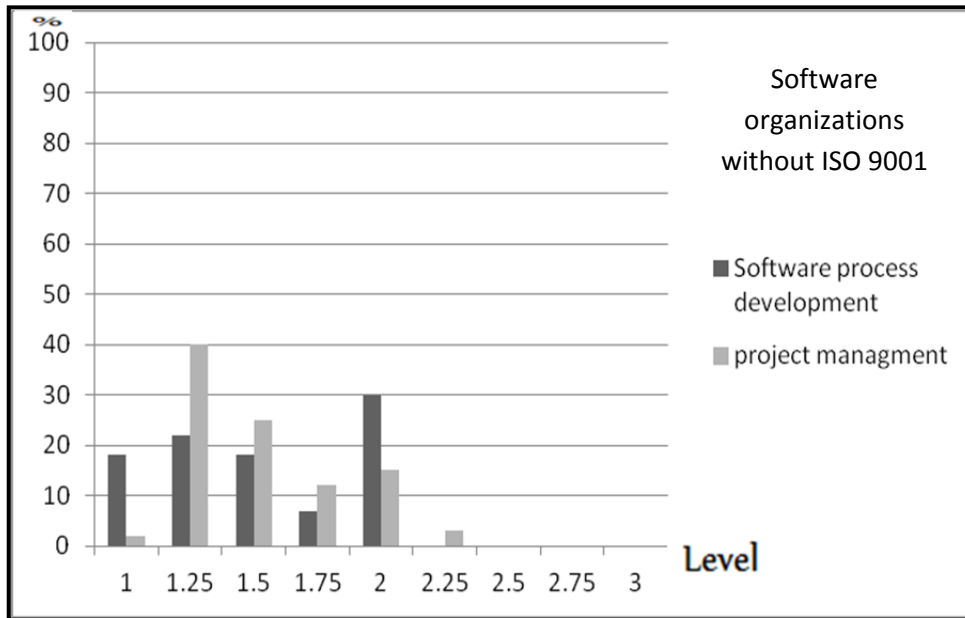


Figure 2. 2 Maturity distributions of Danish organizations without ISO 9001 certification, Hass, Johansen *et al.* (1998)

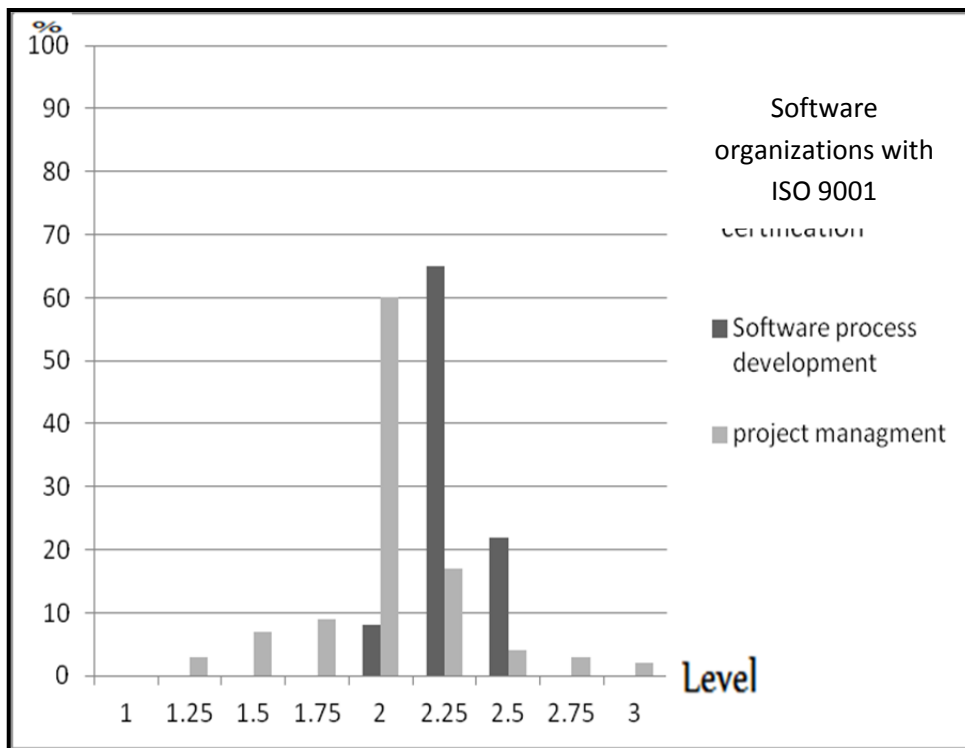


Figure 2.3 Maturity distributions of Danish organizations with ISO 9001 certification, Hass, Johansen *et al.* (1998)

- The maturity distribution of the organizations with an ISO 9001 certification has been improved. Along with this, it has been noted that ISO 9001 certified organizations have more employees specially those who are dedicated for quality assurance, process control and configuration and change management.
- Non-certified organizations are unable in reaching the maturity level of the certified organizations in both software process methodologies and project management.
- It was also observed that most of the organizations have maturity distribution approximately equals to 1.25 for non-certified with ISO 9001 and 2.5 for certified with ISO 9001.
- The data also reveals that the software quality management and other software related processes of the ISO certified organizations improved, compared with ISO non certified organizations (Hass, Johansen *et al*, 1998).

The research conducted by Ferreira, Santos *et al.* (2007) is mainly focused towards the improvement approach for the Brazilian company called BL Informatica with respect to the establishment of the software processes in compliance to the ISO 9001 standard and the maturity models, such as MPS.BR (Model for Brazilian Software Process Improvement) and CMMI (Capability Maturity Model Integration). The process of improving software development was evaluated in term of quality aspects, customer satisfaction, reduction of processing time and cost performance index, where the cost performance index is defined as the ratio of earned value to actual cost of work performed.

The improvement plan of the software processes was based on the international standards and quality models. The maturity levels were assessed based on CMMI and MPS.BR. The process area was represented based on ISO 9001. The return on investments for the projects was measured in quantitative terms. It reveals that (Ferreira, Santos *et al*, 2007):

1- During the first phase of the software improvement process, around 44% of the time allocated for the projects is expended on the rework and adjustment activities. Quality assurance framework reduced the rework time.

2- Implementing the ISO 9001 requirements enable the organization to reach CMMI level 3. A noticeable improvement was found in the software development methodologies - specially the verification and validation process. The development team was better aware of identification of defects during the product life cycle phase, which induced benefits for the organization.

3- The cost performance index (CPI) of the company has improved with the implementation of the different phases of the software processes as per the ISO 9001.

The research conducted by Makdee and Praneetpolgrang (2005) is based on the assessment of the effectiveness of the Thai software organizations certified to the international standards of ISO 9001: 2000 and TQS (Thai Quality Software). Field surveys were carried through interviews and a questionnaire accompanied by analysis of data from the software producers and software developers. The organization process effectiveness was assessed based on four factors: financial satisfaction, customer satisfaction, internal business process and learning growth. The interviews were conducted with 56 organizations producing and developing software (23 organizations having ISO 9001: 2000 certification and 33 organizations having TQS certification). The effectiveness of the organizations having ISO 9001: 2000 and TQS certification were compared.

- ISO certified organizations represent higher effectiveness as compared to TQS certified organizations in terms of quality and customer satisfaction.

- ISO certified organizations are more focused towards customer satisfaction than TQS certified organizations.

- Learning and growth perspective of both kinds of organizations were found equivalent effectiveness, which implies that innovation was considered by both (i.e. ISO certified and TQS certified) organizations.

Up to 88.2% of the professionals agreed that ISO certified organizations represent higher effectiveness as compared to TQS certified organizations in terms of software documentation, processes of the organization and sales services.

The PhD thesis of Griesemer (1999) is based on a field survey of the software development organizations in the USA. The method of data collection used by the researcher is a mailed survey, accompanied by telephone interviews where applicable. The study compared the software development organizations who have received ISO 9001 certification during 1992-1995 and in 1996 to the non-certified software organizations. The major findings of this study are summarized next:

- Differences were found in term of software process improvement. These differences include monitoring of process, process improvement goals and risk management Griesemer (1999).
- Differences in software development and software productivity have been identified by ISO 9001 certified software organizations. ISO 9001 software organizations have become more efficient in terms of job satisfaction and budget constraints and the software process productivity. “These findings indicate obtaining ISO 9001 certification is significant within a software development organization” Griesemer (1999).
- The customer satisfaction levels were improved by the US organizations having ISO 9001 certification.

A more recent PhD thesis by Fuller (2006) is based on the comparison of the US market (North American organizations that attained ISO 9001 certification during 1990 to 1999) and Japanese market (Japanese organizations that attained ISO 9001 certification during July 1994 to October 2000) with respect to the certification of the software engineering processes. The survey methodology is used by the researcher for conducting the study. For assessing the consequences of certification of the software engineering processes, data was collected from the stock exchange listings. The organizations of both markets were also compared on the basis of the CMM model. The major findings of the study are:

- Positive response for the US organizations and negative response for the Japanese organizations having ISO 9001 certification: cost of production for US organizations was

reduced and less quality improvement and marketing advantages have been indicated by Japanese firms.

- US firms rely on benefits with respect to the marketing and quality aspects of the products, while Japanese firms rely on legal contract related benefits Fuller (2006).
- The Canadian organizations were selected from the database. The major findings reveal that investing in the certification process will increase competitiveness in the market, improve the software process activities and reduce the cost of production. Therefore, these benefits must be considered within the software development organization Fuller (2006).

It can be interpreted from the above described evidences from the industrial surveys and papers that the adoption of ISO 9001 in software organizations is beneficial with respect to the enhancement of the quality specifications, customer satisfaction, motivation and job satisfaction and standardization of software development processes and improvement.

2.3 Evolution of auditing practices

The term *audit* came into general use after World War II, when the military issued a standard and specifications for developing complex products and systems. The term *auditing* was introduced to refer to a set of inspection activities conducted in large manufacturing companies (in the electronics industry, for example) and in high risk manufacturing sectors (in the nuclear, food, and pharmaceutical industries, for example) Chorafas (2008).

In 1970, the United States Government Accountability Office (GAO) indicated that auditing in federal agencies needed to be conducted in a more comprehensive manner. Moreover, the GAO was advocating entirely different auditing practices, addressing companies and organizations from various perspectives. For example, according to the GAO, auditing practices should not be limited to the review or examination of financial statements by accountants, and should include investigation of:

- ✓ The organization's level of compliance with laws and regulations;
- ✓ The efficiency of all the activities conducted within the organization;
- ✓ The effectiveness of the activities in achieving their objectives.

In 1980, special standards and new laws were created to ensure more frequent and better auditing practices to cover all organizational sectors and their related activities. Later, in 1990, the amount of federal government auditing increased, and new laws and regulations were mandated to focus on additional issues, including performance, management, compliance, and the effectiveness of the auditing activities themselves. As a result, federal government audit practices have become a key element in meeting the government's responsibilities and providing a degree of confidence that is understood by all parties.

Recently, auditors have begun to scrutinize business process controls to determine the level of adherence of organizations to industrial standards and federal laws. The premise is that, although a financial statement audit is important, it provides incomplete information, since software systems can also affect the organization's business processes. IT auditing should therefore be initiated to covers all aspects of IT practices, with a view to examining the organization in terms of its adherence to industrial standards and federal laws Chambers and Rand (2010).

IT auditing should not be confused with financial auditing, even though there may be some overlaps in the work of the two groups of auditors. IT auditing provides an examination of computers, databases, and software systems. It is a professional discipline involving several different techniques for independently reviewing IT processes (e.g. software processes), as well as IT applications (e.g. financial records databases).

2.4 Auditing for certification

Auditing is a systematic and independent examination for determining whether or not an organization's activities (i.e. business processes) are in conformity with the requirements of

a specific standard or set of rules, and whether or not those activities have been effectively implemented and are suitable for achieving their predefined objectives Paul, Curtiss *et al.* (2009). The activities may be carried out at various levels, such as: organization, system, process, project, or product. This paper focuses on ISO 9001 auditing activities conducted at the software process level. On the other hand, certification is a written assurance that a product or process conforms to specified criteria.

The above generic definition of auditing embodies several important points:

- It is an *independent process*, in that auditors collect and evaluate evidence, and the results, based on their findings, are unaffected by the client or the organization. The role of an auditor in this case is similar to that of a judge who collects and evaluates evidence based on the law.
- It is conducted to *evaluate* whether or not an organization's internal controls are performing as they are supposed to. The primary objective of the auditing process is to establish whether or not these internal controls have been implemented effectively.
- Auditors examine, analyze, and judge the internal controls to determine whether or not they are suitable for achieving their predefined objectives.
- It is a systematic examination, which means that it is carried out in a methodical manner. It is also a planned activity performed systemically on the organization's activities.

Auditors begin by extracting from ISO 9001 the specific information that will be considered later as the *basis* for the auditing process. This basis corresponds to the set of recognized best practices that the organization should implement in order to comply with ISO 9001 requirements. The *evidence* is a set of facts that objectively confirms how those best practices have been implemented and to what extent they have achieved their objective. The results of comparing the audit basis to the evidence are called *observations*. Those observations should be subjected to several analysis cycles before they are summarized into what are called *findings* – See figure 2.4.

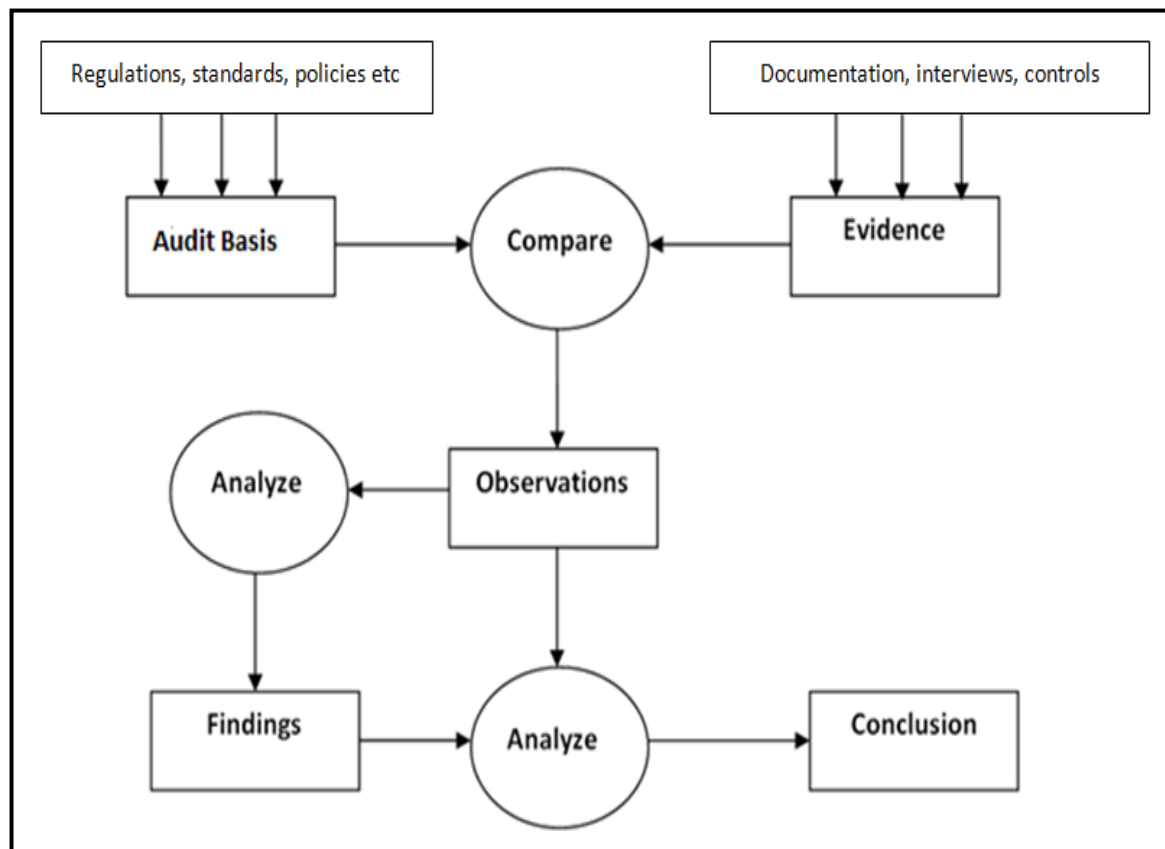


Figure 2.4 Generic auditing model
Paul, Curtiss *et al.* (2009)

The difference between an observation and a finding is that an observation consists of raw data which need exhaustive examination and analysis before they are useful to stakeholders, governments, or senior management. A finding is the result of investigating observations: it is the most important piece of information, and constitutes the final result of the auditing process. Finally, an audit conclusion is prepared and reported to all interested parties.

For many software organizations investing in quality improvement by implementing the requirement and guidelines of a quality model or standard can bring many advantages such as increasing the customer satisfaction and enhancing the value of the product among competitors. Fuller (2006) mentioned that for software organizations investment in quality

improvements based on recognized quality standards can help the organization to organize, control, and improve the development and maintenance of a software system.

For software organization to formally demonstrate that it has implemented the requirements of a specific quality standard such as ISO 9001, it has to show a certificate. Certification requires an audit of the processes by an independent third party who declares that the process meets a defined standard. Many studies have argued that improving the software processes to meet the certification level can reduce the development cost Fuller (2006), Griesemer (1999).

2.5 ISO 9001

ISO 9001 is an international standard that specifies the Quality Management Standard (QMS) requirements for generic product categories. The standard was first established in 1984 and has evolved in 1994, 2001 and 2008- See figure 2.5. A major shift has been made in 2001 and 2008 and the standard has become a process-oriented rather than product-oriented: in other words, it involves establishing processes to ensure that quality is built into the product. On the other hand, software improvement models such SEI-CMM (Software Engineering Institute - Capability Maturity Model) and software process models have evolved with a goal to increase the customer satisfaction and the final quality of the product.

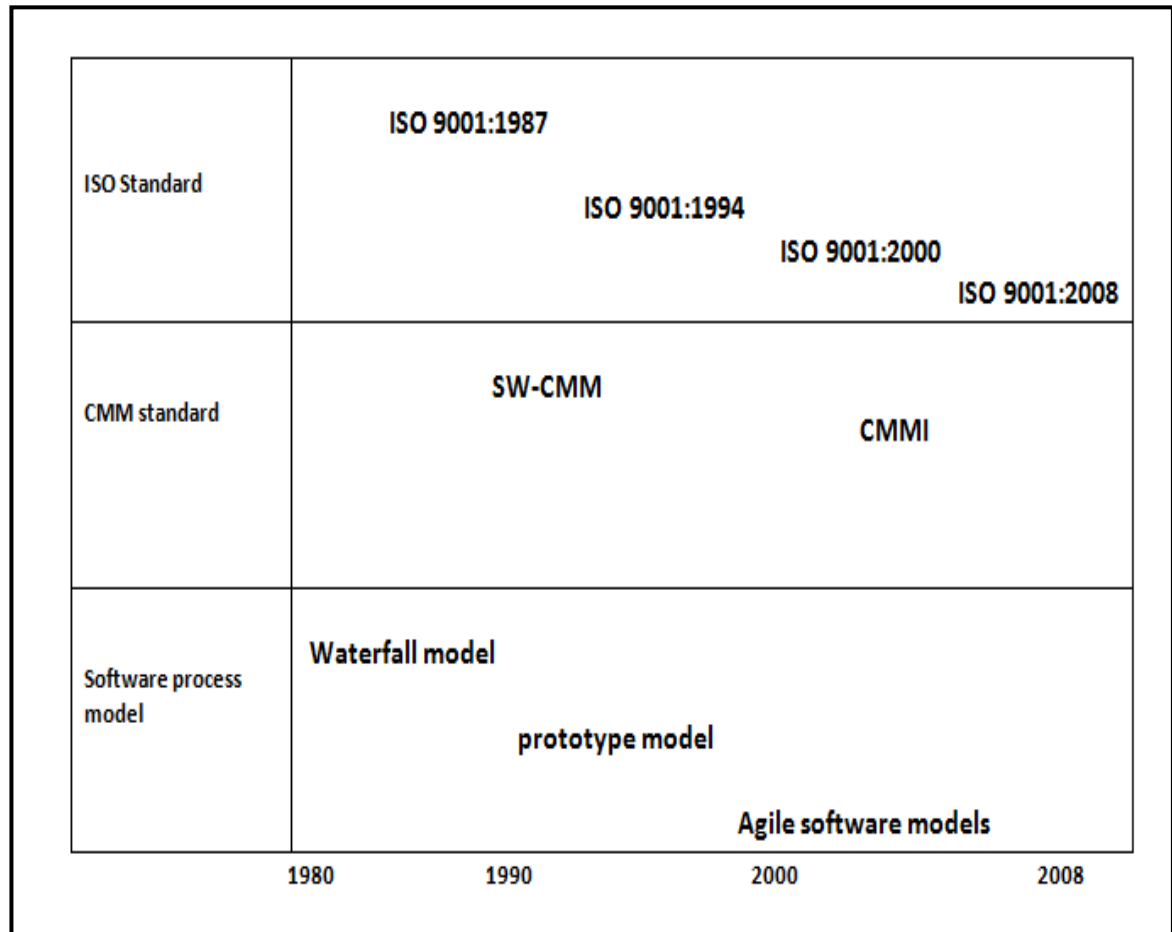


Figure 2.5 ISO 9001 Time line

ISO 9001:2008 requires organizations to explicitly commit to establishing a QMS and setting its basis. Organizations are asked to analyze their activities and processes, identify the interactions between the different procedures, and decide on the improvements to be made. Also, ISO 9001:2008 outlines the importance of documentation management procedures. This standard refers to a set of requirements to create and control the products and services of an organization to enhance customer satisfaction. ISO 9001:2008 consists of four major requirements:

- **Management responsibilities:** Provide requirements for top management to develop a quality management system and make a commitment to the organization's stakeholders.

- Resource management: Provide requirements to support all the resources needed to efficiently operate the organization and enhance customer satisfaction. This includes human resources, infrastructure, and work environment.
- Product realization: Provide requirements that support product development activities. This includes product planning, product design and development, control of monitoring and measuring equipment, and product purchasing.
- Measurement analysis and improvement: Provide a requirement to gather, analyze, and improve the product-related activities. This requires an analysis of key data gathered during audit and customer feedback to improve the final product.

ISO considers software development and maintenance important and a technical report guide (ISO 90003) was developed in 1991 and improved later in 2003 to help software organizations in their effort to obtain ISO certification. In ISO 90003, the engineering process is made up of several phases with defined inputs and outputs of a primary goal to ensure that the most efficient engineering and business practices can be implemented by the software organization.

Some software organizations have successfully obtained ISO 9001 certification through the guidance of ISO 90003 Fuller (2006), while it has been reported that software organizations were only a small part of the overall scope of the business that were certified Fuller (2006).

2.6 Summary

ISO 9001 provides common requirements for the industries and organizations worldwide that consider the best practices of ISO standardization. The objective of ISO 9001 is to improve and standardize the organization processes using common criteria (i.e. requirements) regardless of the organization geographical location, business policy and size. This chapter analyzed several research studies and surveys that report on the implementation of ISO 9001 in software organizations. The following comments can summarize the findings of this chapter:

- ISO 9001 is not designed to standardize the practices of specific type of organizations. The standard can be implemented by any organization including software organizations that consider to target the ISO 9001 certification or to improve their quality activities based on the recognized practices of ISO 9001.
- Since the software industry is important to ISO, ISO has developed a guidance document (ISO 90003) to serve as a guide for the application of the ISO 9001 standard to the development, supply, and maintenance of software.
- Some software organizations have successfully achieved the ISO 9001 certification. Increased market competitiveness, improved software process activities and reduced cost of production were among the reported advantages of achieving the ISO 9001 certification in software organizations.
- The certification process of ISO 9001 requires auditors (i.e. third party bodies) to provide independent confirmation that organizations meet the requirements of ISO 9001. Audits are essential to verify the existence of evidences showing the conformance of organization processes to specific ISO requirements and to assess how processes have been implemented.

Finally, it has been noted that both ISO 9001 and ISO 9003 have not specified any process model for the organization. Therefore, software organizations can implement the process model (i.e. traditional process model or agile process model) that fits their business needs, beside of being interested in ISO 9001 certification. This research thesis focus is to help software organizations that implement the agile software processes to achieve ISO 9001 certification.

CHAPTER 3

RESEARCH GOAL, OBJECTIVES, AND METHODOLOGY

3.1 Introduction

A research methodology is one of the keys to the success of a research project. It helps ensure the validity of research activities and results. Ellis and Levy (2008) mention that “most research problems are too large or too complex to be solved without subdividing them. The strategy therefore, is to divide and conquer. Almost every problem can be broken down into smaller units from a research standpoint, these units are easier to address and resolve”.

This chapter describes the research project definition including: the research motivation, the research goal, the research objectives, the key inputs to this research work, the users of the research results and the research methodology.

3.2 Research motivation

The motivation of this research is to help the software organizations that follow agile software process in their efforts for meeting the ISO 9001 certification requirements. This research project aims also to help the IS-auditors to extract auditing evidences that demonstrate the conformance to ISO 9001 requirements of software organizations with agile software processes.

3.3 Research goal

Obtaining a major certification, such as ISO 9001, can provide marketing and quality advantages to software organizations. However, software organizations which have adopted

agile software processes such as XP usually provide less documentation during the development of a software product making it challenging to demonstrate conformity with ISO 9001. The goal of this research project is to improve the agile-XP process in supporting the auditing requirements of ISO 9001. As well, the goal of the research is to help the agile software organizations in their efforts to become ISO 9001 certified.

3.4 Research objectives

To achieve this research goal, the following specific research objectives must be achieved:

- Identify gaps between agile-XP and ISO 9001, by highlighting the main strengths and weaknesses of agile-XP in handling the ISO 9001 requirements.
- Propose sub processes to enhance the early planning activities of agile-XP according to ISO 9001 requirements.
- Design an auditing model that covers the measurement and traceability requirements of ISO 9001. The auditing model should provide the IS auditors with auditing evidences that the software projects developed with an agile-XP process have fulfilled the requirements of ISO 9001.
- Verify the applicability of the auditing model on agile traceability and agile measurement approaches.

3.5 Research inputs

The next list highlights the main input models and frameworks that will be used to achieve the research objectives:

- ISO 9001:2008 (Quality management systems- Requirements);
- ISO 90003:2003 (Software engineering- Guidelines for the application of ISO 9001 to computer software);
- Vincenti's engineering design Vincenti (1990);

- SWEBOK Guide (2004): The generally accepted body of knowledge on software engineering - the SWEBOK Guide - (ISO-TR-19759 2004);
- An Evaluation Theory Perspective of the Architecture Tradeoff Analysis Method – ATAM Lopez (2000);
- ISO 12207:2008 Systems and software engineering – Software life cycle processes;
- CMMI for Development, Version 1.3;
- Standards, Guidelines and Procedures for information system auditing, (ISACA, 2010).

3.6 Research users

The users of research in this research work are people who are working on agile process improvement and software process certification and who can benefit from the research outcomes:

- Project managers: to address the main weaknesses and strengths of agile-XP processes for developing a software system that conforms to ISO 9001 requirements.
- Software developers who need to utilize the agile-XP auditing model for developing the software system in conformance with ISO 9001.
- IS auditors who need to extract the auditing evidences from agile process to assure that the software organization with an XP process conforms to ISO 9001.

3.7 Overview of the research methodology

This section presents an overview of the research methodology designed to pursue the research objective.

Phase 1: Agile processes in the literature review

Phase 1 of the research methodology will consist of surveying the differences and similarities among different agile processes, and to study their level of adoption in software

organizations from both academic based studies and industrial based studies. This phase also consists of surveying different process improvement models. See chapter 1.

Phase 2: ISO 9001 and auditing principles.

Phase 2 of the research methodology will consist of understanding the impact of ISO 9001 on software organizations, as well as of revealing evidences from the literature of the ISO 9001 adoption level in software organizations. This phase also includes the understanding of the main principles of ISO 9001 auditing process and the review of several software process improvement models. See chapter 2.

Phase 3: Investigation of the capability of agile-XP to achieve the requirements of ISO 9001 software process certification.

Phase 3 of the research methodology will consist of extracting the ISO 9001 requirements that are related to software process life cycle, and to map them to the agile-XP to evaluate the inability of agile-XP for handling the ISO 9001 requirements. See chapter 4.

Phase 4: Modifying the early phase of agile-XP (i.e. release planning phase) using CMMI-DEV.

Phase 5 of the research methodology will consist mainly in designing an extension to the agile- XP user stories. The extension will be designed based on CMMI-DEV model. More precisely, the extension will be based on “Requirement Development”, “Requirement Management” and “Risk Management” CMMI-DEV process areas. See chapter 5.

Phase 5: Design of an auditing model for ISO 9001 traceability requirements.

Phase 5 of the research methodology will investigate of the fundamentals of evaluation theory for the design process of auditing criteria and yardsticks for ISO 9001 traceability requirements. This phase will also use the principles and guidelines of Vincenfi’s engineering design, SWEBOK and CMMI-DEV models. See chapter 6.

Phase 6: Design of an agile-XP auditing model for ISO 9001 measurement requirements.

Phase 6 of the research methodology will consist mainly of extending the auditing model as to cover the ISO 9001 measurement requirements. This phase also will consist of designing an additional auditing criteria and yardsticks based on guidelines of Vincenfi's engineering design, SWEBOK and CMMI-DEV models. See chapter 7.

Phase 7: Case studies from the literature.

Phase 7 of the research methodology will consist mainly of selected case studies of different traceability agile systems and measurement agile systems. The objective of this phase is to evaluate the applicability of the design model for auditing an agile traceability system and agile measurement systems to assure their conformance with ISO 9001 traceability requirements. See chapter 8.

3.8 Detailed research methodology

To achieve the objective of this research, several software improvement models and best practices from the domain of software engineering and from other disciplines outside software engineering have been investigated. The selected models and frameworks will be used to provide a methodological process for each research phase. This section provides a detailed description of the main phases of the research methodology. This research methodology consists of seven phases as seen in figure 3.1.

Phase 1: Agile processes in the literature review

The objective of this phase is to develop an in-depth understanding of different agile software practices and to select an agile process that is suitable to the research objective. This phase of the methodology consists of following steps:

Phase 1.1: Analysis of agile software processes

This step has identified and analyzes five different agile software processes: Extreme programming (XP), scrum, feature driven development (FDD), adaptive software development (ASD), and Crystal methodology.

Phase 1.2: Comparison of agile software processes

A comparison between agile software processes has been conducted to facilitate the understanding of their principles and practices. This step involved the development of a general software project requirements framework and identification of the similarities and differences between agile software processes.

Phase 1.3: Selection of agile software process

A selection of agile processes has been performed based on the outcome of the previous step and the adoption level of the candidate agile software processes found in the literature, from both academic and industrial surveys.

Phase 2: ISO 9001 and auditing principles

The objective of this phase was to develop an understanding of ISO 9001 impact on software organizations and to identify the level of adoption of ISO 9001 in software organizations. This phase of the methodology consists of the following steps:

Phase 2.1: Survey Evidences on ISO 9001 and Software organizations

This step summarized several papers and industrial surveys to understand the adoption of ISO 9001 in software organizations. This step will provide insights on the implications and the adoption level of ISO 9001 in software organizations.

Phase 2.2: Auditing for ISO 9001 Certification

This step provided details of the auditing processes, practices and activities that are usually preformed by IS-auditors to assess the software organization in conforming to ISO 9001 standard.

Phase 3: Analysis of agile-XP from ISO 9001 perspective

The gap between the practices of agile-XP and ISO 9001 requirements has been identified in this phase. This phase consists of the following steps:

Phase 3.1: Extraction of requirements from ISO 9001 and the guidelines from ISO 90003 that are related to software process life cycle.

This step has used ISO 12207 as a filter interface between ISO 9001 & ISO 90003, and XP.

Phase 3.2: Grouping of the ISO 9001 requirements based on their related support process found in ISO 12207.

Phase 3.3: Analysis and report of the findings

This step identifies the activities of agile-XP that can support or handle the extracted requirements from ISO 9001 and ISO 90003. It is important also to report on agile-XP activities limitations and weaknesses in handling the ISO 9001 requirements.

Phase 4: Modification of the early phase of agile-XP (i.e. release planning phase) using CMMI-DEV.

This phase has focused on proposing different sub processes aligned with the agile-XP release planning phase. These sup-processes will contribute together to extend the agile-XP user stories to provide important information for the ISO 9001 IS-auditors. This phase of the methodology consists of the following steps:

Phase 4.1: Identification of ISO 9001 needed information at early process phases.

This step will focus on the analysis of ISO 9001 customer requirements and ISO 9001 planning requirements. The findings of the phase 3 are also considered as an essential input for this step.

Phase 4.2: Proposal of XP-Agile Sub processes based on CMMI-DEV

Four different agile-XP sub processes have been proposed with the guidance of CMMI-DEV. These sub processes are: 1) identification of the source of the user story; 2) categorization of the non-functional requirements; 3) identification of the user story relationships; and 4) prioritization of user stories.

Phase 4.3: Integration in XP-agile user stories

This step focuses on the integration of the information collected based on the proposed sub processes into XP-agile user stories.

Phase 5: Design of an agile- XP auditing model for ISO 9001 traceability requirements

The Evaluation theory along with the best practices found on CMMI-DEV, SWEBOK and Vincenti's engineering design will be investigated to develop an auditing model for agile XP. The auditing model will focus on the ISO 9001 traceability requirements. This phase of the methodology consists of the following steps:

Phase 5.1: ISO 9001 Traceability requirements

An analysis of traceability requirements has been conducted in this step. This will allow for better understanding of the main obligations that are required by the software organization to implement the requirements at this stage.

Phase 5.2: Modeling with the Evaluation theory

The principles of the evaluation theory have been applied at this step to formulate the essential elements of the agile-XP auditing model. The auditing model will be composed of two major auditing criteria: engineering criteria and management criteria. Furthermore, each will be refined into an auditing criterion and an auditing yardstick.

Phase 6: Design of an agile-XP auditing model for ISO 9001 measurement requirements.

An extension of phase 5 has been conducted in this phase. This phase of the methodology consists of the following steps:

Phase 6.1: ISO 9001 Measurement requirements

An analysis of ISO 9001 measurement requirements has been conducted in this step. Several relations between the ISO 9001 measurement requirements and other ISO requirements has been defined in this step to enable for a better understanding of ISO 9001 measurement requirements.

Phase 6.2: Modeling with the Vincenti engineering design

The principles of the Vincenti engineering design has been applied in this step to extend the elements of the agile-XP auditing model. The auditing model has been composed of three major auditing criteria: auditing criteria for measurement plan, auditing criteria for measurement development and auditing criteria for measurement management. Furthermore, each has been refined into an auditing criterion and /or an auditing yardstick.

Phase 7: Case studies different agile traceability and measurement systems.

This phase focused on the selection and development of case studies that are applicable to verify the validity of the auditing model. This phase of the methodology consists of the following steps:

Phase 7.1: Context and Scope

This step focused on the identification of the case studies that are suitable to verify the validity of the auditing model. The case studies has been selected from the literature and composed of two sets, first set is five agile traceability systems and the second set is four agile measurement systems. The selected case studies are not limited to XP.

Phase 7.2: Collection and analysis of auditing evidences

This step focused on collecting the audit evidences from the selected case studies. The audit evidences has been collected based on the auditing criteria and yardstick. This step also focused on the analysis of the collected evidences. The analysis has been accomplished with the support of standards, guidelines and procedures for information system auditing (ISACA, 2010).

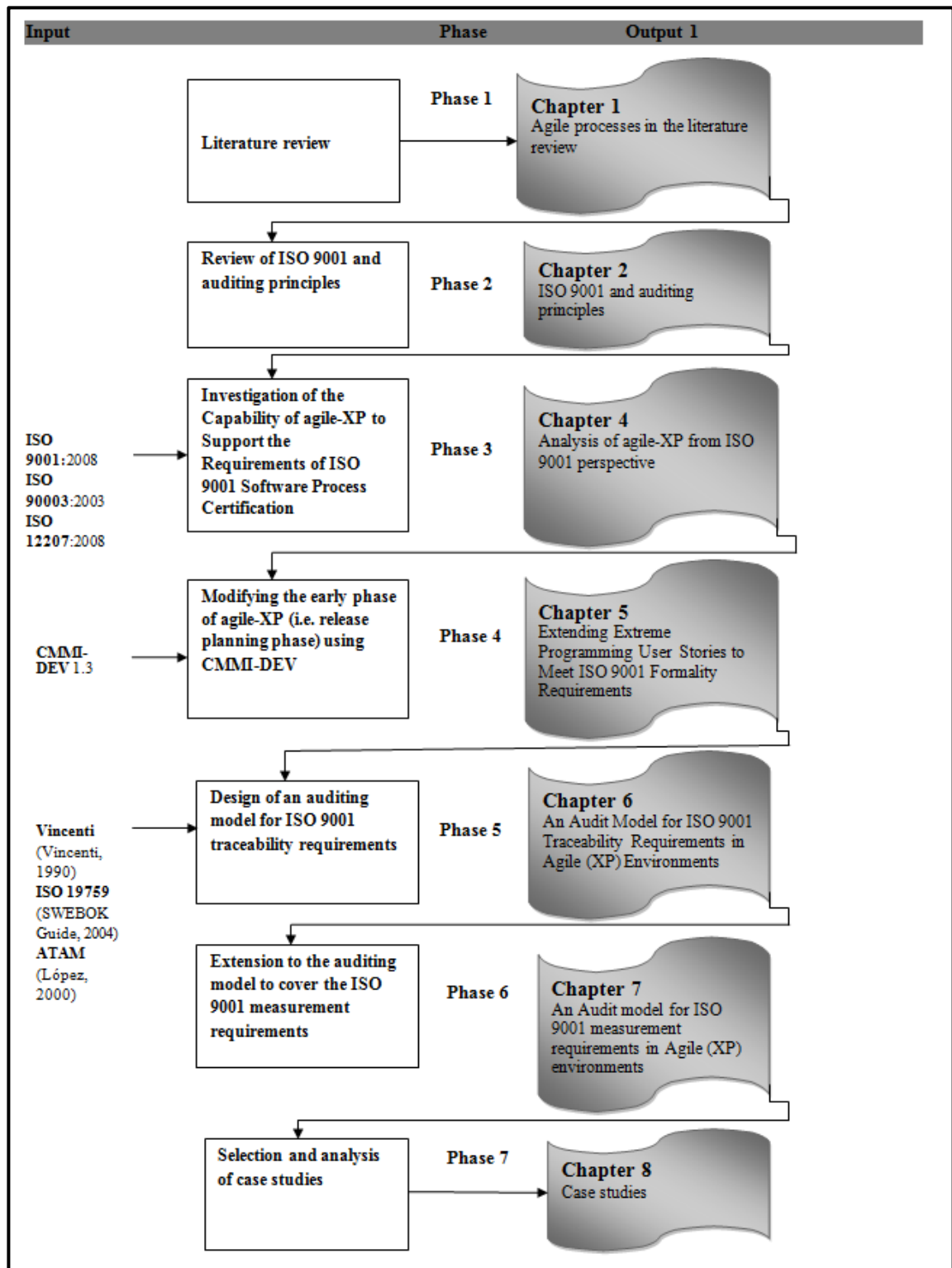


Figure 3.1 Detailed research methodology

CHAPTER 4

ANALYSIS OF AGILE-XP FROM ISO 9001 PERSPECTIVE

4.1 Introduction

This chapter presents an analysis of agile-XP from the ISO 9001 and ISO 90003 perspectives. The focus is to extract the requirements related to the ISO product realization process and to determine the strengths and weaknesses of agile-XP in handling those requirements.

The factors that affect the software certification process have been defined by Yahaya, Deraman *et al.* (2009) as: the personnel factor, the process factor, and the product factor. These three factors are also known as the certification triangle:

- The personnel factor focuses on skills, experience, knowledge, team commitment, user involvement, and management responsibility.
- The process factor includes three basic activities, which are those related to development, management, and support.
- The product factor includes tools, devices, and the software system.

Of the three factors that affect the certification process (process, product, and personnel), this chapter focuses on the process factor. It presents an analysis of the agile-XP approach from the ISO 9001 and ISO 90003 perspectives, and reports on the strengths and weaknesses of agile-XP for supporting the requirements of ISO 9001 and the guidelines of ISO 90003. The motivation of this chapter is to identify enhancement opportunities to agile-XP to support the extracted ISO 9001 requirements and ISO 9003 guidelines.

This chapter is organized as follows:

Section 4.2 presents the scope and the design process of the chapter analysis.

Section 4.3 presents an analysis for the ISO 9001 requirements that are related to software process life cycle. The section also presents the mapping results for the extracted ISO 9001 requirements to agile-XP.

Finally the summary of the main findings is presented in section 4.4

4.2 Analysis scope and design

This section explains the proposed design process for the analysis to achieve the chapter objectives.

4.2.1 Analysis scope

This chapter focuses on extracting the requirements from clause 7 of ISO 9001: the product realization process. Figure 4.1 shows the sub-clauses that need to be mapped to agile-XP. These items are ticked, while the unselected clauses are tagged with an “x”. The focus here is to extract the requirements related to a software process life cycle (i.e. Requirements gathering and evaluation, design and development, design review, and design verification and validation).

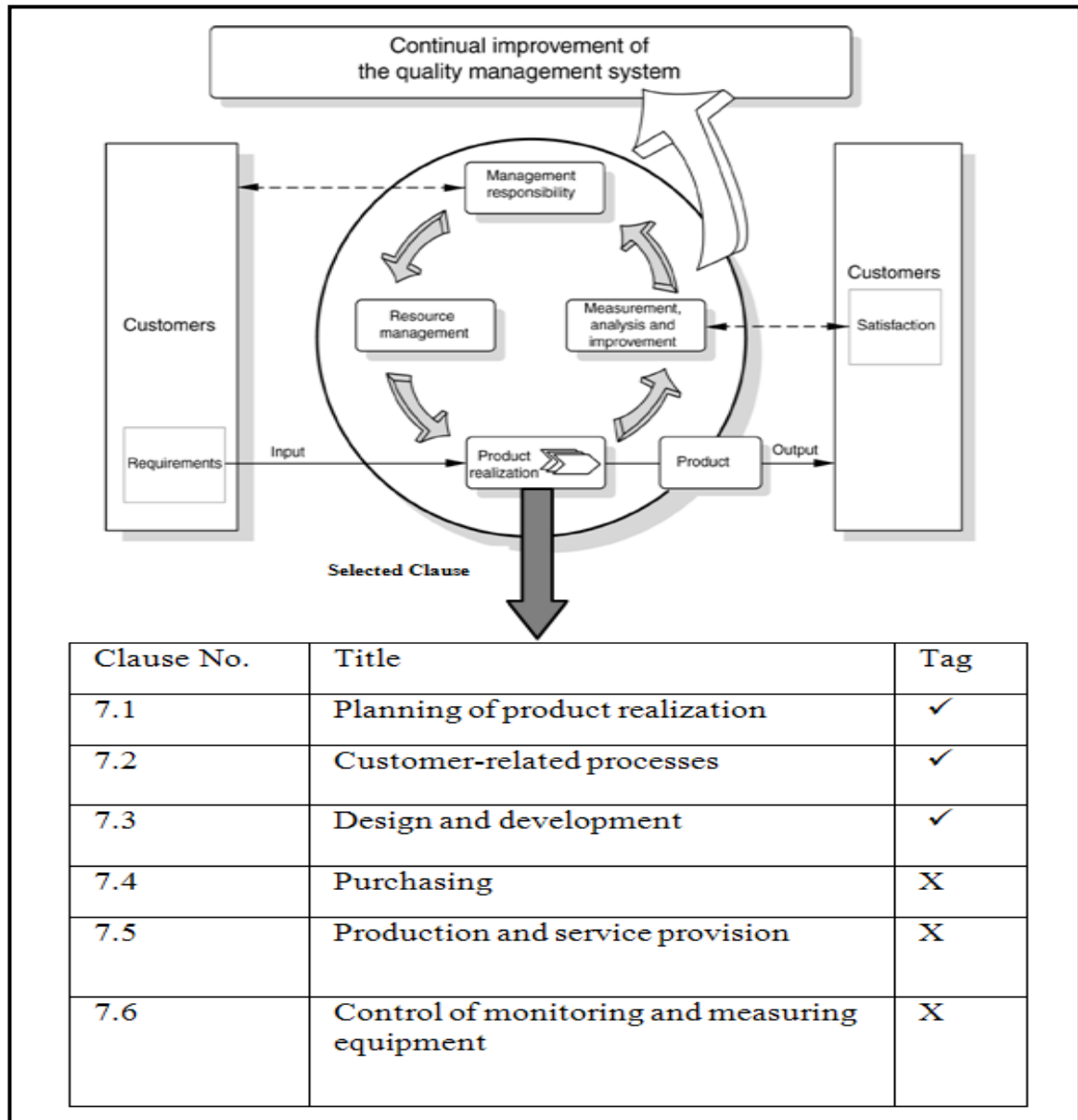


Figure 4.1 ISO 9001, clause 7

4.2.2 Design process for the analysis

The definition of ISO 12207 for the software life cycle model is used to determine the ISO 9001 requirements and the ISO 90003 guidelines that need to be addressed. ISO 12207 defines the software life cycle model as “a framework containing the processes, activities, and tasks involved in the development and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use” (ISO

12207). Based on this definition, the ISO 12207 process model is used to facilitate the extraction of ISO 9001 and ISO 90003-related requirements. Moreover, ISO 90003 states that an organization may choose to use the processes or sub processes from ISO 12207 to support the requirements of software process certification. The ISO 9001 and ISO 90003 certification requirements are next extracted based on their related support processes from ISO 12207, and then mapped to agile-XP. The items highlighted in Figure 4.2 are the processes that need to be mapped for this study.

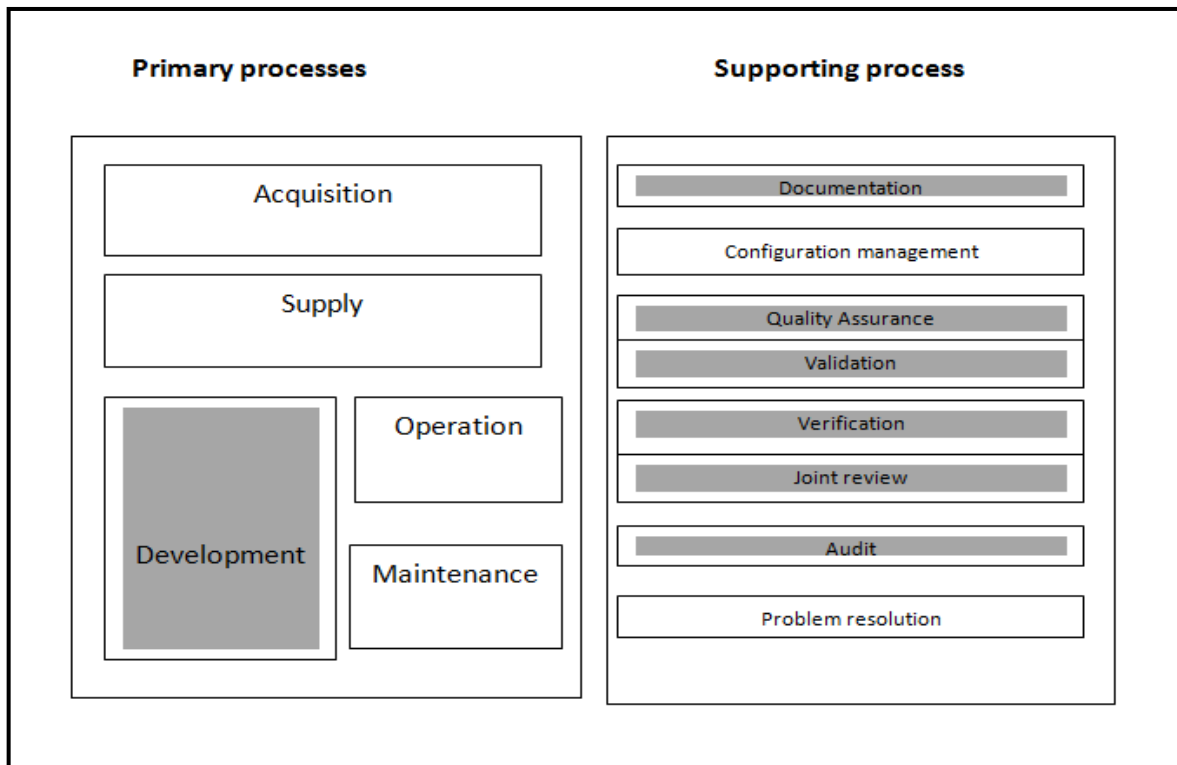


Figure 4.2 ISO 12207 focused processes

ISO 90003 does not recommend any software process model, but rather suggests certain phases that should be considered, such as the determination of software-related requirements, design and development, validation and verification, and customer feedback (joint review in ISO 12207).

This section presents the process used for mapping the selected ISO standards to agile-XP – see Figure 4.3. The steps of this mapping process are described next.

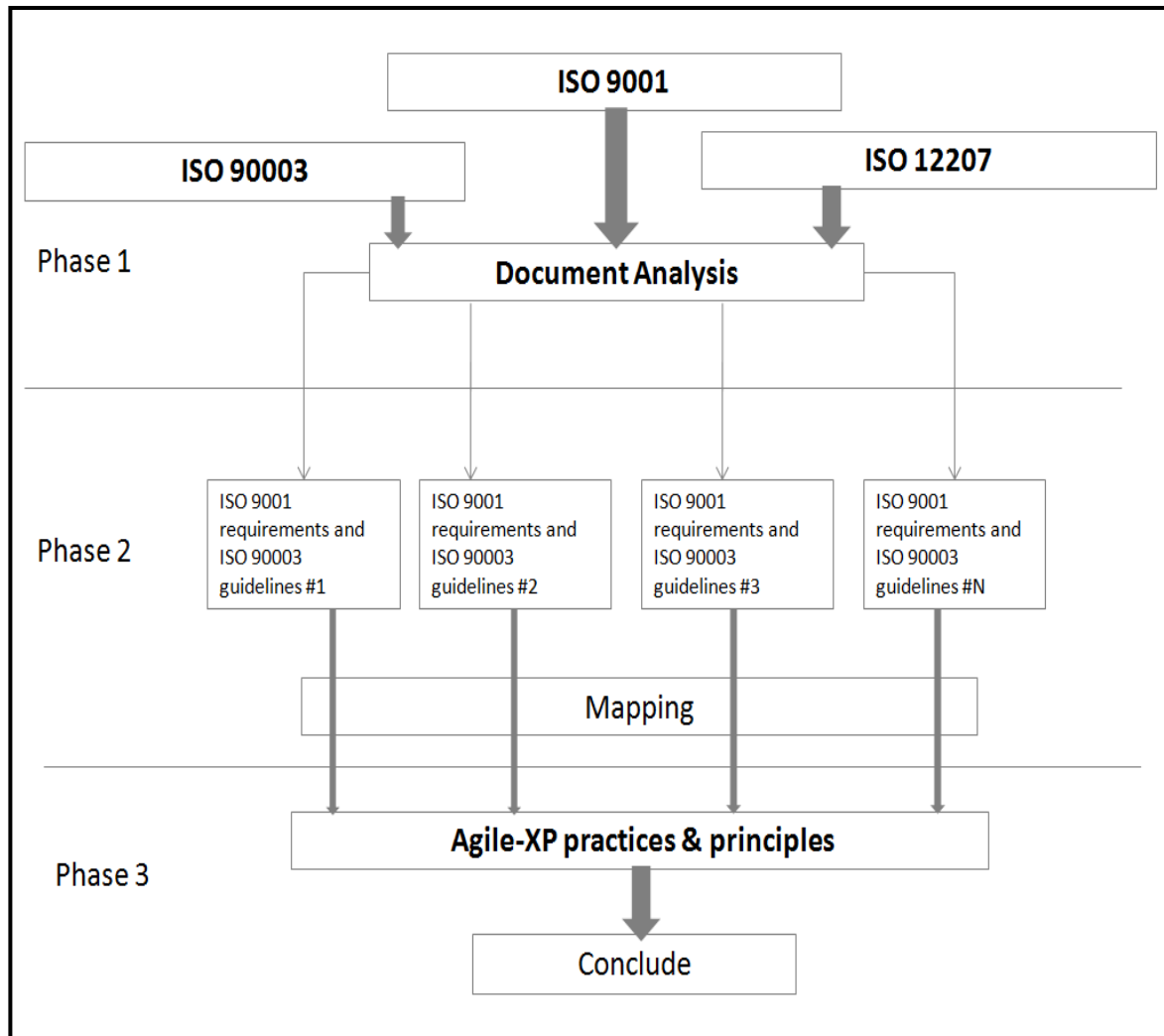


Figure 4. 3 Mapping process & phases

- **Phase 1 – Document analysis :**

ISO 9001 and ISO 90003 specify a set of requirements to provide confidence to customers and other stakeholders that specified quality requirements will be met. To stay within the scope of this study, the ISO 12207 model is used to extract all the activities from ISO 9001 and ISO 90003 that are related to software life cycle (i.e.

requirements phase, design and implementation phase, verification and validation phase).

- **Phase 2 – Mapping:**

The objective of this step is to identify the activities, principles, and procedures involved in agile-XP that can meet the requirements extracted from Phase 1. This mapping phase examines the capability of agile-XP to meet ISO 9001 and ISO 90003 requirements.

- **Phase 3 – Conclusion:**

During this step, a conclusion is drawn as to whether or not the extracted ISO activities can be supported by agile-XP.

4.3 Mapping results

The mapping results of the following ISO requirements are discussed below.

4.3.1 Planning of product realization

4.3.1.1 ISO requirements: planning stage

A software project plan should “describe the technical and management approach to be followed for a project.” IEEE Standard 610.12-1990. According to this standard, the plan typically describes the work to be done, the resources required, the methods to be used, the procedures to be followed, the schedules to be met, and the way the project will be organized.

ISO 9001 requires the details of the design activities and when they will be carried out. Several quality-related design activities have to be clarified at this point: i.e. reviews, as well as validation and verification activities, have to be mapped to their related development stages. According to Meridji (2010) in manufacturing and engineering, the term design does not mean the same thing as the software design phase in software engineering. Meridji

(2010) has compared the design principles described in Vincenti (1990) and the design principles described in SWEBOK Guide. The analysis concludes that design in engineering according to Vincenti is not limited to design as described in the SWEBOK Guide, but goes beyond, in that it is composed of the whole of the software engineering life cycle.

ISO 90003 gives more specific guidelines to be applied to software development planning. Those related to software process can be classified in three categories.

- Development activities: The development plan should clearly identify the software processes or software engineering methodologies needed for every software project. At this stage, the developer needs to identify the activities involved in the development phases. The development activities will include:
 - ✓ Identification of the required inputs and outputs for each phase.
 - ✓ Identification of the practices and procedures used to verify that the software development phases are being followed and that the product has been tested in various phases.
- Management activities: The development plan should include various activities to define how the project is to be managed. The management activities will include:
 - ✓ Identification of the deliverables schedule. Once the schedule has been drawn up, the time, resources, and budget for each deliverable can be determined.
 - ✓ Identification of the organizational and technical interfaces. This includes identification of the development team structure and of the role and responsibilities of each developer. This is important when the software product is developed in a collective ownership manner, which is the way the agile-XP approaches the software development process.
 - ✓ Identification of any regulatory requirements that may affect the software process or product.
- Methods and tools activities: The development plan should identify methods to ensure that all the development activities are carried out in a methodological manner. This will include: Identification of tools and techniques for development. Developers may use tools or methodologies in requirement analysis, design, coding and testing. These tools should be clearly specified in the plan.

The above-mentioned requirements can be supported by the ISO 12207 processes in section 5.3.1 on process implementation. Sections 5.3.1.1 and 5.3.1.4 support and guide the developers through several activities that can support the ISO 9001 and ISO 90003 requirements. For example, section 5.3.1.1 requires developers to select and define the software life cycle model that is appropriate for the scope of the project.

4.3.1.2 Mapping to agile-XP

Planning is an integrated part of agile-XP processes Beck (1999). In agile projects, such as XP, planning and re-planning are regular activities before and during each iteration. Planning in agile-XP can be classified in two levels.

Level 1 (Planning game) – before the release: The goal of the planning game is to maximize the value of a software product. It is a vital activity performed before each software release to decide what is to be included in each release. As indicated in Frank and Karam (2006), three phases are included in the agile-XP planning game:

- **Exploration:** This phase involves several activities called “moves”. The first move involves writing the user story. The second move focuses mainly on estimating the time required for completing each story, as well as determining a story’s required acceptance criteria. During this phase, the developers discuss the tools or techniques required for developing the story. The third move is to split the story into parts, if it could not be estimated as a whole.
- **Commitment:** This phase involves sorting the stories into three categories of functional requirements: essential, important, and nice to have.
- **Steering:** The main goal of this phase is to update the plan. Basically, this phase is performed during the iterations where the developers need to update the plan regularly.

Level 2 (Iterative planning) – during the iterations. Iterative planning is aimed at scheduling the tasks that need to be performed in the next iteration. For example, the plan is checked to detect any duplicate programming task; if such duplications are found, they are

removed. Many agile-XP practices support iterative planning (e.g. daily stand-up meeting, onsite customer) - See table 4.1.

Table 4.1 ISO 9001 planning phase mapping

ISO 9001 Requirements	ISO 12207 support activity	Mapping to agile-XP
ISO Requirements in the planning phase (section 4.3.1.1) Development activities Management activities Methods and tools activities	Process implementation: ISO 12207 in section 5.3.1	-Planning game -Iterative planning

4.3.2 Requirements phase

4.3.2.1 ISO Requirements during software requirement gathering activity

As stated in the SWEBOK Guide Abran, Moore *et al.* (2004), a software requirements specification “establishes the basis for agreement between customers and contractors or suppliers on what the software product is to do, as well as what it is not expected to do.” This process is equivalent to the ISO 9001 and ISO 90003 requirement in section 7.3.2 “design and development input”: both require the developers to identify all the functional and non functional requirements that are related to the software product. Developers should take into account other requirements affecting the software, such as national standards, company standards, etc. The experience gained from other, previous designs should be taken into account as well.

The ISO 90003 guidelines document elaborates more specific activities to be applied at this stage, such as the following:

- Method for tracing the requirement changes during iterative development, and a method for recording the changes.

- Traceability matrices for tracing the requirements to the final product.
- Details (provided by the developer) about any other software interfaces or tools and algorithms needed during development.

The above requirements can be supported by the process of ISO 12207 in section 5.3.4 on software requirements analysis, specifically section 5.3.4.1, which supports and guides the developers on the types of requirements that can be identified (e.g. safety requirements, security requirements, etc.).

4.3.2.2 Mapping to agile-XP

The main technique for software requirement specification in agile-XP is the “user stories”. These user stories focus mainly on clarifying the functional requirements and non functional requirements during software development. According to XP.org¹ user stories provide a high level description of user requirements and when developers start the implementation they start up with a face to face meeting with customers to obtain detailed descriptions for the user stories. User stories usually do not provide details of specific technology, data base layout, and algorithms intended to be used during the development -Table 4.2 illustrates the mapping results between the ISO 9001 requirements during software requirement gathering activity and the agile-XP.

¹ Information based on [www. Extremeprogramming.org](http://www.Extremeprogramming.org)

4.3.2.3 ISO requirements during software requirement validation activity

The SWEBOK Guide Abran, Moore *et al.* (2004) makes the following statement: “Perhaps the most common means of requirements validation is by inspection or reviews of the requirements document(s). A group of reviewers is assigned a brief period to look for errors, mistaken assumptions, lack of clarity, and deviation from standard practice.” ISO 9001 and ISO 90003 point out the importance of requirements review/validation in section 7.3.2, by stressing that developers should review the requirements to verify that they are not ambiguous and do not conflict with one another.

More activities related to software requirement analysis are determined in ISO 90003:

- ✓ Developers should use a suitable method for evaluation of the requirements.
 - ✓ Requirements should be evaluated in the presence of customers and in a closed meeting.
- All the ambiguous and inconsistent requirements should be recorded.

The requirements above can be supported by the software requirements analysis process of ISO 12207, specifically section 5.3.4.2, which provides criteria to help the developers in reviewing and validating the requirements. This will ensure the consistency of the requirements, as well as their implementation feasibility.

4.3.2.4 Mapping to agile-XP

Agile-XP includes several tools for validating the requirements in order to resolve the ambiguity and to keep the requirements consistent. The customer acceptance test is used to keep the requirements specifications unambiguous. Prototyping practices performed in the presence of onsite customers is another tool for keeping the requirements clear and unambiguous. The presence of customers allows the development team to obtain real-time feedback. Table 4.2 illustrates the mapping results between the ISO 9001 requirements during software requirements validation activity and the agile-XP.

Table 4.2 ISO 9001 Requirement gathering and validation mapping


ISO Requirements	ISO 12207 support activity	Mapping to agile-XP
ISO Requirements in the Software Requirements phase (4.3.2.1): Identification of all the functional and non functional requirements that are related to the software product.	Software Requirement Analysis (Section 5.3.4.1)	– User story
ISO Requirements in the Software Requirements Validation phase (4.3.2.3) Developers should use a suitable method for evaluation of the requirements. Requirements should be evaluated in the presence of customers and in a closed meeting.	Software Requirement Analysis (Section 5.3.4.2)	– Customer acceptance test – Prototyping – Onsite customer

4.3. 3 Construction phase

4.3.3. 1 Design and development review

Wiegers (2002) classify different types of reviews that may be used during the software review process. These reviews have been classified from less formal to most formal based on tools and technique used in each review method. The reviews methods have been classified by Wiegers (2002) as shown in table 4.3.

Table 4.3 Classification of Review Methods

Less Formal  More Formal	Pass Around	Several copies of the working product delivered to several reviewers to collate their feedback.
	Peer Desk Check	More formal methods such as: defect checklists, analysis methods, and standard record forms.
	Walkthrough	Walkthroughs typically do not follow a defined procedure, do not specify entry or exit criteria, require no reporting, and generate no quantitative data.
	Team Review	Team Review: typically planned and structured but are less formal and less rigorous than inspections.
	Inspection	Inspection is the most formal review methods which usually follows a well defined procedure which includes an organized examination or formal evaluation exercise.

4.3.3.2 ISO Requirements during design and development review activity

ISO 9001 and ISO 90003 do not recommend any specific review method, and allow the organizations to choose a review method based on the software project requirements.

ISO 9001 and ISO 90003 include several requirements at this stage. The objectives of the ISO 9001 and ISO 90003 review phases can be summarized as follows:

- To ensure that the methods meant to be used during the planning phase were, in fact, used during the development phase.
- To ensure that the software requirements are fulfilled by the software product.

ISO 9001 and ISO 90003 require developers to identify any problem with the software developed, and to evaluate, document, and recommend solutions to current problems. It is also important at this stage to verify whether or not the software functionality can be traced back to specific requirements. The degree of formality of the selected review method should be relevant to the complexity of the software product.

According to ISO 90003, the review of design and development should be performed in accordance with a predefined plan. The plan should identify the elements, such as:

- What needs to be reviewed and what the team responsibilities are.
- The purpose of the review process and the tools and techniques used at this stage.
- Identification of the steps needed to resolve any anomalies found during the review.

The above requirements can be supported by the ISO 12207 joint review process. Specifically, section 6.6 provides criteria to help the developers review and evaluate the software product. A joint review process is carried out by the project manager and the technical developers, as well as the customer and/or supplier. The joint review process guides the developers to establish suitable review stages in accordance with a planned arrangement.

4.3.3.2 Mapping to agile-XP

Agile-XP includes several activities that support the ISO 9001 and ISO 90003 review requirements. The XP activities that support design and code reviews can be clarified as follows:

- One of the main agile-XP principles supporting the design and code review is “pair programming”, which has been defined as “two people working at one machine, with one keyboard and one mouse” (Beck, 1999). It has been reported by researchers that pair programming leads to a better design and fewer defects, and increases the level of confidence for adding and changing the system Canfora, Cimitile, *et al.* (2005). Table 4.4 illustrates the mapping results between the ISO 9001 requirements during design and development review activity and the agile-XP

Pair programming requires one programmer to develop the code, while another programmer monitors the flow and structure of the code. Once a weakness or a need for enhancement is found, it is fixed right away by the programmers. Cockburn and Williams (2001) show that, when pair programming is being used, there is a 15% reduction in the number of bugs and a decrease from 30% to 15% in the number of failed test cases.

Table 4.4 ISO 9001 Construction phase mapping

ISO Requirements	ISO 12207 support activity	Mapping to agile-XP
<p>ISO requirement at Design and Development Review.</p> <p>Identifying of any problem with the software developed, and to evaluate, document, and recommend solutions to current problems.</p>	<p>Joint review process (Section 6.6)</p>	<p>- Pair programming</p>

4.3.4 Design and development verification and validation

Validation and verification is an integral part of any software development life cycle. Verification and validation techniques are used to identify and remove defects introduced during the software development process (Dolores and Fujii, 1989). Researchers have introduced many validation and verification techniques that could be used to improve the quality and the functionality of software products.

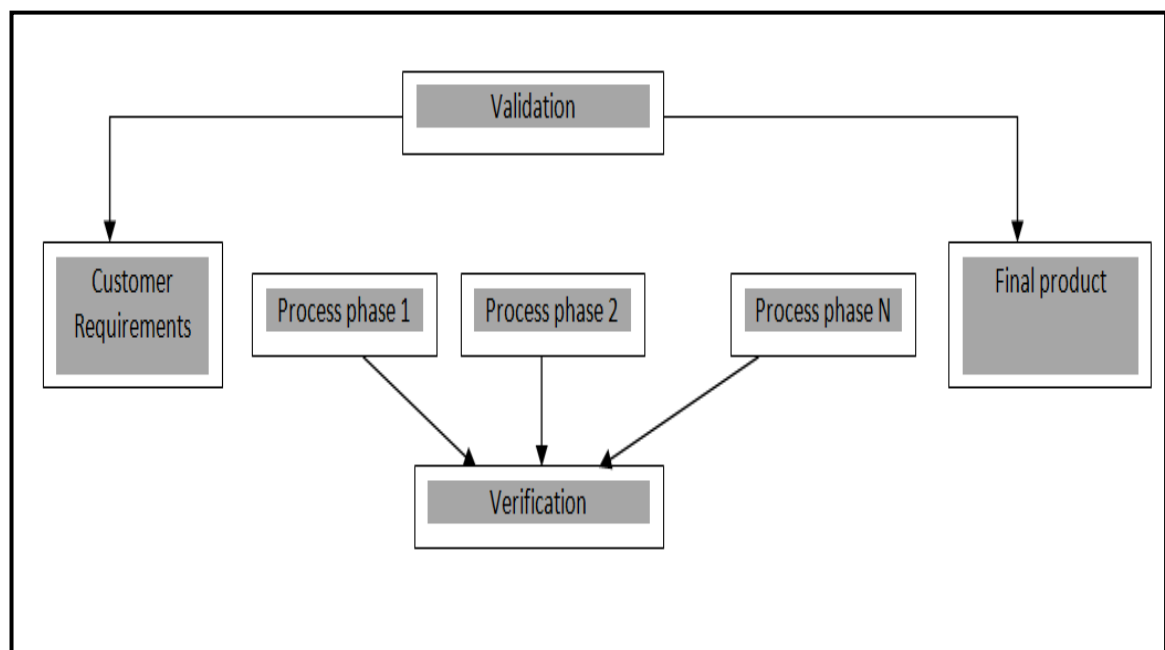


Figure 4.4 Validation and verification process

For the purpose of this analysis the definitions of validation and verification in SWEBOK are used to facilitate the extraction of ISO 9001 and 90003 requirements Abran, Moore *et al.* (2004): “verification is an attempt to ensure that the product is built correctly, in the sense that the output products of an activity meet the specifications imposed on them in previous activities”. Verification is used to help the developers to ensure building the product in the right way, where validation is defined as an attempt to ensure that the right product is built, that is, the product fulfills its specific intended purpose. Figure 4.4 illustrates the difference between validation and verification.

4.3.4.1 ISO Requirements during design and development verification and validation activities

ISO 9001 requires that both validation and verification be performed in accordance with a predefined planned arrangement. It does not elaborate on specific techniques or procedures for verifying and validating the software product: ISO 9001 is not designed specifically for software products, but rather to fit the need for manufacturing an industrial product. The requirements in ISO 9001 are defined as:

- The results of the validation and verification process should be recorded. The design and development problems found should be identified and tracked.
- The verification process should be performed at various design stages to ensure that the product meets the specified requirements. IEEE standard 610.12-1990 defines verification as an evaluation process “to determine whether the products of a given development phase satisfy the conditions imposed at the beginning of that phase.”
- The validation process should be performed at the end of the development process and before the delivery of product to the user. IEEE standard 610.12-1990 defines the validation as an evaluation process” during or at the end of the development process to determine whether the product satisfies specified requirements”.

ISO 90003 specifies that software verification should occur before validation. It elaborates more guidelines at this stage, which can be classified as follows:

- Developers should select the suitable verification method based on the size and complexity of the software project. The review methods that were discussed in section 4.4.3.1, Design and Development Review, can be used during the verification process as well. Moreover, other verification techniques, such as prototyping, simulations, and testing, are recommended in ISO 90003 as verification techniques that can be used based on the project complexity.
- The final version of a software product should be validated before running the customer acceptance test. This validation should take place in conditions that simulate the real environment in which the software product will operate.
- Validation can be performed by testing to validate the software at several levels, from individual software components to a complete software product. ISO 90003 recommends several testing techniques, such as unit testing, integration testing, qualification testing, acceptance testing, and regression testing.
- Any problems and anomalies found during the validation process should be recorded, and the appropriate steps taken to resolve them.

The above requirements can be supported by the ISO 12207 process as described in section 6.4, Verification process, and more specifically in section 6.4.2, which guides developers on several verification activities that can be used during the software life cycle. This includes design verification, code verification, and integration verification. Also, section 6.5, Validation process supports the ISO 90003 validation guidelines through systematic steps to ensure that these validation guidelines are reflected during the validation process.

4.3.4.2 Mapping to agile-XP

XP includes several testing practices during the software life cycle, which can be classified as follows:

- Unit testing: The unit test is performed during the agile-XP life cycle to verify individual programming units or software components (i.e. classes). For every class in the system, developers should write a test case that verifies the functionality of that

class. When a change has been made to a class, the developers should update the corresponding test case and then test the modified class.

- Integration testing: when a new class is coded and ready to be integrated into the system, developers have to verify the whole of the current system after integration. Thus, all the previous and current test units must run correctly during the integration process.
- Acceptance testing: acceptance tests are usually specified by the customer to verify that the overall system can meet the specified requirements. The test is performed on a story basis; the story is considered to be complete if it passes the entire specified customer acceptance test. Ideally, acceptance tests should be automated, either by using the unit testing framework or a separate acceptance testing framework.

Table 4.5 illustrates the mapping results between the ISO requirements during design and development verification and validation activities and agile-XP.

Table 4.5 ISO 9001 Construction phase mapping

ISO Requirements	ISO 12207 support activity	Mapping to agile-XP
<p>ISO requirements at Design and Development Verification and Validation (Section 4.3.4.1):</p> <p>The verification process should be performed at various design stages to ensure that the product meets the specified requirements</p> <p>The validation process should be performed at the end of the development process and before the delivery of product to the user</p>	<p>Verification process (Section 6.4)</p> <p>and</p> <p>Validation process (Section 6.5)</p>	<p>- Unit testing</p> <p>- Integration testing</p> <p>-Acceptance testing</p>

4.4 Summary

This chapter has investigated the capability of agile-XP to implement the software process related to the requirements in ISO 9001 and the guidelines in ISO 90003. The requirements of ISO 9001 and guidelines of ISO 90003 have been extracted using the ISO 12207 terminology. Agile-XP supports partially the ISO 9001 extracted requirements and following comments can be made:

- The main technique for documenting user requirements in agile-XP is the user story. However, the user story provides fewer details than what is specified by ISO 9001 and ISO 90003. For example, it records a high-level description of user requirements and keeps the details for face-to-face communication with the user during the iterations. Moreover, the user story does not take into account the system requirements or any of the technical details needed during development. Also, it is not clear how agile-XP can trace the requirements back to the final product.
- User stories are mainly written in a natural language and formal specifications are not provided by user stories, thus requirement are evaluated by prototypes and on-site customers. Formal evaluations such as model validations are not supported by agile-XP.
- ISO 9001 requires that developers select suitable review methods during the design and development process. The level of formality of the selected review method should be relevant to the project complexity. Agile-XP reviews are mainly based on pair programming, where modifications are made based on the programmers' decisions and no documentation is provided. The reviewing activities in agile-XP lack any formal reviewing methods, such as inspection and walkthrough, and so it fails to provide the ISO 9001-required documents at this stage.
- Agile-XP does not record the appropriate steps for resolving the anomalies found during unit testing; integration testing and acceptance testing. The testing activities in agile-XP are mainly based on test cases and do not provide documented evidence to ISO 9001 auditors on how these testing activities have been planned, scheduled, and carried out

throughout the software life cycle. As a result, it fails to satisfy the ISO 9001 requirements at this stage.

CHAPTER 5

EXTENDING AGILE-XP USER STORIES TO MEET ISO 9001 FORMALITY REQUIREMENTS

5.1 Introduction

The advantages of ISO 9001 certification are understood by some software organizations Fuller (2006). Recently, however, the market penetration of the documentation-light agile software processes (e.g. extreme programming – XP) has been increasing Schindler (2008), Vijayasathy and Turk (2008). "Agile development processes have a different perspective compared to traditional development processes which follow a more linear or waterfall model for performing tasks. One of the differences is that a detailed requirements specification may be missing during a large part of the project or even the whole project duration. Some other differences include the use of stories as a source for requirements. Stories include many details and may be more ambiguous than the conventional requirements specification. A story may also be coarser grained than the traditional requirements specification" Espinoza and Garbajosa (2011).

This chapter proposes four sub processes (activities) aligned with the XP release planning phase. These sub processes are: 1) identification of the user story source; 2) identification of a non functional requirements category; 3) identification of user story relationships; and 4) identification of user story priorities. The aim of these sub processes is to modify the structure of traditional user stories in order to provide the ISO 9001 auditor of XP with sufficient evidence that the data they require have been collected, and to provide traceability for the requirements throughout the earliest phase of XP (i.e. the release planning phase).

This chapter is organized as follows:

Section 5.2 clarifies the main terms and definitions that will be used in this chapter.

Section 5.3 presents the design process for the user stories extension.

Section 5.4 describes in detail each of the proposed sub processes.

Section 5.5 describes the main structure of the extended user story based on the proposed sub processes.

Section 5.6 presents a summary and discusses the potential benefits of this work from the ISO 9001 viewpoint.

5.2. Terminology

This section presents the definitions of the terms that will be used in this chapter.

5.2.1 System

A system is defined by ISO 15288:2008 as a combination of interacting elements organized to achieve one or more stated purposes. An element is a discrete part of the system that can be implemented to fulfill specified requirements, and can be hardware, software, data, humans, or processes (e.g. processes for providing a service to users). In this context, the system is viewed as a collection of interacting elements organized to accomplish a specific function, or set of functions, within a specific environment.

5.2.2 System feature and system function from the XP viewpoint

The differences between a system feature and a system function are poorly defined in the literature. In XP, a user story is designed to specify a goal from the user viewpoint and to specify a feature from the system viewpoint. As a result, user stories often represent user needs, which will ultimately include both essential and nice-to-have features. The collection of those features will be integrated later in the process life cycle into system elements to provide a function to the system. Every XP iteration provides the system with functionality, based on the collection of features originally implemented based on the user

stories. For example, Add User, Grant Privilege to User, Delete User, and List Users are system features that can be represented at the requirements level by means of a user story. The result of implementing user stories is a system function, such as a 'user administration system'. Figure 5.1 illustrates the concepts of feature, function, and system from the XP perspective.

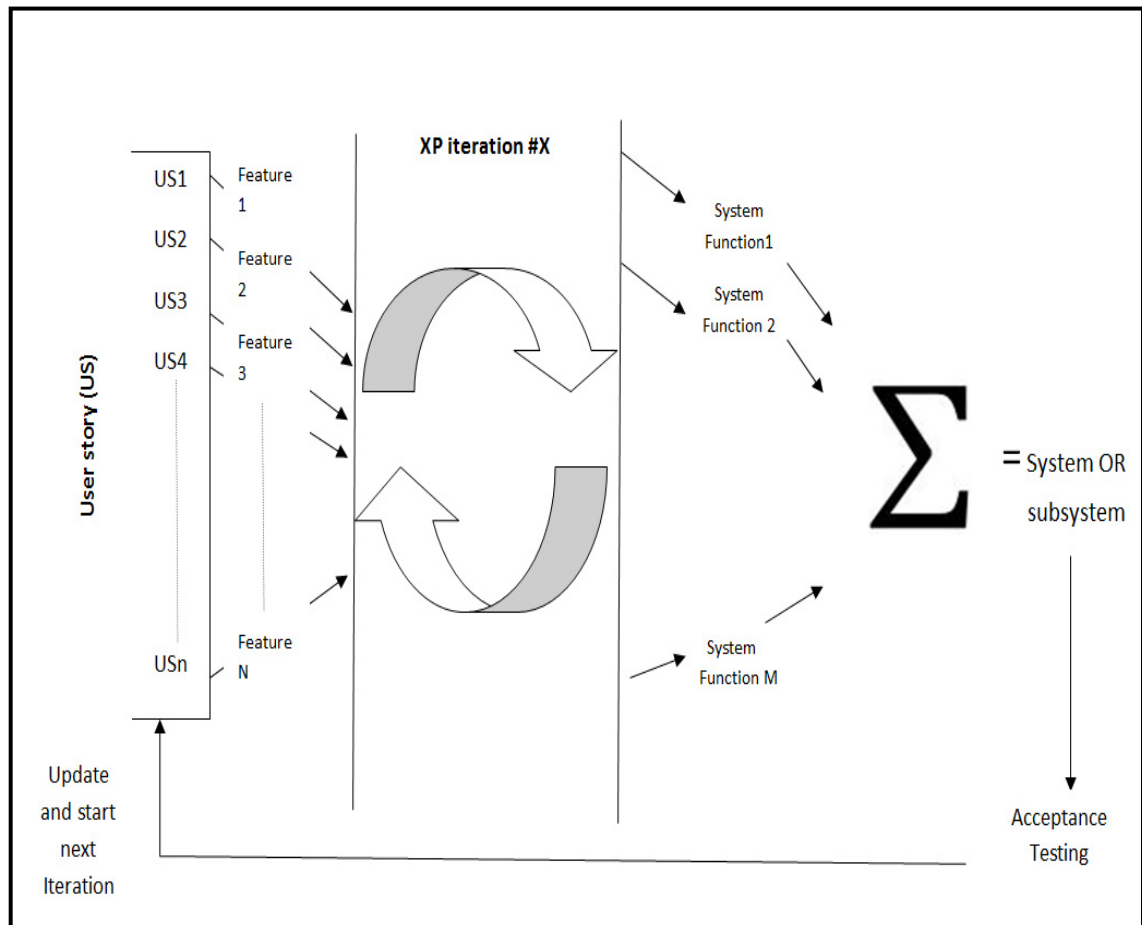


Figure 5.1 Relationship between system features and system functions in XP

5.3 Design for user stories extension

The CMMI for development, version 1.2 (CMMI-DEV, v1.2), includes some process areas for identifying and managing software requirements, and contains useful guidelines and best practices for specifying them. In the context of this chapter, three different CMMI process

areas (i.e. requirement development, requirement management, and risk management) have been analyzed to derive a set of sub processes that could be aligned with the exploration phase of XP release planning – see Figure 5.2.

The objectives of these sub processes can be summarized as follows:

- ✓ Provide the basic metadata for managing the information gathered during XP release planning.
- ✓ Set a standard for the information and the data gathered during XP release planning; this will allow a relationship to be defined between user stories.
- ✓ Provide structured user stories that can present more information concerning dependencies between user stories and other artifacts of the XP development life cycle.
- ✓ Provide standardization across XP processes to support user story management. Standardizing user story cards, for example, will help raise the visibility of the process of capturing both functional and non functional requirements.
- ✓ Provide more information about stakeholders and the source of user stories; this will allow better decisions to be made, development times to be reduced, customer satisfaction to be improved, and the basic information for supporting XP traceability to be provided.

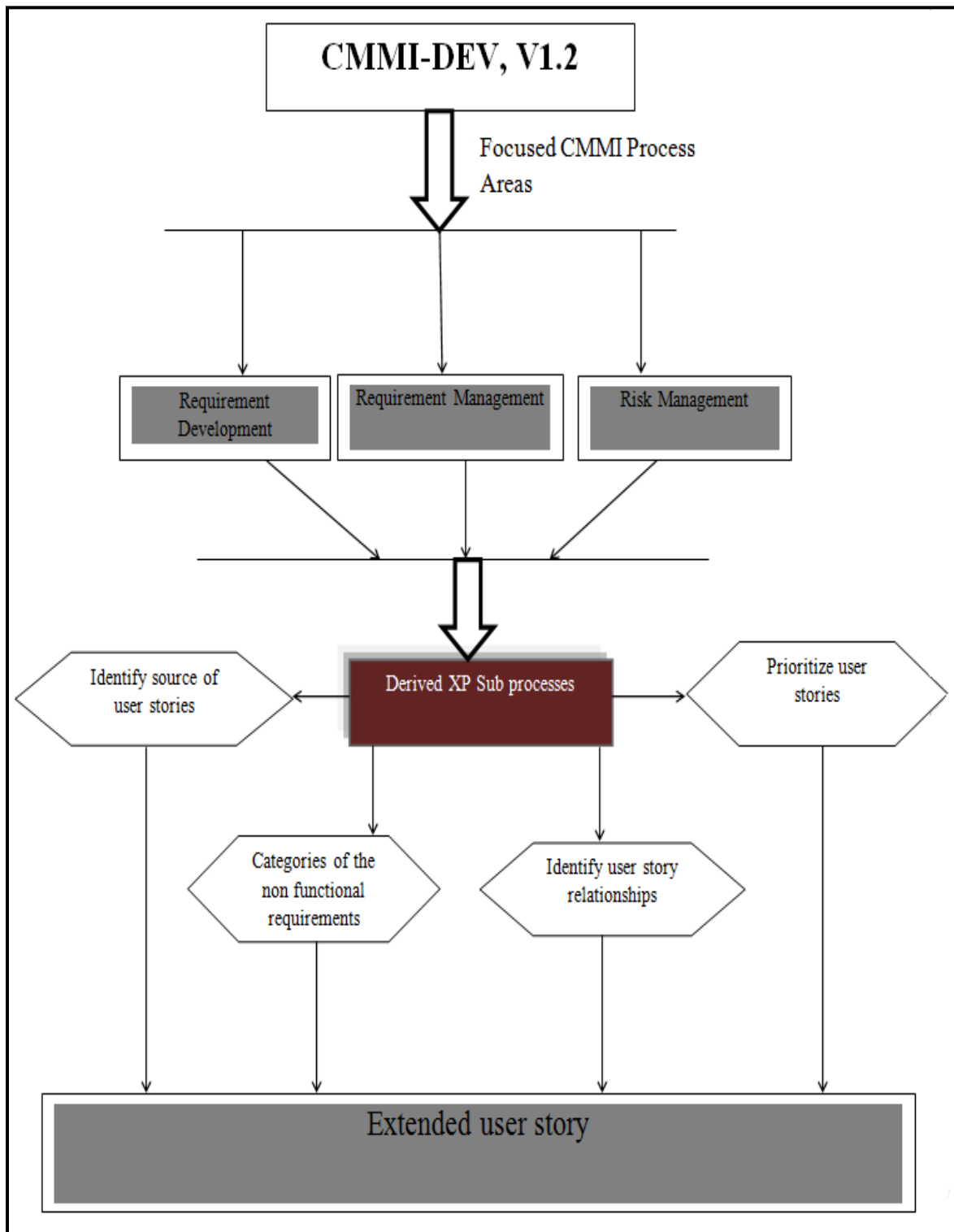


Figure 5.2 Methodology for deriving the XP sub-processes

Table 5.1 shows each process area and the process goals that have been investigated, as well as the related derived XP sub processes. Mapping here is not one to one such that, multiple process goals from different CMMI key process area (KPA) have been investigated to derive one XP sub process.

Table 5.1 Derived XP sub processes

CMMI Process Areas Investigated	Process Goal	Derived XP Sub processes
Requirement development (RD)	Elicit needs Develop customer requirements Establish a definition of required functionality Analyze requirements to achieve balance	The following Sub processes have been derived based on the (RE) process goal: <ul style="list-style-type: none"> - Nonfunctional requirements categorization - User story prioritization
Requirement management (REQM)	Understand requirements Obtain commitment to requirements Manage changes to requirements	The following Sub processes have been derived based on the (REQM) process goal: <ul style="list-style-type: none"> - Identify source of user stories - User story relationships
Risk management (RM)	Determine risk sources and categories Identify risks Evaluate, categorize, and prioritize risks	The following Sub processes have been derived based on the (REQM) process goal: <ul style="list-style-type: none"> - Identify source of user stories - Identify user story relationships - Prioritize user stories

5.4 Proposed sub processes

5.4.1 Identify the source of the user story

The requirements engineering process focuses on stakeholder needs. The goal is to identify all the people, organizations, and other systems that have a direct or indirect impact on the user stories elicited. Much software has proved unsatisfactory because it has stressed the requirements of one group of stakeholders at the expense of those of others. Hence, software is delivered which is difficult to use or which subverts the cultural or political structures of the customer organization.

The software engineer needs to identify, represent, and manage the ‘viewpoints’ of many different types of stakeholders” Abran, Moore *et al.* (2004). Software development teams should understand the sources that directly or indirectly influence the creation of user stories, in order to be able to trace each story back to its original source in the case of an improvement or change request. Therefore, the <<STORY CONTRIBUTOR>> is defined as individuals, including the customers or clients who pay for the system, the developers who design, construct, and maintain the system, and the users who interact with the system to get their work done, as well as other systems or organizations that need to collaborate with the system. The schema proposed by Glinz and Wieringa (2007) has been used to identify the <<STORY CONTRIBUTOR>> from the ISO 9001 perspective. The authors Glinz and Wieringa (2007) suggest a list (provided below) of candidate stakeholders who may contribute to the progress of any software project, i.e. people who:

- ✓ manage, introduce, operate, or maintain the system after its deployment;
- ✓ are involved in developing the system, including architects, developers, testers, quality engineers, or project managers;
- ✓ are responsible for the business or process that the system supports;
- ✓ have a financial interest (for example, they paid for it or are responsible for selling it);

- ✓ constrain the system as regulators (for example, through the laws and international software standards such as ISO 9001 that may impact the system).

Usually, the <<STORY CONTRIBUTOR>> varies according to the nature of the system being developed; for example, the system may be intended to provide special services inside the organization, such as a payroll system or documentation management system, or perhaps the system is related to public services, such as air traffic control. ISO 9001 requires these <<STORY CONTRIBUTORS>> to be clearly identified and categorized. Therefore, to improve the accuracy of the user story, it has been proposed that its source, i.e. the <<STORY CONTRIBUTOR>>, belongs to one or more contributor types – see Figure 5.3 – which have been developed based on Abran, Moore *et al.* (2004), Glinz and Wieringa (2007) and ISO 9001:

- Customer side contributors,
- Development side contributors, and
- Government side contributors.

<<STORY CONTRIBUTORS>> are assumed to provide the features of their system that could affect the various levels of the system, such as the process level, the product level, and the project level. While this list is not exhaustive, it does provide guidance to help in identifying the source of the user stories – see Figure 5.3.

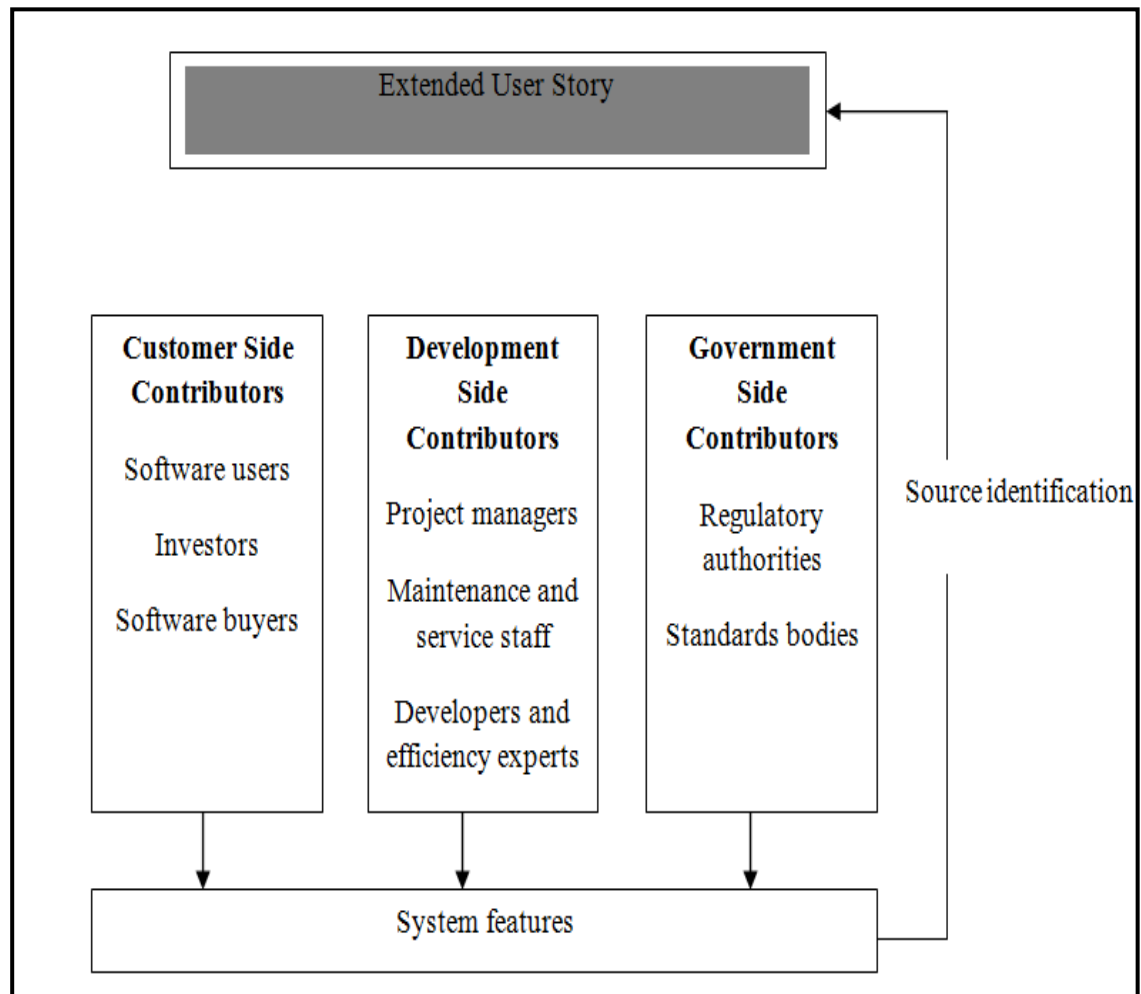


Figure 5.3 User story sources – the various types of contributors

- **Customer side contributors**

- ✓ **Software users:** Those with a direct interest in the functions provided by the proposed new system or services. Software users are valuable sources of knowledge of the features that the system is designed to implement. They can provide insights into how the system should operate.
- ✓ **Investors:** Those responsible for providing the required funding for the proposed system, including the organizations responsible for developing the system or an external party wishing to invest in the system. These contributors may have their own features that they consider would better implement the system's user stories. Usually, features provided by investors are related to system efficiency and to the performance

of the system. The investors play an important role in balancing, and scoping, costs and perceived benefits.

- ✓ Software buyers: Those who purchase large and complex software (such as air traffic control system or online banking system), and who could be different from the users of the software. System features from these contributors are derived from their own expectations on how to better support user needs.

- **Development side contributors**

- ✓ Project managers: Those responsible for managing the technical aspects of the project (e.g. the development process) and its non technical aspects (e.g. budget and development time). Requirements and constraints from project managers are focused as much on bringing discipline to the delivery schedule as to moving the project on to successful completion within the specified budget. Requirements from project managers are usually related to regulating the workflow of the project and focus less on system features.
- ✓ Maintenance and service staff: Those whose main responsibility is to keep the system operating after it has been delivered to the system users. Requirements from these contributors are focused on a set of controls designed to better maintain the system later.
- ✓ Developers and the quality assurance team: Those whose main responsibility is to design, implement, and test the system, and to verify that all the system user stories from all the story contributors have been implemented efficiently. They focus on the overview at the application level, rather than at the component level or individual programming task level. Therefore, they may contribute stories to the system concerning controls and indicators for monitoring and measuring the various characteristics and sub characteristics of system quality.

- **Government side contributors**

- ✓ Regulatory authorities and standards bodies: To ensure the compliance of organizations with codes of practice, government regulations, etc., such as Sarbanes-

Oxley (SOX), the Food and Drug Administration (FDA), and the Health Insurance Portability and Accountability Act (HIPAA). It is the responsibility of every organization to develop its own business processes to address them, and the SWEBOK Guide recognizes that a software development process might be a part of such a business process Abran, Moore *et al.* (2004). The SWEBOK Guide also points out that there is broad acceptance that software development success is highly dependent on the software requirement activities. Therefore, user stories should be able to capture and manage the requirements (functional and non functional) from the government side contributors. At the business process level, organizations react to the regulatory authorities and standards bodies by developing what are called internal controls (i.e. policies and procedures). “Software is often required to support a business process, the selection of which may be conditioned by the structure, culture, and internal politics of the organization” Abran, Moore *et al.* (2004). An organizational policy can be described as a formal statement that guides and steers production methodologies, and so every organization must ensure that their policies comply with the rules of the authority that governs it. An organizational procedure is a series of steps required to implement the organization’s policies. It is essential, therefore, that software developers analyze the applicable rules for implementing the organization’s internal controls. From the software engineering perspective, these internal controls are translated into application support software and control support software – see Figure 5.4.

- Application support software is software that provides a specific set of user-level functions, such as a reporting system or an employment management system.
- Control support software is software that automates the organizational policies and procedures, or provides technical services to the organization.

Control support software includes control components, which can be classified as follows:

- Application level control component: a control element implemented and integrated into the system for a specific automated service; for example, services to ensure that all goods shipped are invoiced.
- Process level control component: a control element implemented and integrated into the system to support the overall business process; it includes adequate security functionality to prevent unauthorized access to secure applications.
- Technical level control component: a control element implemented to support the organization at the operational level; for example, implement the organization's internal policies or procedures, or to ensure that policies and procedures are implemented by the operational system and business processes.

To this end, user stories should capture the sources of the requirements from the government side contributors for the regulatory authorities and standards bodies, in order to ensure that a software system is capable of meeting government and business requirements, and to provide the ISO 9001 certifying authority with evidence that data from those sources have been collected.

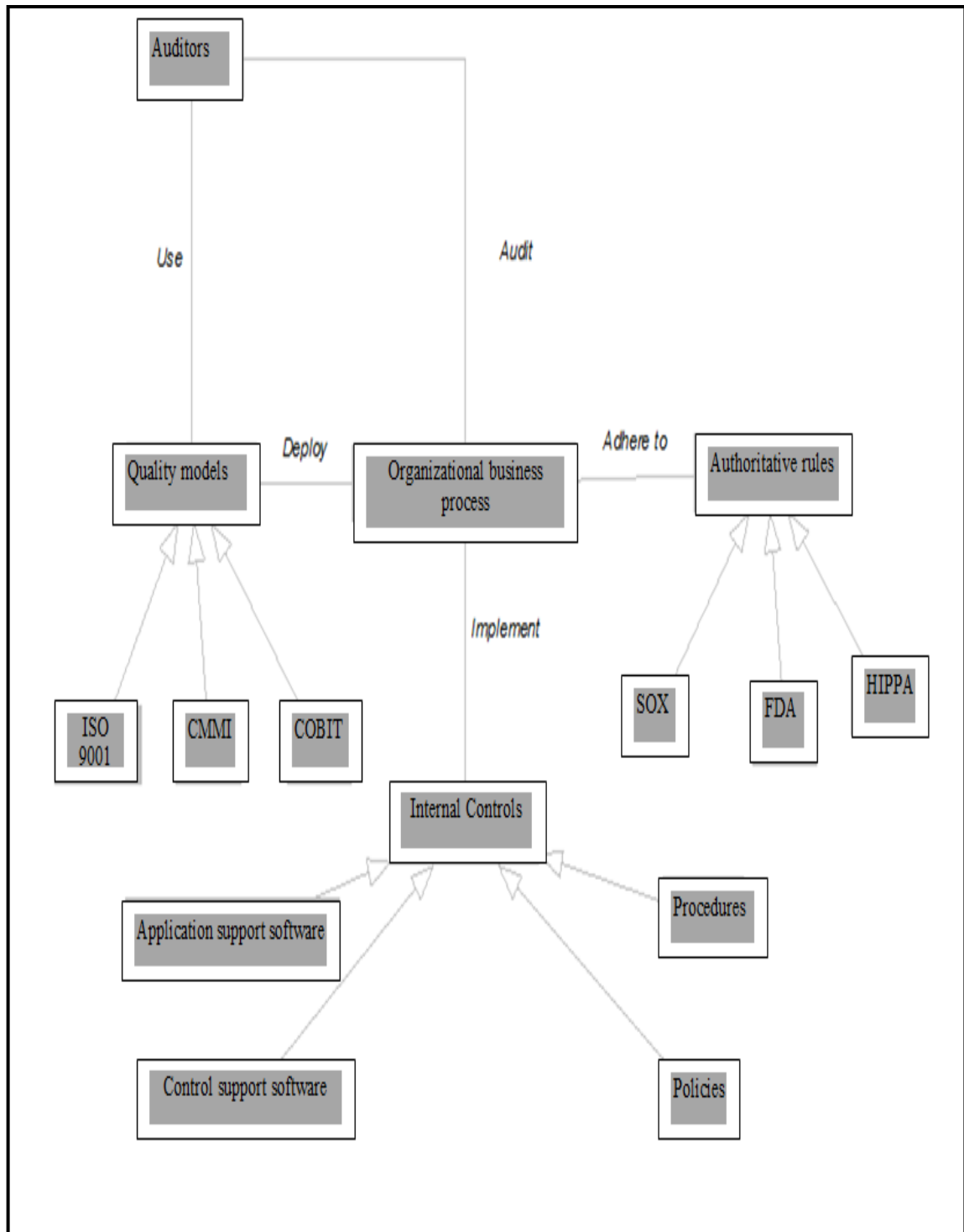


Figure 5.4 Government side contributors

5.4.2 Categories of non functional requirements

The goal of this section is to provide formal evidence that the non functional requirements have been gathered from the user stories and categorized based on their respective groups (a related work on the standards-based specification of non functional requirements can be found in (Abran, Al-Sarayreh *et al*, 2010), (Al-Sarayreh, Abran *et al*, 2010)).

During XP release planning, the <<STORY CONTRIBUTOR>> informally states the non functional requirements that need to be considered for each user story. Every <<STORY CONTRIBUTOR>> sees the problem from a different perspective. As users often do not know which quality attributes they would like to see included, they can express their non functional requirements orally (Tracy, Sarah *et al*, 2008). Developers must therefore be able to understand and categorize those non functional requirements and map them to the corresponding quality attribute(s) in order to comprehend the entire problem domain.

To enhance the ability of user stories to capture non functional requirements during the early phases of XP, a semi structured format is proposed for defining them. This allows developers to identify the category to which the non functional requirements of each user story belong, as well as to provide a flexible format for both the functional and non functional requirements. The set of quality attributes is represented in the format {Q1,Q2,...Qn}, and the sub quality attributes associated with the non functional requirements required by a user story in the format {SQ1,SQ2,...SQn}. Also, there are many quality models that address the quality attributes and non functional requirements of software systems, such as the European Cooperation on Space Standardization (ECSS), Boehm, McCall, and ISO 9126 models. The ISO 9126 quality model refers to six quality characteristics, subdivided into twenty-seven quality sub characteristics for the internal and external quality of a software product – see Table 5.2.

Table 5.2 ISO 9126 quality characteristics

Characteristics	Sub characteristics
<i>Functionality</i>	<i>Suitability</i> <i>Accuracy</i> <i>Interoperability</i> <i>Compliance</i> <i>Security</i>
<i>Reliability</i>	<i>Maturity</i> <i>Recoverability</i> <i>Fault Tolerance</i>
<i>Usability</i>	<i>Learnability</i> <i>Understandability</i> <i>Operability</i>
<i>Efficiency</i>	<i>Time behavior</i> <i>Resource behavior</i>
<i>Maintainability</i>	<i>Stability</i> <i>Analyzability</i> <i>Changeability</i> <i>Testability</i>
<i>Portability</i>	<i>Installability</i> <i>Conformance</i> <i>Replaceability</i> <i>Adaptability</i>

AS a <<STORY CONTRIBUTOR>>, I want the system to <<DO REQUIREMENTS>>
 AND incorporate <<NON FUNCTIONAL CAPABILITIES>>, which belong to
 Quality characteristics {Q1, Q2.....,Qn} AND
 Sub quality characteristics {SQ1, SQ2,.....SQm} respectively

Each user story is primarily associated with a <<NON FUNCTIONAL CAPABILITY>> entity that represents the category of non functional requirement intended for each story. Information related to the user story <<NON FUNCTIONAL CAPABILITY>> should be obtained during exploration phase of agile-XP where the information related to the product is collected. In agile-XP these information are collected during a face to face session with the user.

The purpose of a <<NON FUNCTIONAL CAPABILITY>> entity is to keep the user story as lightweight as possible, but at the same time to provide evidence for an ISO 9001 auditor that non functional requirements have been obtained during the early phases of XP. The <<NON FUNCTIONAL CAPABILITY>> category could represent one or more quality characteristics and sub quality characteristics belonging to the non functional requirements stipulated by the <<STORY CONTRIBUTOR>>. Table 5.3 shows examples of non functional capability categories.

Table 5.3 Examples of non functional capability categories

Example	<<NON FUNCTIONAL CAPABILITY>>
The customer must place an order within two minutes of registering.	Performance
The customer must be able to access their account 24 hours a day, 7 days a week.	Availability
"Update Customer" will be available to users during 98% of normal working hours.	Reliability
Up to 200 new sites per year may start to use "Update Customer".	Scalability

5.4.3 Identify the user story relationships

Based on the description of system features and system functionality in the previous section, we next define the relationships between dependent user stories. For example, a user story “j” that depends on another user story “i” is called dependent, and is denoted <US,j>. Such a pair of dependent user stories will be read as follows: <US,j> depends on <US,i>. The dependencies between user stories are then classified into four categories: logical dependencies, data dependencies, temporal dependencies, and resource dependencies. This classification is based on the user story features that require implementation.

- **A logical dependency** occurs when the feature implemented by a user story X cannot be executed before the feature implemented by user story Y, because they are logically dependent. This can be the case if user story X provides services or interfaces to user story Y. For ex-ample, in an employment management system, the employee will not

be granted access to perform restricted operations unless he has been approved as a legitimate employee. This can be read as follows: $\langle US, j \rangle$ logically depends on $\langle US, i \rangle$. This relation can be represented as in Figure 5.5.

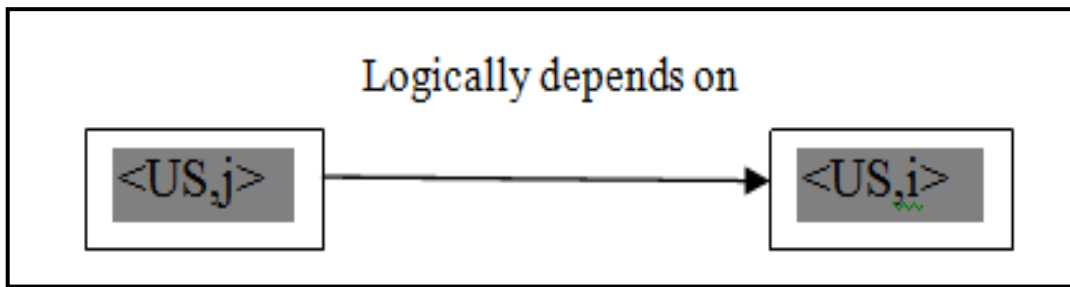


Figure 5.5 Logical dependency

- A **data dependency** occurs if the feature implemented by user story X cannot be executed before the feature implemented by user story Y, because they are data dependent. This can be the case if user story X provides input data for user story Y. For example, sorting the entries in the database should be performed after this entry has been stored. This can be read as follows: $\langle US, j \rangle$ data depend on $\langle US, i \rangle$. This relation can be represented as in Figure 5.6.

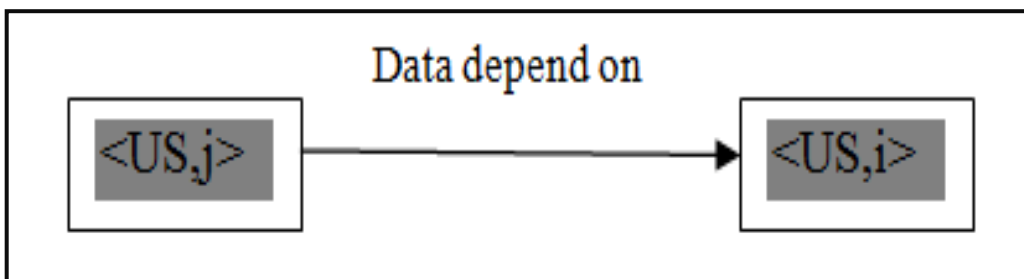


Figure 5.6 Data dependency

- A **temporal dependency** occurs if the feature implemented by user story X cannot be executed before the feature implemented by user story Y, because they are time-dependent. In this case, feature x specifies the time frame for an event to occur, for a process to be completed, or a condition to hold true, for example, in order for feature y

to start processing. Temporal dependencies can be found in designing the user stories of a real-time system, where the system features must execute respecting strict response time constraints. This can be read as follows: $\langle \text{US}, j \rangle$ depends temporally on $\langle \text{US}, i \rangle$. This relation can be represented as in Figure 5.7.

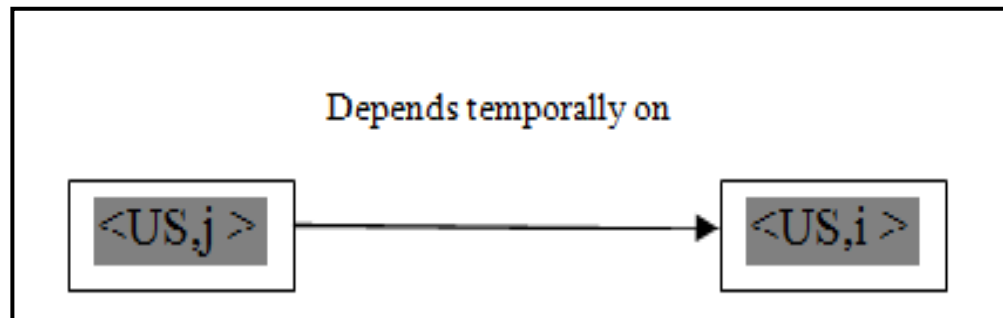


Figure 5.7 Temporal dependency

- A **resource dependency** occurs if the feature implemented by user story X cannot be executed before the feature implemented by user story Y, because they are resource dependent. In this case, the system consists of several concurrent threads (i.e. features) which are competing for limited resources (i.e. hardware resources or software resources). User stories should be analyzed first, so that precautions can be taken to ensure fairness. This can be read as follows: $\langle \text{US}, j \rangle$ resource depends on $\langle \text{US}, i \rangle$. This relation can be represented as in Figure 5.8.

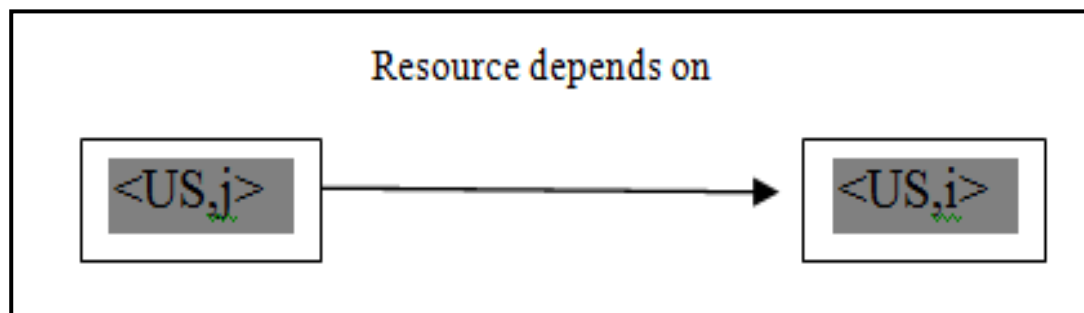


Figure 5.8 Resource dependency

5.4.4 Prioritizing the user stories

Prioritization is the process of making a choice among multiple options Karlsson, Berander *et al.* (2008). It is also considered an important activity in requirements engineering, as it helps developers analyze requirements in order to rank them according to their importance from the perspective of the requirements analyzer or the stakeholder who is involved in the requirements elicitation activity Lehtola and Kauppinen (2004).

Requirement prioritization processes can be categorized into methods-based solutions and negotiation-based solutions. Methods-based solutions are aimed at assigning quantitative values to the requirements, such as the binary priority list methods in Bebensee, Weerd *et al.* (2010), while negotiation-based solutions focus on resolving conflicts by brokering an agreement between stakeholders on ranking requirements using a method selection framework designed for the purpose, such as the Negotiation Constellations in Fricker and Grünbacher (2008).

In XP, user stories are usually prioritized before each iteration during the exploration phase of release planning, specifically in the Planning Game activity, in which the on-site customer classifies the user stories into three groups: “those without which the system will not function,” “those that are less essential, but provide significant business value,” and “those [it] would be nice to have” (Abrahamsson, Solo *et al.*, 2000). This XP activity can be considered as a type of negotiation-based solution that is less formal from the ISO 9001 perspective and which normally provides evidence that criteria have been met by the on-site customer on sorting the user stories into their corresponding categories. Therefore, we propose that the AHP (Analytic Hierarchy Process) be integrated into the XP Planning Game, for the following reasons:

- The AHP combines the advantages of both the methods-based solutions and the negotiation-based solutions, in that the developers, along with any <<STORY CONTRIBUTORS>>, can set the criteria for ranking the user stories into “important” and “less important” stories, based on qualitative and quantitative analysis Forman and

Selly (1996).

- The AHP provides formal evidence that the user stories have been evaluated using criteria which have been determined to support the priority given by the <<STORY CONTRIBUTOR>> to the various alternatives (such as time, costs, risks, etc.).
- The result of the AHP is highly correlated to the criteria and to the <<STORY CONTRIBUTOR>> viewpoint of what is “important” and “less important”. Therefore, developers should establish criteria that balance the goals of the project from different business value perspectives.

Figure 5.9 depicts the procedure for prioritizing the user stories in XP using the AHP method.

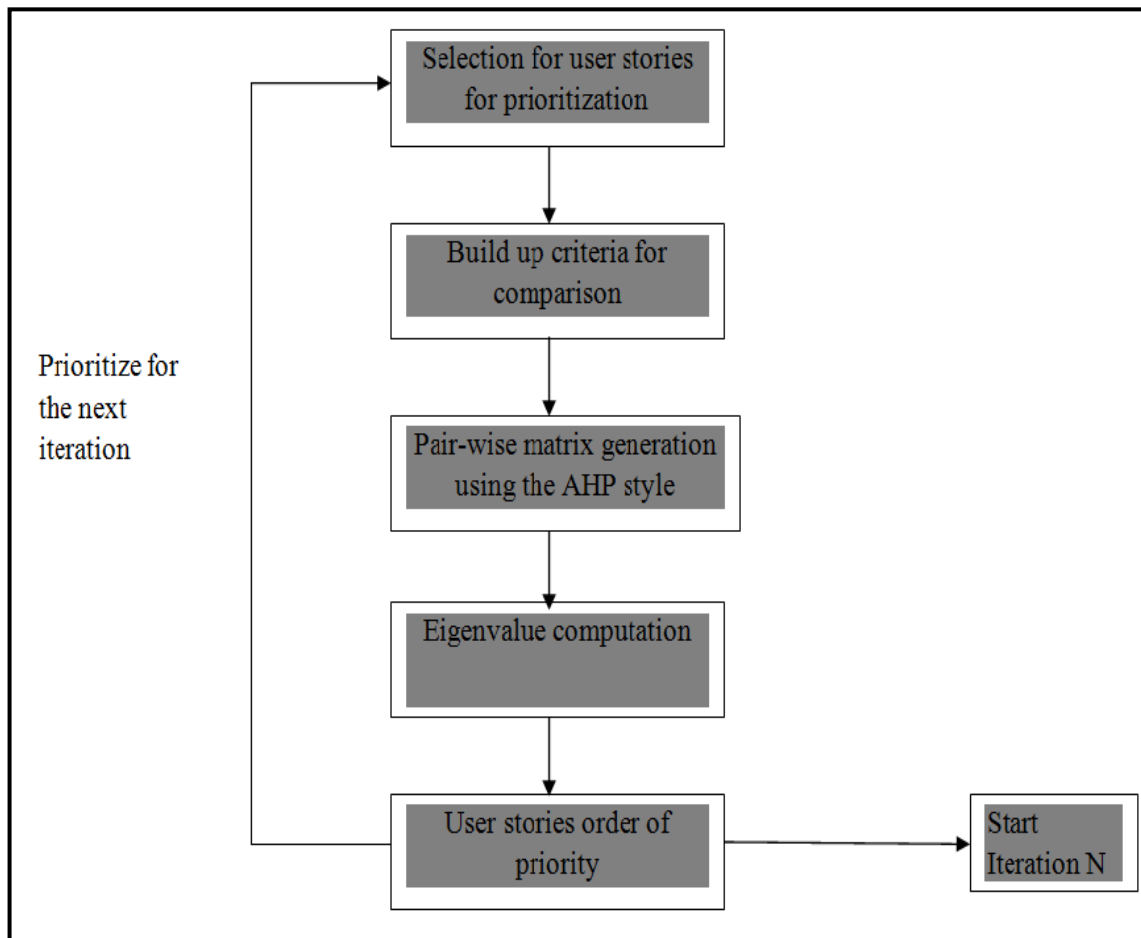


Figure 5.9 Procedure for prioritizing the user stories in XP using the AHP

- **Selection of user stories for prioritization**

The AHP process begins by defining a set of alternatives from which a decision maker wants to choose (e.g. selection of faculty members, assessment of financial management models, etc.) Grandzol (2005). There is a variety of methods available for generating those alternatives, such as a brainstorming session, a literature review, or the outcome of a specific process, such as release planning in XP, where the developers, in consultation with the customer, come up with a set of user stories that need to be implemented in subsequent iterations.

At the beginning of each iteration of the exploration phase in XP release planning, the

developer gets together with the customer for a planning meeting. In that meeting, they go over the features the customer wants to implement in that iteration, breaking each feature down into individual engineering tasks. In this step, the developers are required to determine the set of user stories that need to become input for AHP prioritization.

- **Building up criteria for comparison purposes**

The AHP allows developers to model the user story ranking as a hierarchical structure, as shown in Figure 5.10. Using AHP, the definition of criteria is based on the decision maker's viewpoint of what is important from his perspective in evaluating and prioritizing the alternatives. In the context of this chapter, each <<STORY CONTRIBUTOR>> can generate his own criteria for ranking the set of user stories. Therefore, the customer side contributors, the development side contributors, and the government side contributors can all generate criteria that can be used to consider different aspects of user story evaluation, such as financial benefits, strategic benefits, competitors, the ability to adhere to standards or regulations, the ability to sell, etc. Next, we give some examples of criteria for developing user stories that consider cost, time, and risk:

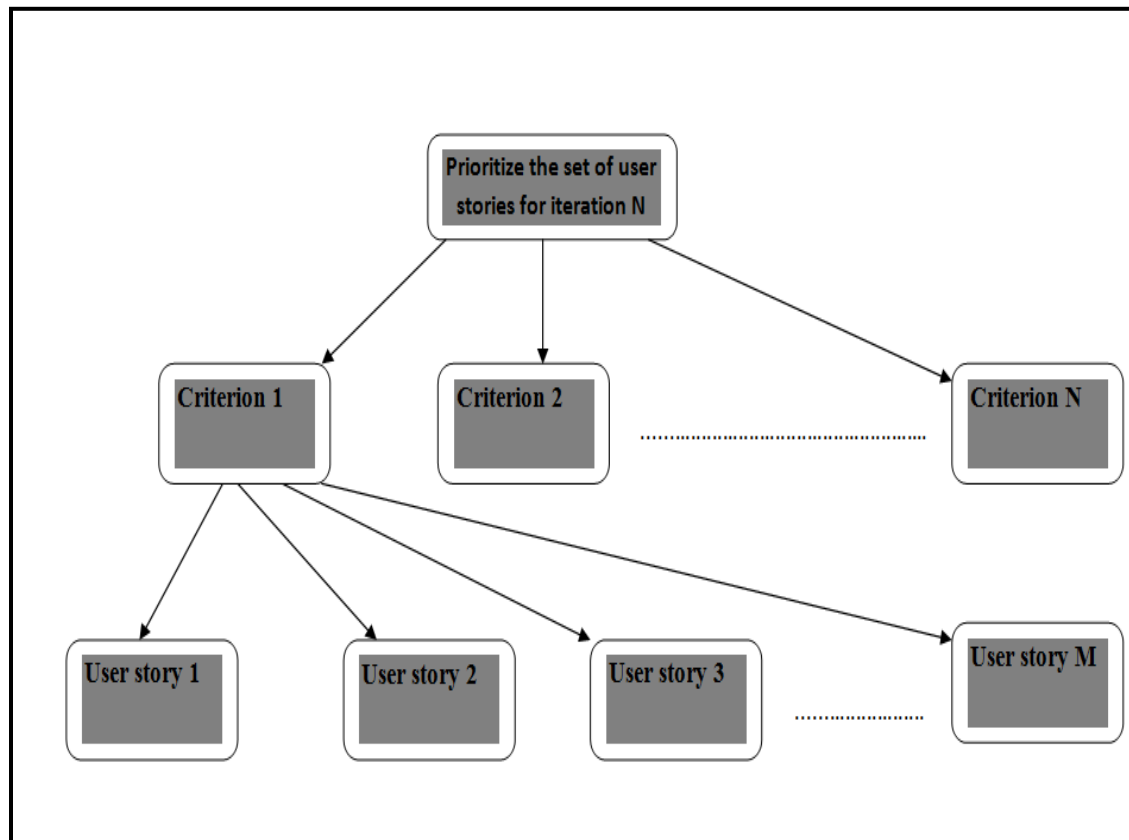


Figure 5.10 AHP diagram for user story selection

- ✓ Cost is often expressed in terms of the number of hours spent developing the software. It is determined by considering the criticality of the requirements and the quality required Berander and Andrews (2005).
- ✓ Cost is often calculated in terms of hours, which is directly related to time. Time is in turn influenced by factors such as degree of parallelism in development, training needs, the need to develop support infrastructure, the need to meet industry standards, etc. Berander and Andrews (2005).
- ✓ There is a degree of risk in every project. Risk management is a process for planning ways to handle the risks that may cause difficulties in development. Among the risks that may be encountered are those related to performance, financial managements, and scheduling, for example. Calculating the risk per requirement enables engineers to forecast the potential risk at project level Berander and Andrews (2005).

- **Pair-wise matrix generation using the AHP style**

Using the AHP pairwise comparison process, weights or priorities are assigned to a set of human judgments based on the AHP scale in Table 5.4. While it is difficult to justify weights that are arbitrarily assigned, it is relatively easy to justify judgments and the basis for those judgments Forman and Selly (1996).

The concept of pairwise comparison for prioritizing user stories works as follows: developers begin by computing the priority of their criteria, which are cost, time, and risk in this context. The first step is to generate a pairwise matrix by comparing these three criteria, according to the scale in Table 5.4.

Table 5.4 AHP scale

Intensity of importance	Definition
1	Equally important
3	One moderately more important than the other
5	Much more important
7	Very much more important
9	Extremely important

Assume that the following relationships have been determined for these criteria:

- ✓ Cost is much more important than Time (degree of importance: 5).
- ✓ Cost is moderately more important than Risk (degree of importance: 3).
- ✓ Risk is very much more important than Time (degree of importance: 7).

Then, the following pairwise matrix will be generated – see Table 5.5.

Table 5.5 Pairwise matrix for the selected criteria

$A_{\text{criteria}} =$		Cost	Time	Risk
	Cost	1	5	3
	Time	1/5	1	1/7
	Risk	1/3	7	1

Suppose the developers intended to rank three different user stories: <US1>, <US2>, and <US3>. The pair-wise matrix for each criterion should be generated as in Tables 5.6, 5.7, and 5.8.

Table 5.6 Pairwise matrix for the cost criterion

$A_{\text{cost}} =$		US1	US2	US3
	US1	1	3	5
	US2	1/3	1	1/7
	US3	1/5	7	1

Table 5.7 Pairwise matrix for time criterion

$A_{\text{time}} =$		US1	US2	US3
	US1	1	9	3
	US2	1/9	1	1/5
	US3	1/3	5	1

Table 5.8 Pairwise matrix for the risk criterion

$A_{\text{risk}} =$		US1	US1	US1
	US1	1	3	5
	US2	1/3	1	1/3
	US3	1/5	3	1

- Eigenvalue computation**

The AHP obtains the weight vector (priority vector) by calculating the eigenvector for the largest eigenvalue of matrix A. This can be obtained using formula (1) where w is the eigenvector and λ is the eigenvalue corresponding to that vector.

$$Aw = \lambda w \dots\dots\dots(1)$$

By solving (1) for $A_{criteria}$, A_{cost} , A_{time} , and A_{risk} , the priority hierarchy will be generated as in Figure 5.11.

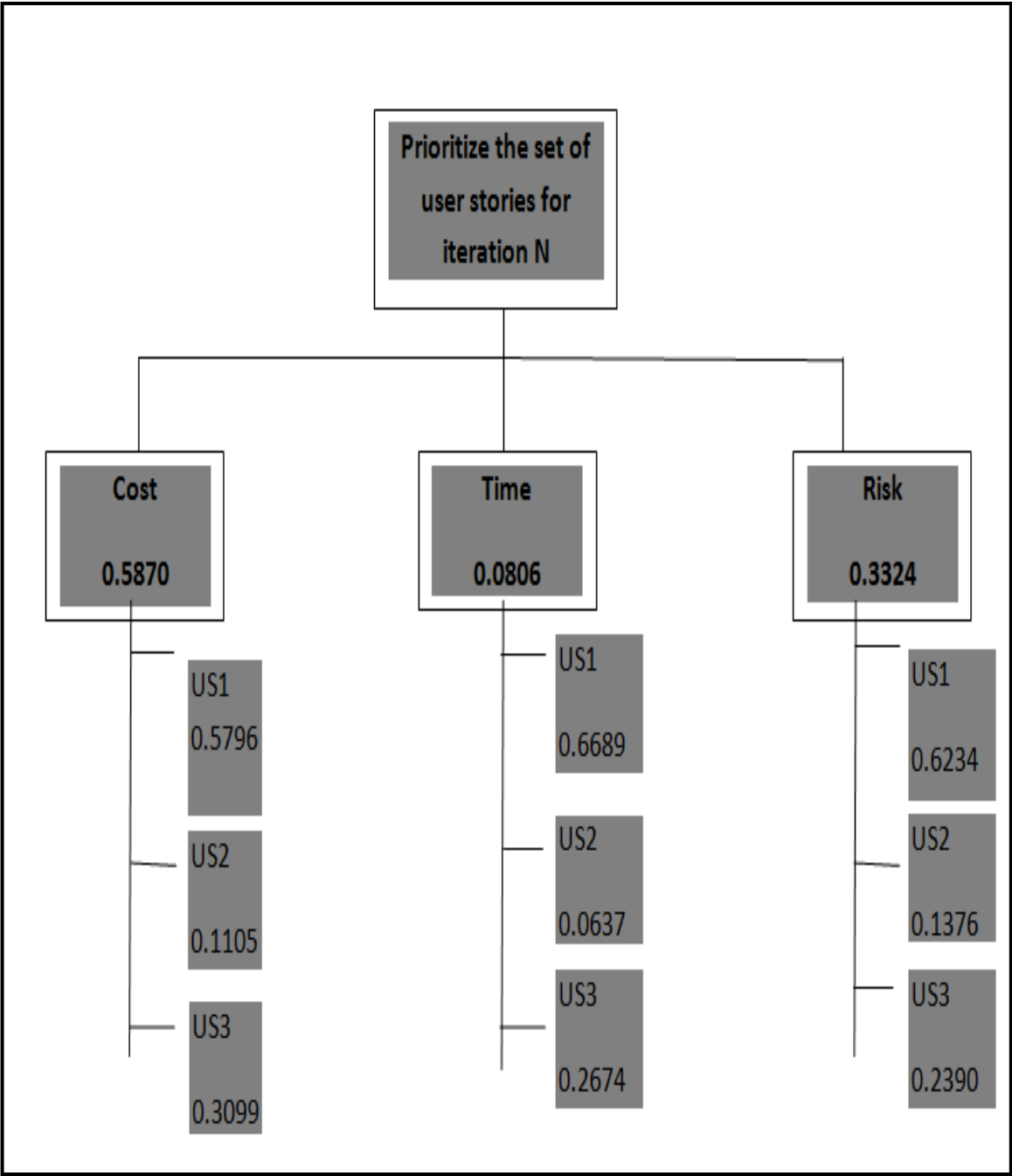


Figure 5.11 A priority hierarchy

$US1_{priority}$, $US2_{priority}$, and $US3_{priority}$ can be obtained as of Figure 5.12.

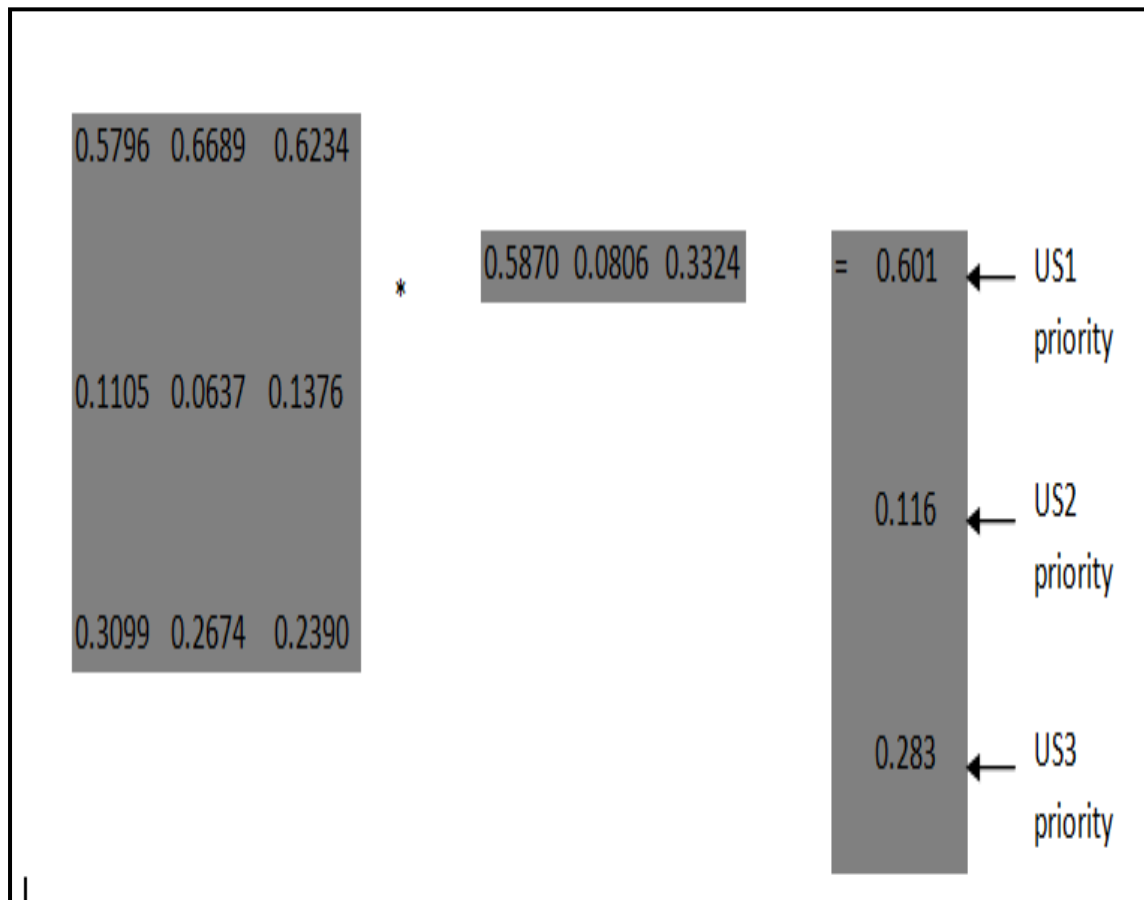


Figure 5.12 Calculation of user stories priority

5.5. Extended user story for XP

This chapter has introduced an extension to the user story to help XP software developers in specifying important information for the ISO 9001 requirements that should be gathered in the earlier phases of software process development. The main content of the extended user story will be as follows – see Figure 5.13.

Extended user story	
Requirement	A plain to indicate the user functional requirements
<<STORY CONTRIBUTOR>>	Identification of user story sources: <<customer side contributors >><< development side contributors >><< government side contributors >>
Non Functional capability	<p>Category could represent one or more a quality characteristic and sub-quality characteristic belong to the non-functional requirements required by the << STORY CONTRIBUTOR >>.</p> <p>Quality characteristic {Q1, Q2.....,Qn} AND Sub-quality characteristic {SQ, SQ2,.....SQ2}</p>
Story Relations	Identification of user story dependencies. The pair of user stories <US _i > and <US _j > is called dependent and will be read as <US _j > depends on <US _i >, if there exists a dependency relation that belongs to one or more of the following categories: Logical dependencies, Data dependencies, Temporal dependencies, and Resource dependencies.
Priority ranking	Assign of numerical value that indicate the importance of the user story from the <<STORY CONTRIBUTOR>> point view using AHP method.

Figure 5.13 Extended user story

- **Requirements:** Identification of the user's functional requirements.
- **User story sources:** Identification of user story sources: <<customer side contributor>>, <<development side contributor>>, and/or <<government side contributor>>.
- **Non functional capability:** Identification of the non functional category that represents one or more quality attributes and sub quality attributes belonging to the non functional requirements needed by the <<STORY CONTRIBUTOR>>.

- **Story relationships:** Dependencies between the user stories are identified and classified into logical dependencies, data dependencies, temporal dependencies, and resource dependencies.
- **Priority ranking:** The priority of each user story is calculated based on the AHP method. The <<STORY CONTRIBUTOR>> can generate a priority list for user stories based on predefined criteria.

5.6. Summary and Discussion

The main contribution of this chapter is the proposed sub process, aligned with XP release planning, for deriving the extended user story. The following comments illustrate the advantages of the proposed extended user story from the ISO 9001 perspective:

- **Formality:** ISO 9001 auditors need documented evidence at every phase of the development process to clarify that processes are compliant with ISO 9001. The extended user story that we propose here will provide formal evidence that the sources of each user story have been identified. It will also provide formal evidence that each user story has been prioritized from the <<STORY CONTRIBUTOR>> viewpoint. This can be supported by showing documented evidence that every <<STORY CONTRIBUTOR>> generated comparison criteria and pairwise matrices, as well as documented evidence of the final numerical values of the priorities assigned for each user story.
- **Change management:** The extended user story can also provide support for better XP change management. For example, the identification of user story relationships and dependencies will improve the developer's ability to specify the impact of change requests on the system. Developers will be able to understand what types of dependencies exist between user stories: a change in <US,i> will generate a change in <US,j>, based on the kind of relationship that has been identified.
- **Process visibility:** The visible process has been characterized as the ability to define contact points between customers and organizations, where customers are allowed, or

even required, to interact with the process activities Yang and Vandenbosch (1998). The theory of visibility claims that organizations can improve their competitive advantage by deliberately managing the degree of visibility of their processes. Also, XP supports process visibility by mandating that on-site customers participate during the XP life cycle. The proposed sub processes allow for process visibility from the development perspective by allowing the developers to trace back every user story to its source and allowing the development team to rank user stories from the <<STORY CONTRIBUTOR>> viewpoint. This will enhance process visibility for both customers and developers.

- **Traceability:** The implementation of traceability requires software developers to identify the deliverables and artifacts of the software life cycle and provide information about the relationships between those deliverables at an early stage of the software project. This can be accomplished once the system has been divided into modules and the information flow (interaction) between these modules has been determined. The extended user story can support traceability by providing early information about the interaction of user stories based on the defined relationships of user stories (i.e. logical dependencies, data dependencies, temporal dependencies, and resource dependencies). Moreover, for large software systems that include multiple interrelated software modules, the developers can build a dependency graph that identifies the various types of interactions between the user stories.
- **Accountability:** Software project managers are responsible for ensuring that the software life cycle has been executed in conformity with ISO 9001, even before the software organization is audited by external ISO 9001 auditors. The proposed sub processes will allow software project managers to ensure that the software development activities are being performed in conformity with ISO 9001. For example, at any time in the software life cycle, the project manager can identify the source of user stories by referring to their <<STORY CONTRIBUTOR>> category. Moreover, the software project managers can find documented evidence about the non functional requirements that have been gathered during the software life cycle. The pair-wise matrices and the relationship of user stories can also provide documented

evidence for software project managers as to how the user stories interact in the system and the priority ranking for each user story.

CHAPTER 6

AN AUDIT MODEL FOR ISO 9001 TRACEABILITY REQUIREMENTS IN AGILE-XP ENVIRONMENTS

6.1. Introduction

Among the important challenges reported in the agile process (e.g. XP) literature is the traceability of the user requirements during the development process Espinoza and Garbajosa (2011), Ghazarian (2008), (Lee, Guadagno *et al*, 2003). Software traceability is defined in ISO 12207:2008 as “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another.” Ramesh and Jarke (2001) defines requirement traceability as “a characteristic of a system in which the requirements are clearly linked to their sources and to the artifacts created during the system development life cycle based on these requirements.” In agile development, verifying that the requirements have been implemented, designed, and tested in the final product depends mainly on undocumented test cases and user accepted tests, without documented evidence having been provided on how these requirements have been traced through the project life cycle. This creates challenges for software auditors, in terms of ensuring that the processes are in conformity with a specific standard, such as ISO 9001. For example, according to Cohn and Ford (2003) a manager cannot track progress in agile projects in the same way as in plan-driven projects. In plan-driven projects manager simply asks whether or not the necessary documents have been produced.

Software development-related documents constitute valuable audit evidence for Information Systems (IS) auditors. However, this is not the only type of evidence that can be obtained by the auditors: the standards, guidelines and procedures for information system auditing (ISACA, 2010) point out that other audit evidence types are also important, such as observed processes and the existence of physical items, activity and control logs, and system

flowcharts. In addition, analysis of the information through comparisons, simulations, calculations, and reasoning can also be used as audit evidence.

This chapter proposes a design of an auditing model for agile software processes (e.g. XP) based on evaluation theory, which can provide IS auditors with a methodological approach to the auditing process. The motivation for this work is to help auditors obtain evidence in conformity to ISO 9001. The proposed model is aimed at providing evidence of process traceability based on the observation of techniques and mechanisms intended to implement the traceability requirements. The proposed auditing model is designed from an engineering perspective, as we based it on an investigation of the principles of engineering design Vincenti (1990), Abran, Moore *et al.* (2004), Meridji (2010) and on the CMMI-DEV guidelines for requirement management and traceability for each audit yardstick.

This chapter is organized as follows:

Section 6.2 presents an analysis of traceability requirements in ISO 9001 and their potential advantages in software organizations.

Section 6.3 presents the design process for the auditing model and reviews the evaluation theory.

Section 6.4 presents the formulation of the auditing criteria and the yardsticks.

Finally, section 6.5 presents the summary of the chapter.

6.2. Analysis of traceability requirements in ISO 9001

ISO 9001 is a quality management standard that identifies a set of requirements designed to ensure consistency in terms of the activities, techniques, and methods used in the organization. As a result, it provides a set of requirements for the process of gathering customer needs and for creating a product that achieves customer satisfaction.

In non software organizations, such as pressure vessel manufacturers, for example, it is common for a particular material to be monitored throughout all the manufacturing stages,

and for the changes it undergoes to be recorded. In this way, the final component can be traced back to the original material. For ISO 9001, the material must be uniquely identified and the changes recorded to show evidence of traceability.

For software systems, traceability of the software process is a major requirement that has been described in ISO 9001 and in ISO 90003 in clause 7.5. Even though ISO 90003 does not elaborate on the techniques for achieving the traceability of a software process, nor does it recommend a specific method for doing so, the ISO 90003 guidelines for the application of ISO 9001 for software state that traceability is usually implemented through configuration management: “Throughout the product life cycle, there should be a process to trace the components of a software item or product, and this process may vary in scope, according to contract or marketplace requirements, from being able to place a certain change request in a specific release, to recording the destination and usage of each variant of the product”

The reasons for implementing traceability analysis are not discussed in either ISO 9001 or in the guidelines document. However, the advantages of doing so for a quality management system are described in the next sections.

6.2.1 Support for change management

Software projects are subject to dynamic changes at the technical level, such as changing software project requirements or replacing development tools, or at the managerial level, such as changing the development schedule or making changes because of budget constraints. According to Kowalczykiewicz and Weiss (2002), for larger and more complex software projects, change management practices are challenging without a traceability mechanism in place, because, at some point, the increasing number of people involved in the project and its growing size will significantly aggravate the communication difficulties between project management and developers.

The process of change management should be formalized, so that every change request follows a sequence of activities, starting with the initiation of a request for a change

(assignment of a number to the change process and acceptance of the change by the team manager) and ending with the implementation and testing of the change request. Kowalczykiewicz and Weiss (2002) maintains that the change management process should be supported with tracking techniques, so that every change request can be tracked throughout the project life cycle.

From a development team point of view, the traceability mechanism will allow the team to keep their system updated, because every requested change will be handled individually, and all the related artifacts that have been affected by the change request will be updated at the same time; for example, if a change has been made to improve a module N, then developers should ensure that all the related artifacts that have a relationship with module N are modified as well, such as a modification to the associated test cases and a modification of the requirements related to module N.

From the ISO 9001 point view, support of traceability at the project level implies support of software maintainability, because project and maintenance teams will easily understand the relationships and dependencies between the project components and artifacts, and they will have the opportunity to more effectively modify the software system based on updated customer requirements.

6.2.2 Cost management

The change request must first be analyzed and translated into software terms, a process known as *impact analysis*. It is performed after a change request enters the software configuration management process. The objectives of impact analysis are Abran, Moore *et al.* (2004):

- Determination of the scope of the change, in order to plan and implement work;
- Development of accurate estimates of the resources needed to perform the work;
- Analysis of the costs/benefits of the requested change;

- Communication to others of the complexity of the change.

A quality management system requires project managers to perform an impact analysis when a change is requested by the customer. The impact analysis statement will help the development team estimate the budget needed to implement the change request before beginning the change process. The statement will be analyzed by both the project manager and the customer. According to Abran, Moore *et al.* (2004), the software change request is impacted by many factors, such as:

- Application type;
- Novelty of the software;
- Software maintenance staff availability;
- Hardware characteristics;
- Quality of the software design, construction, the documentation, and testing.

Abran, Moore *et al.* (2004) also point out that the software development team should have knowledge of the structure and content of the software system before they begin implementing the requested change. They gain this knowledge by identifying all the systems and software products affected by a software change request, and estimating the resources needed to accomplish the change. This initial knowledge will be enhanced by the availability of traceability mechanisms that will enable developers and software managers to better estimate the cost of changing the content of the system. It will also make it easier to determine the risk associated with implementing the change.

6.2.3 Process improvements

Organizations are complex systems with processes that run concurrently and interact. Improving those processes requires discipline on the part of organizations and a defined reference model to systematically consider their process and project management strategies, as shown in Table 6.1.

The focus of ISO 9001:2008 is on process quality improvement, and a set of requirements and guidelines (in ISO 90003) is defined to help organizations set up their improvement program goals in alignment with their business objectives. Table 1 set out the improvement areas in ISO 9001 at both the process and project levels, and their corresponding CMMI key process areas (KPs).

Table 6.1 ISO 9001 obligations and CMMI KPs corresponding to process and project improvement areas

ISO 9001 and ISO 90003 obligations at the process and project levels	
Organizational process planning Defined team responsibilities, authority, and communication procedures Project resource management Product realization planning Production and service provision Process control and monitoring Project measurement and data analysis for improvement purposes	
CMMI Process management KPs	CMMI Project management KPs
<ul style="list-style-type: none"> •Organizational Process Focus •Organizational Process Definition •Organizational Training •Organizational Process Performance •Organizational Innovation and Deployment 	<ul style="list-style-type: none"> •Project Planning •Project Monitoring and Control •Supplier Agreement Management •Integrated Project Management •Risk Management •Integrated Teaming •Integrated Supplier Management •Quantitative Project Management

In terms of the relationships between software process improvement and traceability techniques, the SWEBOK Guide Abran, Moore *et al.* (2004) points out that the tools and techniques intended to manage the tracking of software documentation and that of software releases can also contribute to improving software process. Briefly stated, traceability for process improvement can:

- Positively impact the communication procedures shared by the process improvement team members, and improve the availability of the software project status throughout all the development phases.
- Facilitate tracking of the sources and causes of defects arising during the software process life cycle, and help address them in a timely manner.
- Help to quickly determine the requirements affected by potential changes to the source code and to any associated test cases.

6.3. Design process

In this section, we present our design for an audit model for software process traceability, focusing on ISO 9001 and the agile software processes. The design process for this model is based on the work of Lopez (2000): ‘An Evaluation Theory Perspective of the Architecture Tradeoff Analysis Method – ATAM’. The use of evaluation theory in the domain of software engineering has been investigated by Lopez (2003) and Zarour (2009), with a view to helping software engineering researchers develop their evaluation criteria, procedures, and conclusions. Those concepts are used in the research reported here for developing our auditing model for ISO 9001 traceability requirements.

6.3.1 Evaluation fundamentals

To design an evaluation procedure, the researcher should consider the components proposed in Lopez (2000) and presented in Figure 6.1. We use these components to design an audit

model to evaluate ISO 9001 traceability and to select a case study that demonstrates the applicability of our audit model – see Figure 6.1.

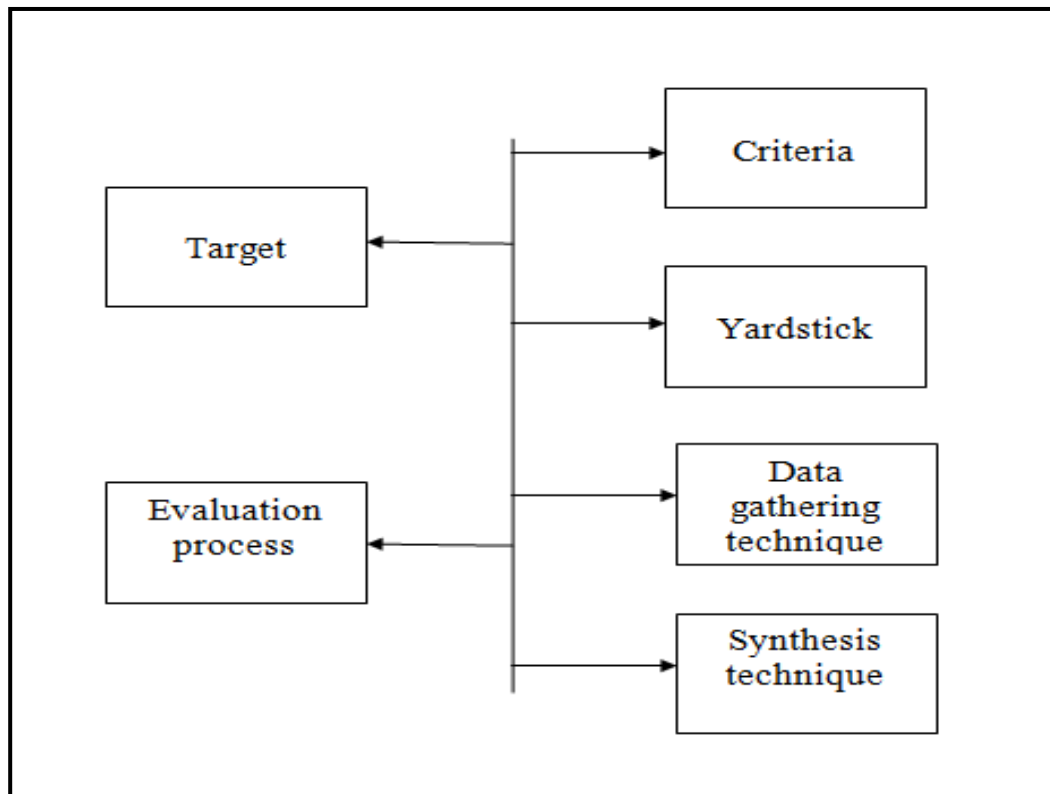


Figure 6.1 Components of an evaluation procedure - Lopez (2000)

The components of an evaluation procedure are highly interrelated with the target, and the delimitation of the target is the first evaluation component that could impact the selection of the evaluation method. Lopez (2000) has classified the evaluation methods into objective-oriented evaluation, management-oriented evaluation, consumer-oriented evaluation, expertise-oriented evaluation, adversary-oriented evaluation, and participant-oriented evaluation.

The design of our audit model considers the steps of an evaluation procedure as described by Lopez (2000):

- Target: the object under evaluation;
- Criteria: the characteristics of the target that are to be evaluated;
- Yardstick: the ideal target against which the real target is to be compared;
- Data gathering techniques: the techniques needed to assess each criterion under analysis;
- Synthesis techniques: the techniques used to organize and synthesize the information obtained with the assessment techniques, the results of which are compared to the yardstick.
- Evaluation process: A series of activities and tasks by means of which an evaluation is performed.

For our purposes here, the design of an audit model can be considered as a type of hybrid approach that combines the principles of management-oriented evaluation Lopez (2000) and adversary-oriented evaluation Lopez (2000), because it is aimed at providing useful information to aid in decision making and at providing a balanced examination of all sides of controversial issues.

Once the target is known and delimited, its characteristics must be identified for evaluation purposes Lopez (2000), Zarour (2009). All the characteristics and their ideal values, which indicate the nature of the target under ideal conditions, make up what is known as the yardstick or standard.

Data about the real target should be obtained using particular data gathering techniques, and assigning a value (data, information set, numerical, etc.) to each criterion. The data, once collected, are organized into an appropriate structure and compared against the yardstick by applying synthesis techniques. This comparison yields the results of the evaluation. Finally, all the above components are linked through the evaluation process Lopez (2000). Figure 6.2 presents the main process for designing an audit model for ISO 9001 traceability requirements based on the evaluation described in Lopez (2000).

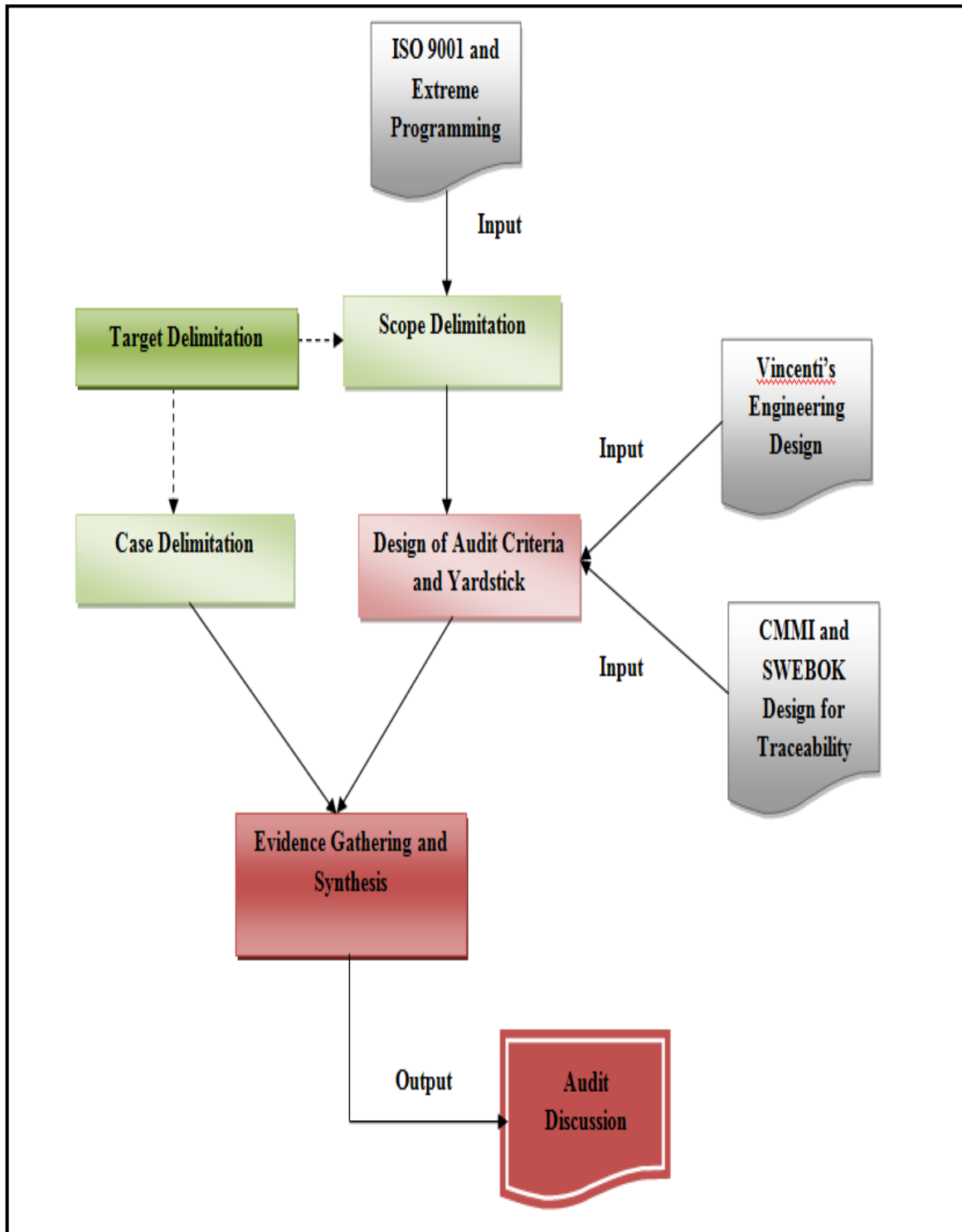


Figure 6.2 Design process for the audit model

6.4 Design of the auditing model

6.4.1 Scope delimitation

For agile software processes (e.g. XP), implementing a traceability technique can help software developers and project managers in tracking the status of the software project and responding efficiently to change requests. The objective of this chapter is to design an auditing model for traceability requirements in agile-XP using evaluation theory. ISO 9001 is the main target standard for deriving the auditing model. The process for designing this auditing model takes as its inputs the guidelines of CMMI and the SWEBOK, as well as Vincenti's engineering design concepts for identifying audit criteria.

The aim of the traceability auditing model is to help ISO 9001 software auditors to audit the agile software processes for traceability requirements for agile-XP. It can also be useful for auditing traditional software processes.

6.4.2 Design of the audit criteria and yardsticks

As stated by Lopez (2000), criteria can be elicited either using a mandatory standard that implicitly contains the criteria to be applied in the evaluation, or, if no such standard has been defined, the auditors should refer to any relevant study of targets, relevant standards, or ideals that might be relevant to the target in question. In our research work here, the mandatory standard is ISO 9001.

The development of an audit model for agile process traceability could not be achieved without support from other relevant software engineering standards, such as CMMI and the engineering design concepts in Vincenti (1990). The structure of the proposed auditing model is presented in Figure 6.3.

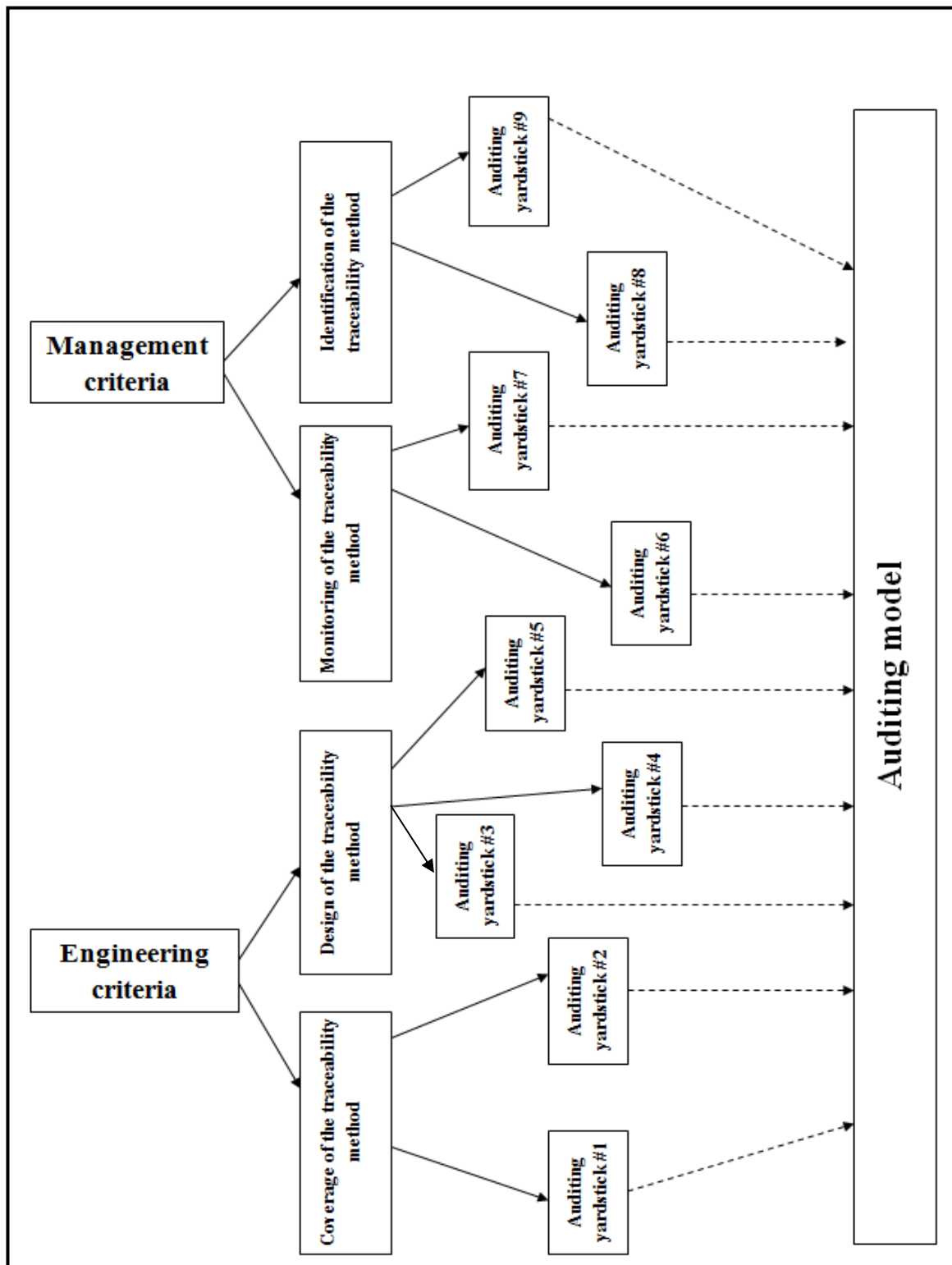


Figure 6.3 The structure of the proposed auditing model

6.4.2.1 Engineering criteria

The list of the audit criteria presented next is based on the concepts of theoretical tools and the operational principles of engineering in Vincenti (1990), Meridji (2010), Zarour (2009).

(A) Design of the traceability method for agile

The main objective of an agile software traceability method is to provide the software developers and project managers with a tool that supports their development tasks. Vincenti's classifications of theoretical engineering tools have enabled us to see what kinds of engineering tools have been used in the design of traceability methods. In Vincenti (1990), these tools are used by engineers to help them with the design process. They include intellectual concepts for thinking about designs, as well as mathematical methods, theories, and formulas, which can be simple or complex, for performing design calculations.

The following are the audit yardsticks identified for each criterion:

Yardstick #1:

Intellectual concepts, which represent the design ideas people have in mind, are expressed in natural language. These concepts are subject to the qualitative reasoning of engineers, before quantitative analysis and design calculations are performed.

Yardstick #2:

Mathematical models, which are useful for quantitative analysis and design, can be either simple or complex. This scientific knowledge must be reformulated to make it applicable to providing engineering knowledge about the design.

(B) Coverage of the traceability method

The set of operational principles underlying an engineering design is classified as a fundamental design concept in Vincenti (1990). These principles define the essential fundamental concept of a device (in this context, a traceability method) and provide a high-

level description of the design objectives, either of the whole design or of each design component. Thus, designers provide either a complete engineering design for the problem in the domain, or a design component that partially addresses the problem in the domain based on the objectives of the operational principles.

The following are the audit yardsticks for this criterion:

Yardstick #3:

Full operational principles: The engineering design of a traceability method considers different life cycle phases, such as requirement specifications, architecture, detailed design, source code, and testing phases.

Yardstick #4:

Partial operational principles: The engineering design of the traceability method focuses on the relationships between entities developed in the same phase of the process life cycle; for example, the artifacts produced during the requirements phase (e.g. the user stories in XP).

6.4.2.2 Management Criteria

In both CMMI and the SWEBOK Guide, traceability management activities are described as a part of the configuration management process area. The SWEBOK Guide describes configuration management as a software engineering knowledge area focused on systematically controlling changes to the configuration, and on maintaining the integrity and traceability of the configuration throughout the system life cycle. The viewpoint of a configuration management system in the SWEBOK Guide is not limited to a software product, but rather covers the functional and/or physical characteristics of hardware, firmware, or software.

CMMI describes configuration management as a supporting process at maturity level 2, which focuses on identification, control, status reporting, and auditing for the traceability items. These items are intended to describe any artifact produced during the software life

cycle, such as requirements specifications, architectural design, source code, test cases, and so on.

The audit criteria presented next are based on the concepts of configuration management described in the SWEBOK Guide and CMMI.

(A) identification of the traceability method

In the SWEBOK Guide, identification of a software traceability item is considered a fundamental step in the construction of a software system that can be controlled and traced during the software process life cycle. At the same time, both the SWEBOK Guide and CMMI stress the importance of assigning unique identifiers to traceability items and developing a strategy for labeling software items and describing their relationships.

The following are the audit yardsticks for this criterion:

Yardstick #5:

Traceability item identification: The traceability method should consider the related traceability identification activities, which include mechanisms for identifying and labeling the traceability items and/or establishing identification schemes that automatically assign unique identifiers to each traceability item.

Yardstick #6:

Traceability item relationships: The proposed schemas for the identification of the relationships and dependencies between the traceability items are considered within a specific development phase or within the entire software life cycle.

Yardstick #7:

Traceability role identification: The traceability method assigns privileges to the software project stakeholders to access or modify the software items in the project baseline or to monitor the status of the software project according to their role in the project. The aim is to comply with the best practices for building a traceability management system in CMMI, and identifying the owner responsible for each traceability item is one of those practices.

(B) Monitoring of the traceability method

Status monitoring and updating of the software project is a requirement for designing a software life cycle traceability mechanism. As discussed in section 6.2, it helps software developers and project managers determine the status of the software project and gauge the impact of changes to the cost, resources, and duration of the project.

The following are the audit yardsticks for this criterion:

Yardstick #8:

Traceability documentation: The information produced during the software life cycle to support the traceability method is reported. The documentation in this case is different from that produced during the software process life cycle, such as software requirements or test cases.

The traceability method produces the required documentation, which covers the entire software life cycle and provides project stakeholders with useful information regarding project status. This information can take the form of ad hoc queries to answer specific questions, or the periodic production of design reports. Examples of such documentation are traceability logs, the history of traceability items, and the relationship of traceability items, and so on.

Yardstick #9:

Documentation access: Traceability documentation and items should be stored in repositories in such a way that software stakeholders are able to access and retrieve them at any stage of the development process. The storage and retrieval mechanisms are evaluated, and the right of access that has been granted based on the role of the traceability stakeholders to assess them is monitored.

6.5 Summary

This chapter has proposed an auditing model for ISO 9001 traceability requirements for agile software processes, in particular for XP. This model can help software organizations in their effort to achieve ISO 9001 certification and help software auditors to extract auditing evidence that demonstrates the ability of a software organization to implement the ISO 9001 traceability requirements. The design methodology for the proposed auditing model is based on evaluation theory. The model consists of two major categories of auditing criteria: engineering criteria and management criteria. Each auditing criterion consists of several auditing yardsticks, which focus on the evidence that can be extracted to demonstrate process conformity to the ISO 9001 traceability requirements.

CHAPTER 7

EXTENDING THE AUDITING MODEL BY COVERING THE ISO 9001 MEASUREMENT REQUIREMENTS

7.1 Introduction

The measurement requirements of ISO 9001 are sparsely described by the standard. Many sections and sub sections of ISO 9001 highlights the importance of measurement and analysis: for example, section 7 “Product realization”, mentions the importance of measurement and analysis during the design and development of new products. Section 8 ‘Measurement, analysis and improvement’ is the main section where the ISO 9001 describes the requirements of measurement for the purpose of the development of a quality management system. This section impacts many other ISO 9001 sections such as section 7 “Product realization”, section 5.4.2 “Quality management system planning”, section 4.1 "General requirements", etc.

This chapter intends to extend the auditing model that has been proposed in chapter 6. First, this chapter analyzes the ISO 9001 requirements of measurement and introduces two types of relation (i.e. implicit relation and explicit relation) to highlight the impact of ISO 9001 measurement requirements on the development activities of a software system. Second, this chapter proposes a set of auditing criteria and auditing yardsticks that can help the XP development team to comply with ISO 9001 measurement requirements. The proposed auditing criteria and auditing yardsticks are also indented to help the IS-auditors to assure that the XP activities had been implemented in conformance to ISO 9001 measurement requirements.

This chapter is organized as follows:

Section 7.2 presents an analysis of ISO 9001 measurement requirements.

Section 7.3 highlights the main finding of the analysis outcome.

Section 7.4 provides description of the current agile measurement techniques.

Section 7.5 describes the design process of the extension for auditing model proposed in chapter 6.

Finally a summary is presented in section 7.6.

7.2 Analysis of measurement requirements in ISO 9001

The requirements of measurement in ISO 9001:2008 cover multiple activities of the quality management system such as customer satisfaction, process quality attributes and product quality attributes. In ISO 9001:2008 these requirements are presented in four different subsections:

ISO 9001-Section 8.2.1: Customer satisfaction.

ISO 9001-Section 8.2.2: Internal auditing.

ISO 9001-Section 8.2.3: Measurement of processes.

ISO 9001-Section 8.2.4: Measurement of products.

7.2.1 Analysis of section 8.2.1: customer satisfaction

The ISO 9001 highlights the importance of customer satisfaction as one of the measurement factors for the performance of quality management system. Thus the software organizations are required to provide methods and techniques for the gathering and analysis of the customer feedback data. The output of ISO 9001 section 8.2.1 on Customer Satisfaction will become essential to the project managers for continuous improvement, corrective actions, and management review.

ISO 9001 Section 8.2.1 describes the management responsibilities at an abstract level and more elaboration can be found in other ISO 9001 subsections. For example, ISO 9001 section 5.5.2 “Management representative” identifies the role of project management more clearly by requiring the project managers to ensure the promotion of awareness of customer requirements throughout the organization. The importance of customer communication procedures are also highlighted in ISO 9001 section “5.5.3 internal communication” where

the organization is required to ensure that the appropriate communication processes are established between project stakeholders.

Obviously the relations of ISO 9001 section 8.2.1 to the mentioned examples are implicit; the section does not directly refer to any other sections within the standard, but the goal of this section could not be possible without the existence of relations to the other parts of ISO 9001 sections and subsection. Figure 7.1 illustrates the implicit relation between ISO 9001 Section 8.2.1 and other sections and subsections in ISO 9001.

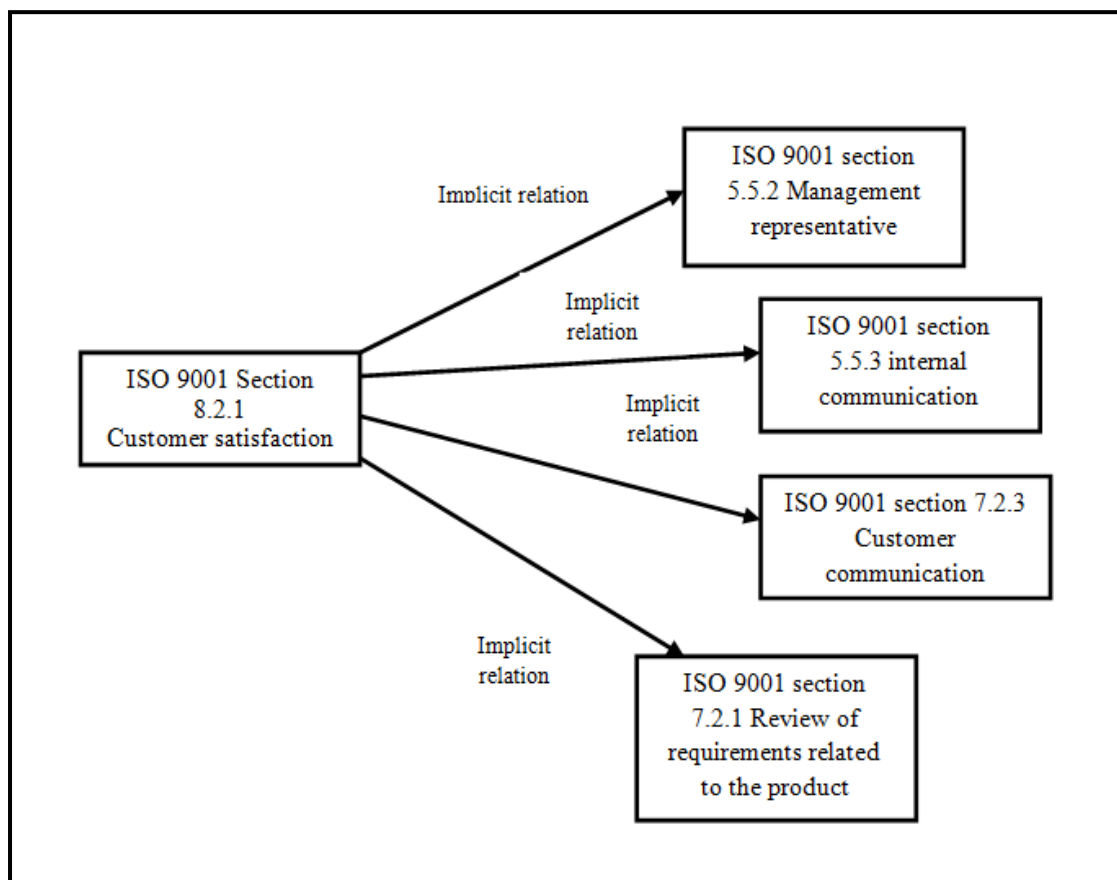


Figure 7.1 Relations of ISO 9001 section 8.2.1 "Customer satisfaction" to other sections in ISO 9001

ISO 90003 elaborates more on useful resources for the project managers to extract the related measurement data, which could provide them with a feedback related to customer satisfaction. This data can be obtained by;

- 1- Analysis of help desk controls related to the customer feedback on the product quality and the product performance.
- 2- Quality-in-use measurement results derived from the direct customer feedback (i.e. customer interview,) and indirect customer feedback (i.e. Surveys that tackle multiple users).
- 3- Number of software releases that needed to be maintained and fixed after the initial delivery.

Regarding quality-in-use measurements, ISO 90003 makes a direct relation to ISO 9126-4 to clarify the type of quality-in-use measurements that can be collected whereas ISO 9126-4 categorizes the quality in-use into four sub-characteristics; effectiveness, productivity, safety and satisfaction.

7.2.2 Analysis of section 8.2.3 of ISO 9001: measurement of processes

Even though ISO 9001 does not clearly define the differences between the activities of process measurement and process monitoring, ISO 9001 emphasizes the importance of monitoring and measurement techniques at the process level during the development of a product.

The SWEBOK Guide defines the term “process measurement” as the collection, analysis and interpretation of quantitative information about the process. Measurement at this level is performed to identify the strengths and weaknesses of processes, and to evaluate processes after they have been implemented and/or changed. On the other hand, the “process monitoring” is meant to cover all activities that steer the implementation of projects by continuous assessment of their possibility to achieve expected goals. The activities that could

be involved during the process monitoring can be summarized next as Abran, Moore *et al.* (2004);

- 1- Continuously assessment of adherence to the various plans at predetermined intervals.
- 2- Analysis of the outputs and completion conditions for each task.
- 3- Evaluation of deliverables in terms of their required characteristics (for example, via verification, reviews and audits).
- 4- Effort expenditure, schedule adherence and costs to date are investigated, and resource usage is examined. The project risk profile is revisited, and adherence to quality requirements is evaluated.

The requirements of ISO 9001 at this stage have explicit and implicit relations to other sections and sub-sections. For example, ISO 9001 section "7.6 Control of monitoring and measuring equipment" mention that the organization shall determine measurement to be used and measuring tools needed to provide evidence of conformity for the process of measurement for product. It is also mention that the purpose of the monitoring and measurement is to demonstrate the "ability of the processes to achieve planned result": this implies implicit relations to ISO 9001 sections "5.4.2 Quality management system planning", "7.1 Planning of product realization" and "7.3.1 Design and development planning". It is also mentioned that when planning results are not achieved, an appropriate corrective action should be involved to ensure the product conformity to the previously stated goals in the initial plan. This implies an explicit relation to ISO 9001 section 8.5.2 "Corrective action" and ISO 9001 section 8.5.3 "Preventive action". The importance of this explicit relation comes from the objectives of ISO 9001 sections 8.5.2 and 8.5.3 in the domain of process measurement and monitoring such that, the organization is required to evaluate the needs of appropriate action to correct and prevent the reoccurrence of nonconformities in the previously stated plans. Figure 7.2 illustrates the implicit relations of ISO 9001 section 8.2.3 to other sections and subsections in ISO 9001.

In ISO 90003, more elaboration is provided on useful resources for the extraction of related measurement data, which could provide the project managers with a feedback concerning the monitoring and measurement of processes. This data can be obtained after the analysis of:

- The planned and actual process activities.
- The planned and actual cost activities.
- The planned quality level and the output measures of selected quality characteristics of the process. It is noticed that ISO 9001 does not specify the acceptable level of quality or the process quality characteristics that could be measured to provide such useful information.

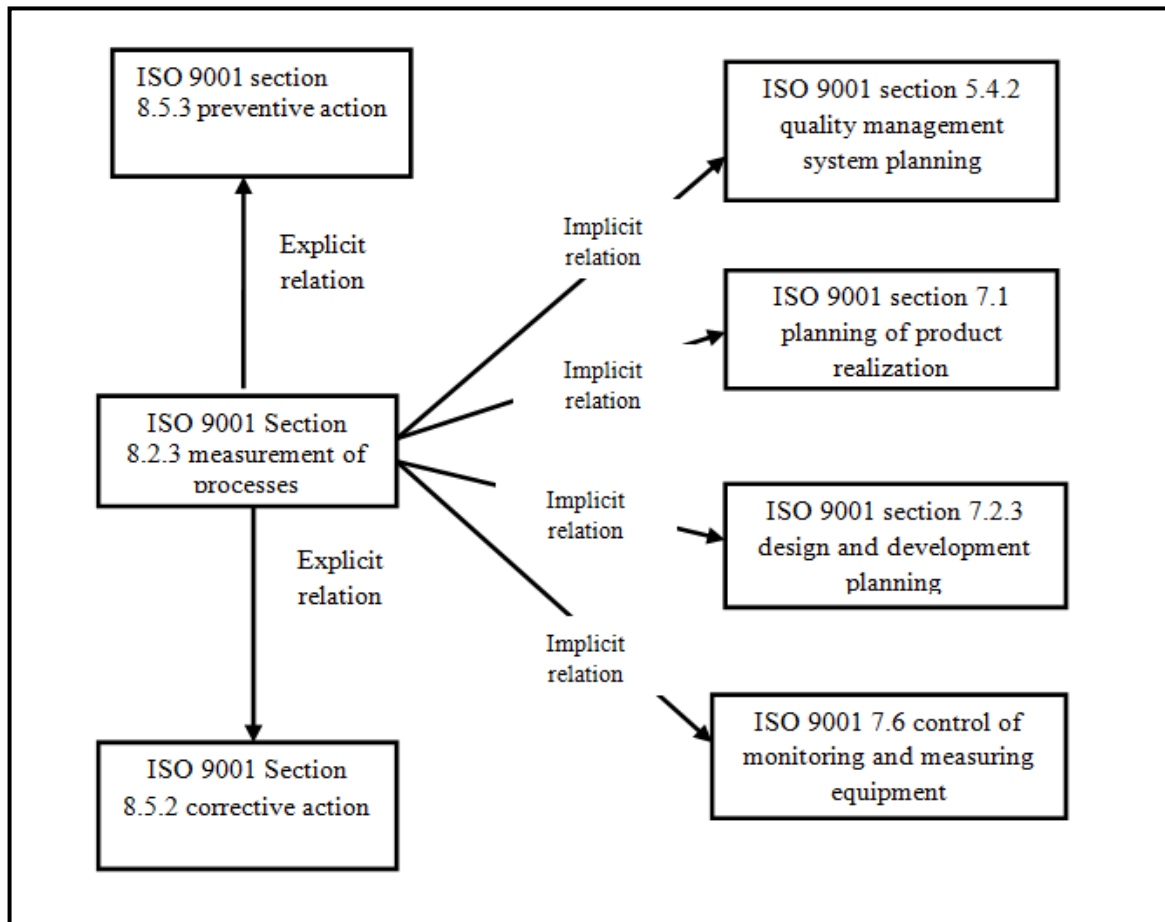


Figure 7.2 Relations of ISO 9001 section 8.2.3 “Measurement of processes” to other sections in ISO 9001

The “corrective action” section of ISO 90003 also states the importance of configuration management to manage changes in the software product. ISO 9001 and ISO 90003 Section 8.5.2 states an explicit relation to section 7.2.8 “Software Problem Resolution Process” of ISO 12207:2008 where the objective is to ensure that all discovered problems are identified, analyzed, managed and controlled to resolution. ISO 12207 Section 7.2.8 was selected by ISO 9001 as a reference model that can support the activities of section 8.5.2 at this level: it has been noted that section 7.2.8 focuses on a set of recommendations that can be implemented to support the corrective action of ISO 9001.

ISO 90003 mention that “corrective action” can be supported through the implementation of configuration management. A detailed activity that could be involved during the configuration management can be found in SWEBOK Abran, Moore *et al.* (2004). The software configuration management (SCM) KPA of SWEBOK is composed of six different sub-areas, which are:

- 1) Management of the SCM process.
- 2) Software configuration identification.
- 3) Software configuration control.
- 4) Configuration status accounting.
- 5) Software configuration auditing.
- 6) Software release management and delivery.

The sub-areas of configuration management are described in details in Chapter 7 of SWEBOK. These sub-areas provide a discipline to identify the configuration of software at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle. As a result, Chapter 7 of SWEBOK is a suggested relation to Section 8.5.2 of ISO 9001. Figure 7.3 illustrates the relation of section 8.5.2 “corrective action” to the ISO standards.

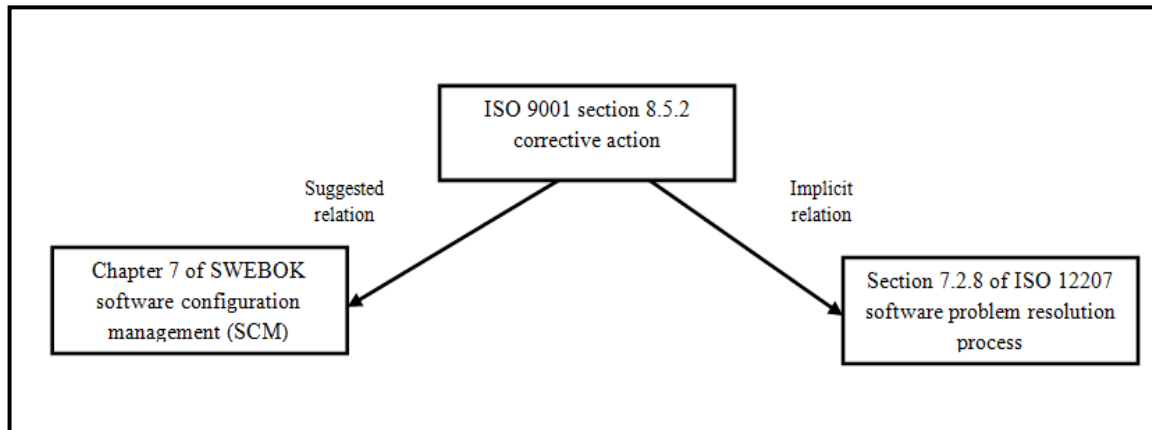


Figure 7.3 Relation of ISO 9001 section 8.5.2 “corrective action” to the ISO standards.

7.2.3 Analysis of ISO 9001 section 8.2.4: measurement of products

ISO 9001 section 8.2.4 requires the organization to “monitor and measure the product characteristics”. These characteristic should be measured to “verify that the product requirements have been met”. The term “verify” has been used clearly in the requirement sentence in this section: it should be not confused with the meaning of other terms such as “verification” and that is used frequently in software engineering to determine specific activities that are performed at the process level of software development.

From the software engineering perspective, the use of the term “verify” in this section can be interpreted as in Al-Qutaish (2007) such that a software measurement has been seen to help the developers in:

- Understanding the software development process better.
- Providing common terminology for key controlling elements of the process.
- Identifying complex software elements.
- Estimating and scheduling better.
- Evaluating the competitive position better.
- Understanding where automation is needed.

- Identifying engineering practices which lead to the highest quality and productivity.
- Making critical decisions earlier in the development process.
- Eliminating fundamental causes of defects.
- Encouraging the use of software engineering techniques.
- Encouraging the definition of long-term software development strategy based upon a measured understanding of current needs and practices.

The requirements of ISO 9001 Section 8.2.4 have an explicit relation with several stages of ISO 9001 section 7 “product realization “, such as sub-sections 7.3.1 “Design and development planning” and “7.1 Planning of product realization”. Figure 7.4 illustrates the relation of section 8.2.4 to other sections and subsections in ISO 9001.

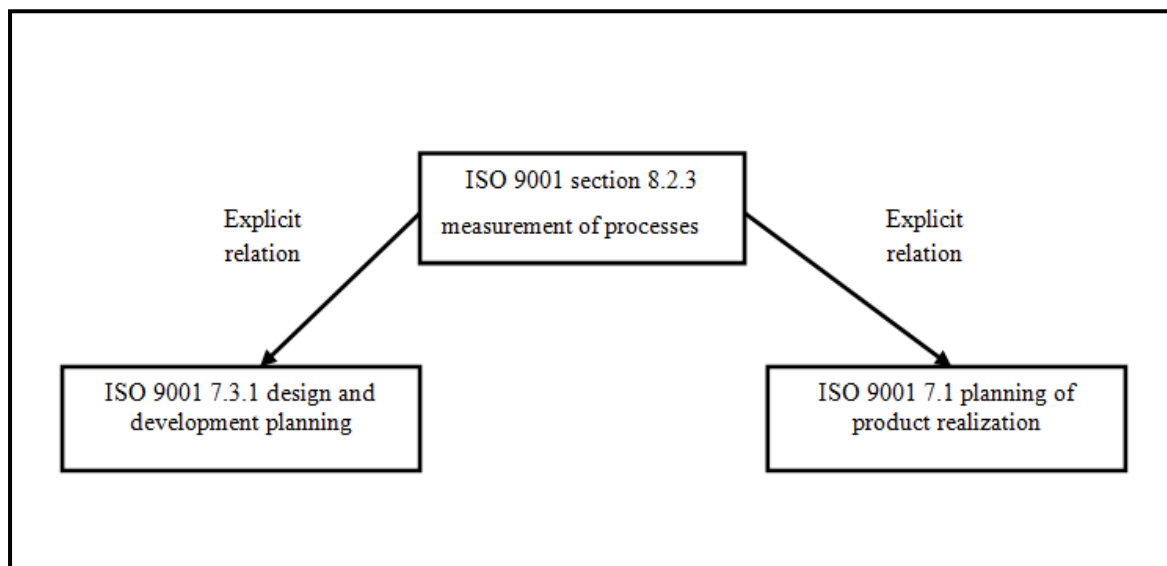


Figure 7.4 Relation of ISO 9001 section 8.2.3 “Measurement of processes” to other sections in ISO 9001

7.2.4 ISO 9001 section 8.2.2: Internal Auditing

The role of internal auditing is defined in section 8.2.2 in the ISO 9001. The standard characterizes the internal auditing process in term of the organization resources, scope and planning activities. The set of requirements in ISO 9001 section 8.2.2 specifies that the

internal auditing should be performed by the organization itself at planned intervals to determine whether the organization processes and activities confirm to the following points: The activities described in ISO 9001 section 7, such as the planning of product realization, determination of requirements related to the product, review of the requirements related to the product, verification and validation has been audited. The internal auditing is conducted at this level to ensure that the organization process has implemented the requirements of ISO 9001 in this section and this is found as an explicit relation between section 8.2.2 and section 7.1 of ISO 9001 - See figure 7.5.

- Section 4.1 "General requirements" of quality management system such that the internal auditing is performed at this level to ensure that the requirements of this section have been carried out as specified i.e. the processes, methods, resources for implementing the quality management system have been performed as described in the plan. ISO 9001 section 8.2.2 implies an implicit relation to ISO 9001 section 4.1 - See figure 7.5.
- Any other quality management system requirements established by the organization in response to the ISO 9001 obligations such as Quality objectives (5.4.1), Quality management system planning (5.4.2) and Quality policy (5.3). See figure 7.5 that clarifies the relations of section 8.2.2 to other sections in ISO 9001 - See figure 7.5.

The main difference between internal and external auditing is their domain of interest. The external auditing focuses on the organization activities that have been implemented in accordance to a specific standard (e.g. ISO 9001). This also may be an important concern for the internal auditors; but the focus is on the conformance of the organization processes to their own plans and policies.

Internal auditors may derive recommendations from auditing results for outlining the process of improvements in the software organization. Internal auditors may also participate with external auditors to achieve their work. This may enhance the accuracy of the audit result as well as reduce the auditing fees. Auditing can be classified into three different types (Paul, Curtiss *et al.* 2009);

- First party audit: First party audit is performed within an organization to assess its strengths and weaknesses against their own policies and strategies and/or against external standards such as ISO 9001. The first party audit is a type of internal audit that is conducted by auditors employed by the organization.
- Second party audit: Second party audit is a type of external auditing which is usually conducted by an external organization on behalf of the organization being audited. Second party audit is more formal than first party audit because it is initiated with a contract which specifies the role of second party auditors.
- Third party audit: The third party audit is considered as a type of external auditing. The key element of the third party audit is the independence from the organization being audited. The result of the third party audit has many outcomes such as organization certification, organization license approved, and/or penalties issued by the third party. For example, organization should be audited by third party agencies to verify their level of adherence to the ISO 9001 requirements.

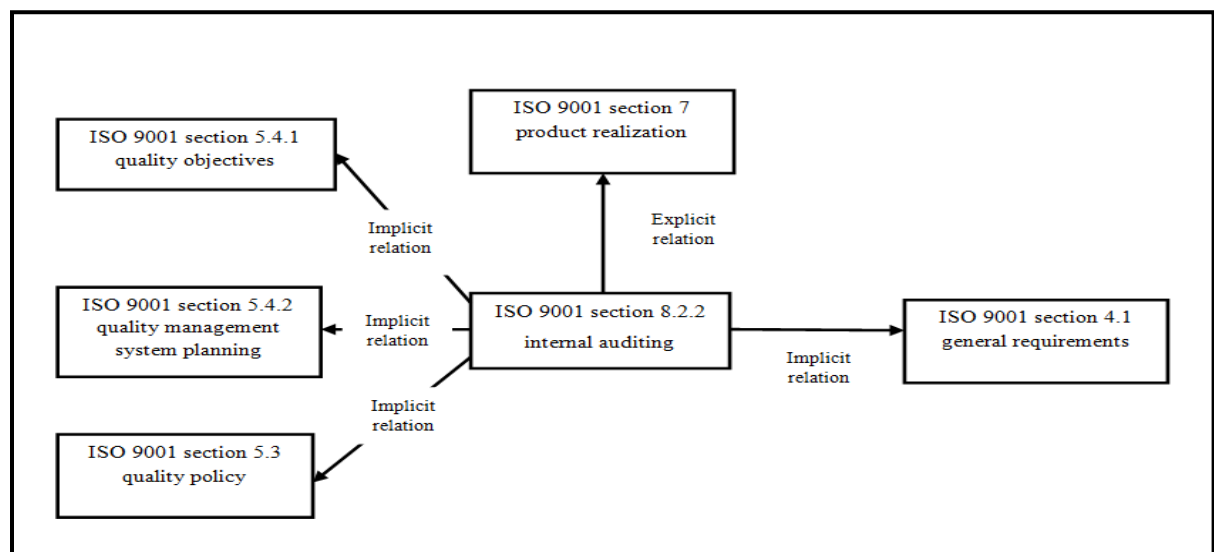


Figure 7.5 Relations of ISO 9001 section 8.2.2 “Internal auditing” to other sections in ISO 9001

7.3 Discussion on ISO 9001 measurement requirements.

- ISO 9001 section 8.2 of ISO 9001 neither specifies nor recommends any measurement techniques to be used by the organization; the standard determines a set of requirements to help the organizations (including software organizations) to develop their own measurement program aligned with an organization's goals and objectives.
- ISO 9001 section 8.2 consists of several implicit and explicit relations to other sections in the standard. These relations highlight the importance of the measurement activities for the organization and specify links between different levels of measurement activities inside the organization. For example, top management may be interested in a specific set of measurements such as customer satisfaction and effort estimation. On the other hand, developers may be interested in obtaining the measures that are related to the development process level such as performance measurement, reliability measurement etc. The identified relations will help for maintaining the measurement activities at different levels aligned with an organization's goals and objectives.
- Neither ISO 9001 section 8.2 nor any of ISO 9001 referred sections suggest the collection of measures at the product or process levels. ISO 90003, on the other hand, suggests measurement of certain attributes at the product level, such as functionality, maintainability, efficiency, portability, usability and reliability. Furthermore, ISO 90003 refers to ISO 9126 which specifies a set of quality characteristics that can be measured at product level.
- ISO 9001 does not specify any requirements for external auditing; however, the requirements of internal auditing have been described at a high level to standardize the activities in industrial organizations. For example, the ISO 9001 internal auditing requirements may cover different auditing activities such as IT auditing, financial auditing and quality auditing.
- The measurement requirements of ISO 9001 are integrated into the system and software development. This can be notable after tracking the existing relationships between the different subsections of ISO 9001.

7.4 Existing agile measurement and estimation techniques

This section discusses from the viewpoint of ISO 9001 several measurement and estimation techniques proposed for agile software processes. This section also discusses the weaknesses of those techniques from an auditing perspective.

- **Planning poker:** This technique has been introduced first by Grenning (2002). Planning poker is used to estimate effort or size of tasks in agile software development and more precisely in XP Cohn (2005). All the project stakeholders can participate in the planning poker such as programmers, testers, designers and analysts. At the start of planning poker, each estimator is given a deck of cards. Each card has written on it one estimate. Each estimator may, for example, be given a deck of cards that reads 0, 1, 2, 3, 5, 8, 13, 20, 40, and 100. The session moderator presents a short description about the user stories which need to be estimated. Then each individual lays a card face down representing their estimate. The selected number by each individual usually represents the User Story Point (USP). USP can be interpreted as an evaluation-estimation of the efforts required to implement the story. Then the estimation process is repeated until a consensus is reached.
- **Velocity:** In an agile software process such as XP and Scrum, the term "Velocity" is used to measure the effort invested to produce software Cohn (2005). Calculation of "Velocity" is based on the historical data obtained from several previous iterations, such that it can be calculated by averaging the estimates delivered of features per iteration. Project managers can specify several measurement units, such as USP, days, ideal days, or hours. "Velocity can be calculated as a weighted historical average favouring recent iterations (as these are most representative of the current rate of progress looking forward) or as a simple average of the most recent two or three iterations" Karlesky and Vander (2008).

Karlesky and Vander (2008) claims that by using velocity, project managers can estimate the amount of work and resources available, lessons are learned from past projects which have completion difficulties and requests for budget extension, delivery date, and

resources. Priorities and corrective actions can be adjusted before the over estimation can appear during the development processes of a software systems.

- **Burn down chart:** Burn down chart is a graphical representation of the amount of work that still needs to be completed before the end of a project, which is usually calculated as the sum of the estimated remaining effort for all tasks defined in each iteration (COSMIC, 2011). The charts are usually showing the work remaining in the project, determining team velocity, and estimating how many iterations it will require to complete the project. The charts are represented by Y-axis that can track the story points, ideal days, hours, etc. X-axis remaining against project iterations or days in each iteration. See figure 7.6 of an example of a Burn down chart. The X-axis represents the iteration number while the Y-axis represents the story points completed and the total story points in the project.

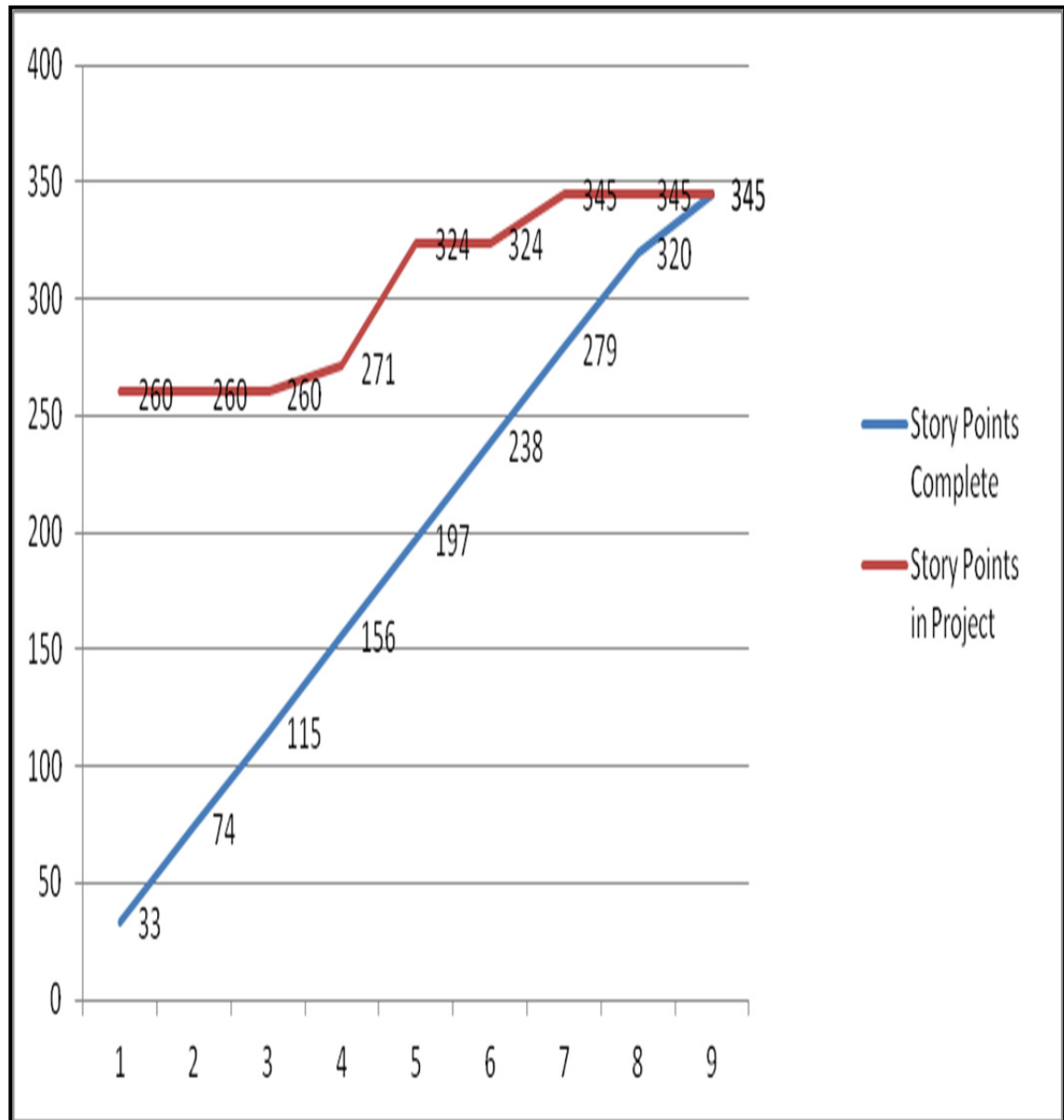


Figure 7.6 An example of a burn down chart

The drawbacks of the above agile estimation techniques can be seen from the measurement and auditing perspectives in Table 7.1.

Table 7.1 Agile estimation techniques and their weaknesses from measurement and auditing perspectives

Agile techniques	Measurement perspectives	ISO 9001 Auditing perspectives
Planning poker	The USP value cannot be considered as a reliable measure of a User Story size simply because in practice it is a value representing the relative effort to develop the User Story, not a measure of its size (COSMIC, 2011). USP is not an objective product size unit, does not comply with basic metrology concepts and cannot be defined as a standard software sizing measure (COSMIC, 2011).	USP estimated by the planning poker is not based on defined or documented estimation criteria. Each estimator will assign the estimation value that reflects his point view on how to predict the efforts needed for developing the user stories. This practice will provide fewer evidences for ISO 9001 to assure that the estimation has been preformed based on an engineering approach.
Velocity	Cannot be used as a benchmark value because its USP component is not a standard measure, so velocity cannot be considered as a standard measure (COSMIC, 2011).	The calculation of project velocity is based on historical data which can provide an estimate for new set of backlog User Stories. This historical data is also a valuable resource for ISO 9001 auditors. However, it has been noted in (COSMIC, 2011) that not all teams collect or have access to historical data, and they often rely on an estimate of their upcoming velocity to perform their preliminary project estimate.
Burn down chart	As the agile project accepts the changes of requirements frequently during the development, the changes should be updated every time new user stories are added or removed from the product backlog.	Multiple versions of the Burn down chart may be required to assure that the project team has conducted their estimation to reflect the changes of project requirements along the development process.

7.5 Design process

In the context of this section, the design process for the auditing model will be clarified using the concepts of engineering design process in Vincenti (1990), Meridji (2010).

7.5.1 Engineering design process

According to Vincenti, the engineering “design concept” denotes both the content of a set of plans and the processes by which those plans are produced" Vincenti (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 engineering device; these plans are produced, and finally, the release of these plans for production. Devices are defined as single, relatively compact entities, such as airplanes, electric generators, turret lathes, and so forth. In the context of this report, devices refer to the auditing model and its components such as the auditing criteria and auditing yardsticks.

Vincenti also mentions that design is a multilevel and hierarchical process. The designer starts by taking the problem as input. The design hierarchy starts from the project definition level, located at the upper level of the hierarchy where problems are abstract and unstructured. At level 2, the project is divided into its major components. At level 3, each component is subdivided. At level 4, the subcomponents from level 4 may further be divided into specific design components based on the design needs. At the lower levels, design components are well defined and structured. Finally, the overall design level, the layout and the proportions of the device are set to meet the project definition. The design process is iterative, both up and down and horizontally throughout the hierarchy. Figure 7.7 shows the Vincenti's levels of design that will be used for designing the auditing model.

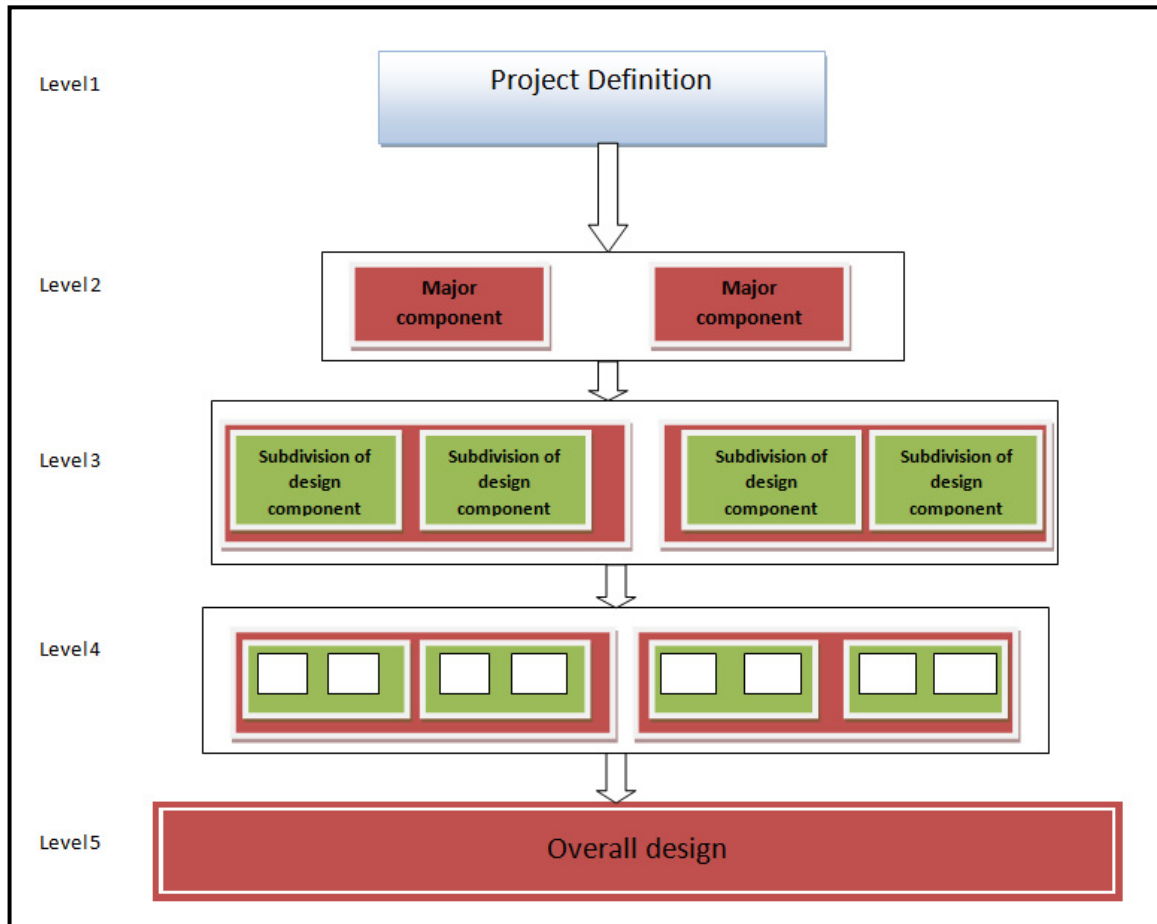


Figure 7.7 Vincenti's levels of engineering design

7.5.2 Design formulation

The Vincenti principles of engineering design have been inspired from the domain of aeronautical engineering. However, Vincenti has mentioned that this classification is not specific to the aeronautical engineering domain and a transformation can be made for design and analysis purposes to any other engineering domain. For example, the work presented in Meridji (2010) proposes some pioneering work in modeling Vincenti's engineering principles, and utilizes Vincenti's engineering domain areas as constituting criteria for investigating software engineering from an engineering perspective. Also, Zarour (2009) presents the use of Vincenti's engineering principles for the development of evaluation methods as to evaluate the software assessment methods from engineering perspective. In the context of this chapter, the hierarchical levels of engineering design - see figure 7.7 - will be

used as a modeling guidelines for designing the main components of the auditing criteria and auditing yardstick, for ISO 9001 measurement requirements in the agile-XP environment.

7.5.2.1 Project definition

The objective of this chapter is to design an auditing model for ISO 9001 measurement requirements, following the principles of engineering design defined by Vincenti. Also, ISO 9001 has been identified as the main target standard for deriving the auditing model components (i.e. auditing criteria and yardsticks). The design process for this auditing model takes as its inputs the guidelines of CMMI, SWEBOK and ISO 15939 for the identification of design process for audit criteria and yardsticks.

The aim of the measurement auditing model is to help the ISO 9001 software auditors audit the agile software processes for measurement requirements.

7.5.2.2 Major Design Component

Defining the major design component is the second level of design based on Vincenti's hierarchy of engineering design. The major design component for this chapter is to elicit the auditing criteria for the ISO 9001 requirements. The auditing criteria is defined as the major area of interest for the software process auditor where it is expected to reveal audit evidences to assure that certain activities have been planned, executed and/or evaluated. It is important at this level of design to determine the main standard (i.e. ISO 9001) and supplementary documents that will be used to complete the auditing criteria of ISO 9001 measurement requirements.

Based on Lopez (2000), criteria elicitation can be made using a mandatory standard that contains implicitly the criteria to be applied. For measurement auditing criteria, the mandatory standard is ISO 9001, more precisely the measurement requirements of ISO 9001. However, after the analysis of Sections 8.2.1, 8.2.3, and 8.2.4 in ISO 9001 it has been noted that deriving the auditing criteria for the ISO 9001 measurement requirements would be

difficult without a support from other related software engineering based models, such as CMMI, SWEBOK and ISO 15939. For this purpose, Lopez (2000) mentions that several criteria elicitation techniques can be used such as:

- 1- Functional analysis: Detailed analysis of the obligatory standard as to gain in depth description for the obligatory standard requirements.
- 2- Needs assessment: A designer may refer to any study of the needs, wants, market preferences, standards, or ideals that might be relevant to the target.
- 3- Complex logical analysis: when the definition needs more clarifications in order to figure out its implications. This is more often the case when the criterion is complex and may need to be decomposed into several yardsticks. The analysis is a complex inferential process starting from data and definitions.

Figure 7.8 shows the main source documents for criteria elicitation and techniques to define the auditing criteria for ISO 9001 measurement requirements.

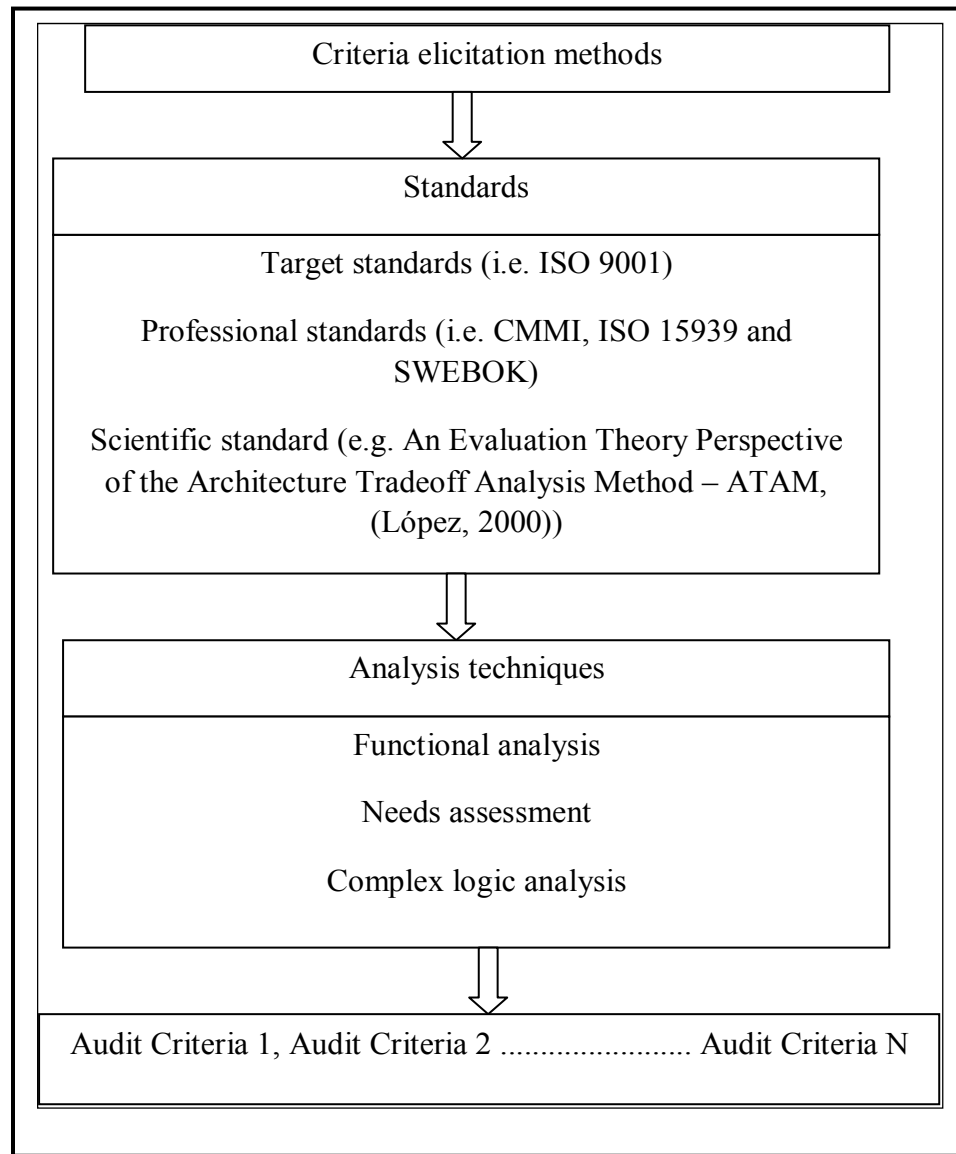


Figure 7.8 The elicitation of auditing audit criteria

Next is the description of auditing criteria/creation and their related yardstick - see figure 7.9.

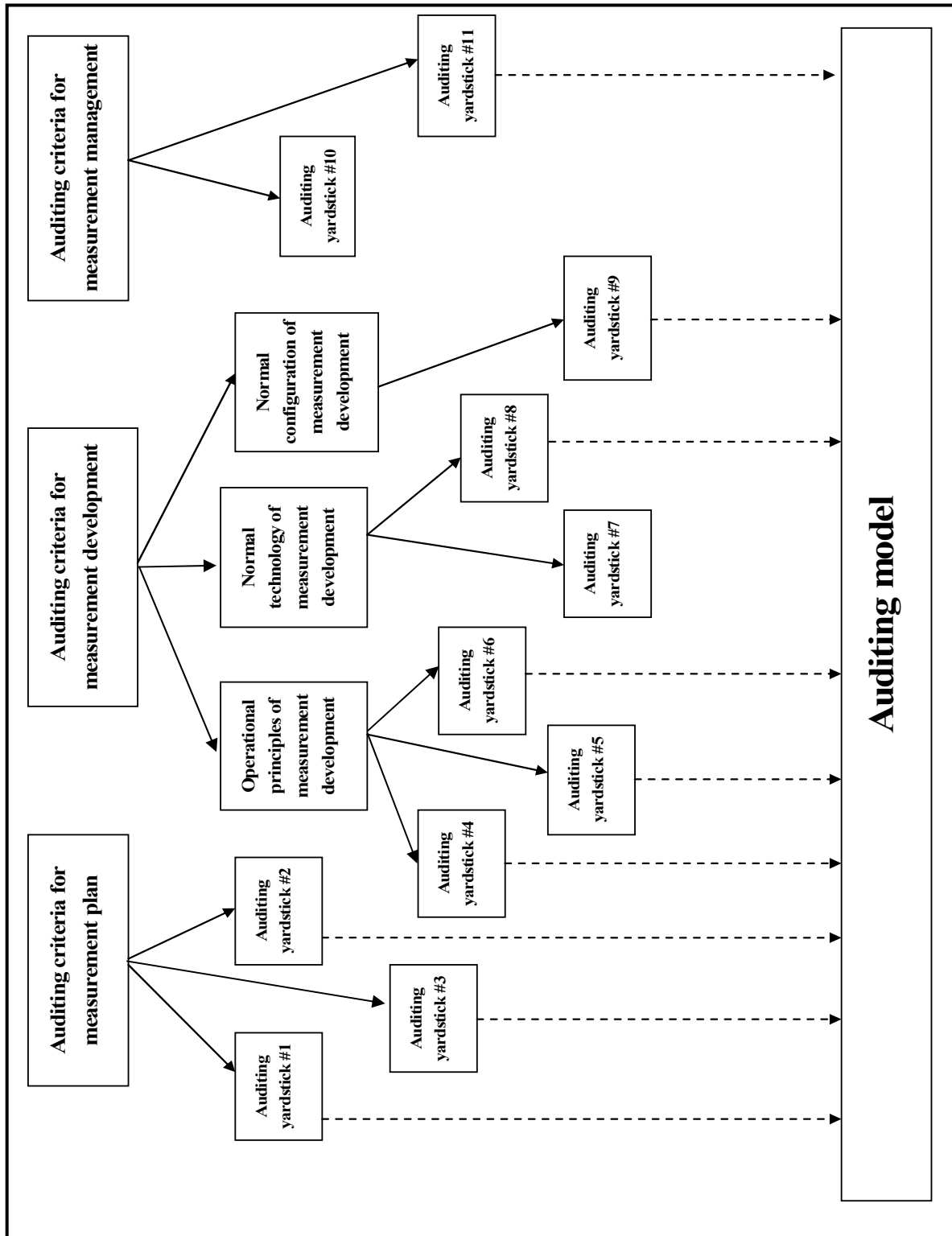


Figure 7.9 The structure of the measurement auditing model

7.5.3 Subdivision of design component

The previous section has mainly focused on identifying the main components of the design based on Vincenti hierarchy of engineering design, and by defining the main standards that will be used for the elicitation for the auditing criteria. This section focuses on describing those auditing criteria and their related auditing yardstick. The auditing yardstick will be defined in this context as a specific area of interest for the IS auditor to extract detailed information concerning the auditing criteria. Each auditing criterion is composed of 1 to N yardsticks.

Based on Lopez (2000), the following principles will be applied for the process of designing the auditing yardsticks:

- The yardstick should be developed from the standards described in figure 7.8. The general structure of the auditing criterion and auditing yardstick should be obtained using the analysis techniques described in figure 7.8.
- The yardstick must contain specifications for specific defined criterion /criteria.
- For each criterion, whenever possible, the yardstick must define the specifications structured as pairs [criterion, yardstick/information].

The ISO 15939:2007 has been selected to derive the main measurement auditing criteria - see figure 7.10. The basis of this selection is the following:

- **Scope alignment:** from the analysis of ISO 9001 measurement requirements, this standard is intended to guide the measurement program inside the organization rather than focusing on a specific measurement technique. ISO 15939 details different phases during the measurement process, such as: Establishing & sustaining measurement commitment, Plan the measurement process, Performing the measurement process and Measurement evaluation.

- **Field of application:** ISO 15939 mentions several circumstances in which this standard is useful for the software organizations. For example, the standard can be used by “suppliers to implement a measurement process to address specific project or the organization information requirements”. Beside, the standard is useful to be used by “acquirer (or third-party agents)” for evaluating conformance of the supplier’s measurement process to specific measurement requirements. This standard will be used in this context for deriving the main measurement auditing criteria, and to define the process and product measurement information that are useful for the software organization that implement the agile process (i.e. XP), in order to achieve the ISO 9001 certification. Those auditing criteria will be useful to the IS auditor to extract the audit evidences that assure the process conformance. Agile software organizations can also implement their measurement process to address specific technical and/or project measurement techniques related to the auditing criteria.
- **Standard reputation:** The purpose of this standard is claimed to "define a measurement process applicable to system and software engineering and management disciplines". The process is described through a model that defines the activities of the measurement process required to adequately specify what measurement information are required, how measurement and analysis results are to be applied, and how to validate the analysis results. “The measurement process is flexible, tailorable, and adaptable to the needs of different users". As a result, the standard has been used in several areas of software engineering and some examples are described in table 7.2. Moreover, a publication of International Standards, such as ISO 15939, requires approval by at “least 75 % of the national bodies casting a vote”.

However, a limitation noted of ISO 15939 is that “the measurement process should be appropriately integrated with the organizational quality system. Not all aspects of internal audits and non-compliance reporting are covered explicitly in this International Standard, as they are assumed to be in the domain of the quality system”. As a result, some other

standards beside ISO 15939 have been used in designing the auditing criteria and yardsticks, such as SWEBOK (ISO 19759) and CMMI models.

Table 7.2 Examples of publications work based on ISO 15939

Work authors	Work Title	Role of ISO 15939 in the work	Work type and year
Alain Abran et, al.	Analysis of the ISO 9126 on Software Product Quality Evaluation from the Metrology and ISO 15939 Perspectives	The concept and terminologies described by ISO 15939 is used as a basis for analyzing ISO 9126 on area of product quality evaluation	WSEAS Transactions on Computers, 2006
Luc Bégnoche, et, al.	A Measurement Approach Integrating ISO 15939, CMMI and the ISBSG	The ISO 15939 has been used as a supportive tool to design an approach that supports software engineering measurement program	Software Measurement European Forum, 2007
Alain Abran et, al.	An Information Model for Software Quality Measurement with ISO Standards	The measurement information model of ISO 15939 on software measurement process has been used to assess the maturity for the concepts of measurement primitives and quality measures, and highlights some of their weaknesses to recommend future improvement for those concepts.	International Conference on Software Development, 2005

Table 7.2 Examples of publications work based on ISO 15939 (Continued)

Work authors	Work Title	Role of ISO 15939 in the work	Work type and year
Dias Belchior et, al.	Measurement Process: A Mapping Among CMMI-SW, ISO 15939, IEEE Std 1061, Six Sigma and PSM.	Presents a mapping of several measurement processes: CMMI-SW, ISO 15939, IEEE Std 1061, and Six Sigma, which aim to investigate the similarities and the gaps among these approaches along with the focus on software projects.	International Conference on Service Systems and Service Management, 2006
Verbo, E	A Methodology Based on ISO 15939 to Elaborate Data Quality Measurement Plans	This work presents a methodology called (MEPLAMECAL) which aims to develop plans for quality measurement. The development of MEPLAMECA is mainly based on ISO 15939.	IEEE Latin America Transactions, 2009

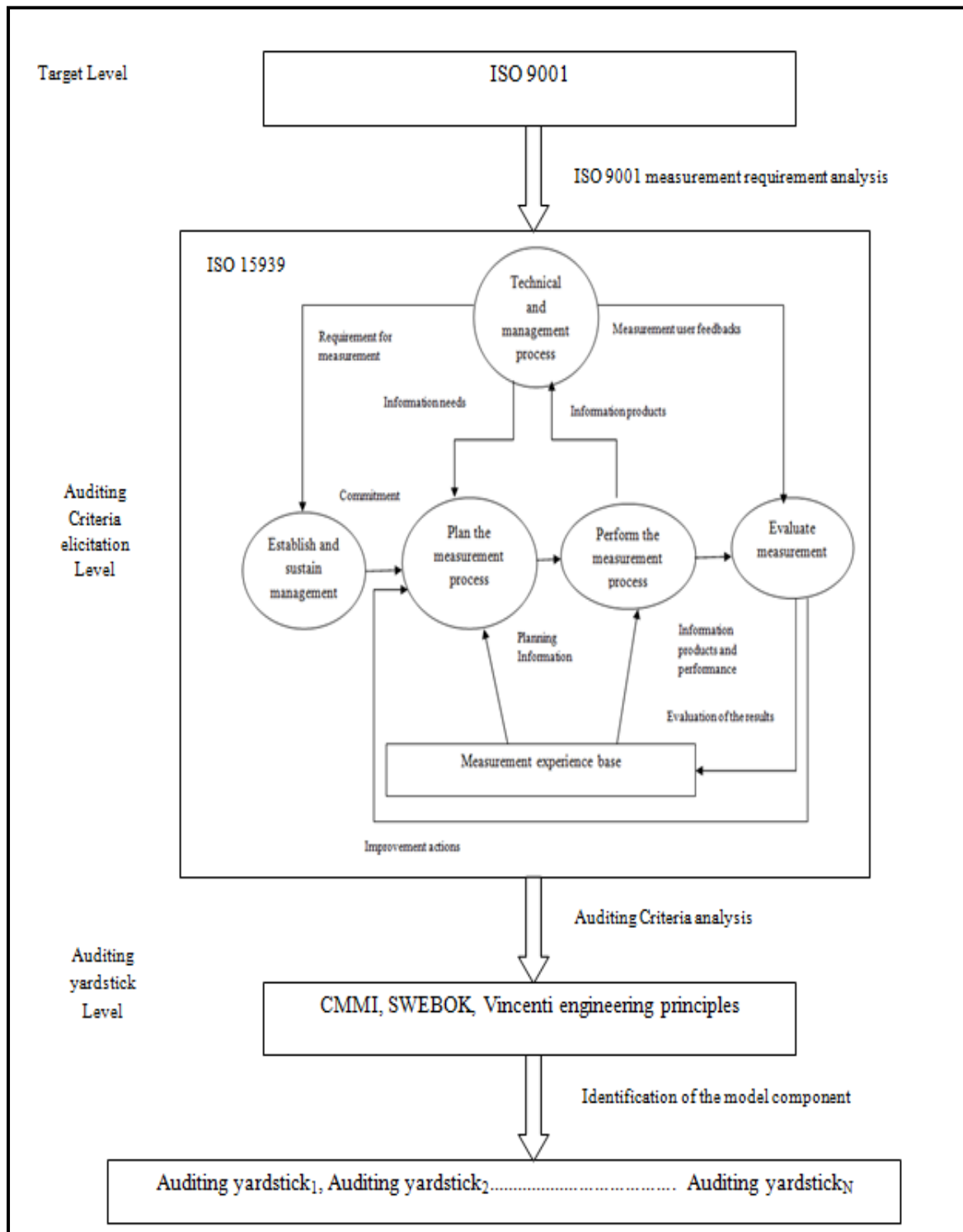


Figure 7.10 Auditing model design levels

The next sections present the description of the auditing criteria identification and their related yardstick.

7.6.1 A: Auditing criteria for measurement plans

In the context of measurement process, the SWEBOK and ISO 15939 stress the importance of planning at the beginning of the measurement process. The same is found in CMMI model where the planning for measurement is classified as a support process at maturity level 2 which improves the corrective and preventive actions. The mentioned standards highlight the importance of early planning for software process, as to specify the objectives of the measurement program and to align the identified information needs within the organization objectives, i.e. improving quality, reduce cost, enhance the infrastructure security etc.

According to CMMI the software process models, such as agile-XP, can take advantages of measurement planning to:

- Track actual progress and performance against established plans and objectives.
- Establish for ongoing program for continuous process improvement.
- Manage budget for development project more efficiently.
- Benchmark the current process, tools and techniques.

Auditing criterion at this phase focuses on investigation of evidences to assure that activities of planning have been carried during the measurement process. The key point of this auditing criterion is to provide a link between the development team of agile-XP software processes and IS auditors, to understand what evidences they need at this phase, and how evidences can provide them with facts.

The following are the audit yardsticks for this criterion:

Yardstick #1:

Identification of the measurement context: The measurement plan should identify the context of the measurement process. This can be accomplished by defining the focus of measurement process to the intended goal of organization improvement. For example, the SWEBOK mentions several improvement goals at the organizational level, such as the organizational processes, application domains, technology, and organizational interfaces. The SEWBOK mentions that “it is important to make this context explicit and to articulate the assumptions that it embodies and the constraints that it imposes”.

For agile-XP process which imposes lightweight documentation methodology, the measurement context can be identified by briefly answering several questions based on goal-question-metric (GQM) style. (We do not claim this is a complete set of questions, but it provides an example of questions that can reveal important information for this yardstick). Table 7.3 is designed based on Statz (2005).

Table 7.3 Questions based context analysis for auditing evidences extraction

Measurement Context	Impact of the measurement program to the organization improvement
Financial	<p>How much will it cost for designing the measurement program?</p> <p>What financial benefits will be achieved by implementing the measurement program?</p> <p>Which financial overhead will be avoided?</p>
Customer satisfaction	<p>How the measurement program can impact the customer satisfaction?</p> <p>Which area of customer interest will be improved?</p> <p>How the measurement program will reduce the overhead of customer support?</p>
Business process improvement	<p>How can the measurement program reduce the cost toward quality products?</p> <p>How can the measurement program improve the organization productivity?</p> <p>How can the measurement program reduce time to market?</p> <p>How can the measurement program improve the internal business process?</p> <p>How can the measurement program utilize the organization resources?</p>
Training and learning benefits	<p>Will the measurement program enhance the management capability?</p> <p>Will the measurement program enhance the project stakeholder satisfaction?</p> <p>Will the measurement program increase the collaboration and feedbacks between the project stakeholders?</p>

To reduce the documentation for agile software process for this auditing criterion the approach of Alali and Issa (2011) can be useful, which is mainly based on the identification of special patterns from the text and then classify those patterns based on defined categories. Another suggestion that could reduce the documentation is videotaping the development sessions of measurement context discussion.

Yardstick #2:

Role Assignment: This yardstick is focused on identifying the measurement team roles and the responsibilities of each team member. This yardstick has been derived from ISO 15939 that mentions “Individuals shall be assigned responsibility for the measurement process within the organization”. On the other hand, CMMI model mentions that when the organization establishes a measurement program, the measurement plan should include a specific role for each team member, as to ensure that analysis will properly address the identified information needs.

The agile-XP can take advantages of the role assignment proposed by ISO 15939, as described next.

- Measurement user: Individual or organization that uses the information products.
- Measurement analyst: individual or organization that is responsible for the planning, performance, evaluation and improvement of measurement.
- Measurement librarian: individual or organization that is responsible for managing the measurement data stores.

Yardstick #3:

Resources and budget constraints: The plan for the measurement process should be integrated with the main business improvement plan. For example, the CMMI model suggests several activities in business improvement plan such as:

- Identify major milestones: Milestones can be classified as event based or calendar based. If it is calendar based, it will be difficult to change later during the project progress. Those defined milestones should also be reviewed once the measurement plan is defined as to keep the measurement process aligned with the project scope and time constraints.
- Identify schedule assumptions: Assumptions are judgments and forecasting of several development activities’ duration and required resources. With assumptions, little data is available for estimations based on engineering methodologies. CMMI mentions that: "When schedules are initially developed, it is common to make assumptions about the duration of certain activities".

- Identify constraints: The CMMI defines the constraints as factors that limit the flexibility of development and design. For agile-XP, this can include team expertise, customer involvement, project size, project criticality (e.g. safety critical systems), requirement changes etc.
- Identify task dependencies: The dependencies between project tasks and/or services should be defined. CMMI mentions that an ordered sequence of task dependencies for the software project can minimize the project duration. For agile-XP, several dependencies can be defined at this level, such as: logical dependency, data dependency, temporal dependency and resource dependency Qasaimeh and Abran (2011).

7.6.2 B: Auditing criteria for measurement development

This auditing criteria focus is to help the IS Auditors of agile-XP in finding evidences of the development activities for the measurement process. The objective is to design auditing yardsticks that specify the kinds of actions that can be taken during the design as well as auditing the measurement process.

The design process is defined by SWEBOK as “the process of defining the architecture, components, interfaces, and other characteristics of a system or component”. 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" Abran, Moore *et al.* (2004).

For the design of these auditing criteria, several standards and guideline principles will be used to complete the design process, such as Vincenti principles of engineering design, SWEBOK and ISO 15939. The auditing criteria for measurement development will be divided into four criteria, which are mainly based on Vincenti “fundamental design concept”. According to the analysis of Meridji (2010) for Vincenti design principles, the fundamental design concepts are composed of four elements: operational principles, normal configuration,

normal technology and concepts in people's minds. At the beginning of the design process, these concepts exist only in the designer's mind.

The auditing criterion of the measurement process presented next focuses on the principles of these four concepts by designing their associated yardsticks.

- **Operational principles of measurement development**

For the IS Auditor, the operational principles define essential the fundamental concepts of the measurement process. The operational principles must be known by the designers first, and subsequently to the IS Auditors as to find the evidences for this auditing criterion. Generally, as mentioned by Vincenti, the operational principles specify how the different parts of the designed device fulfill special functions in combination with overall operation to achieve the purpose. Vincenti (1990) defined the operational principles as “how the device works”. Next are the auditing yardsticks which are associated with this auditing criterion.

Yardstick #4:

Identification of measures: software measures are classified into base measures and derived measures, which are used to construct the software quality measures Al Qutaish (2007). This auditing yardstick is focused to investigate the classification of measures that have been specified for the agile-XP project. As explained by the CMMI model, data for the base measures are obtained by direct measurement, whereas data for derived measures come from other measures, typically by combining two or more base measures. In other words, the base measures could be calculated based on the attribute itself and without using any other measures, while derived measures could be calculated based only on other measures. Table 7.4 presents a list of base measures and derived measures suggested by the CMMI model. The list can be used by the IS auditor of an agile-XP project.

Table 7.4 Examples of base measures and derived measures in the CMMI model

Examples of base measures	Examples of derived measures
Measures of work product size (e.g., number of pages, number of lines of code)	Earned value
Measures of effort and cost (e.g., number of person hours)	Schedule performance index
Information security measures (e.g., number of system vulnerabilities identified)	Defect density (e.g number of confirmed defects/ software size)
Customer satisfaction survey scores (e.g., survey scores)	Peer review coverage
Process productivity (e.g. number of tasks completed)	Reliability measures (e.g., mean time to failure)
Error time (i.e. time required to correct the code errors)	Information security measures (e.g., percentage of system vulnerabilities mitigated)

Table 7.4 contains some examples of base measures and derived measures that can be useful for the IS auditors to investigate. The IS auditors can extract those measures either directly or indirectly from the measurement objectives of the agile-XP project. More examples of base measures and derived measures can be found in Al Qutaish (2007).

Yardstick #5:

Measurement data and models: The CMMI mentions that "Existing sources of data may have been identified when specifying the measures. Appropriate collection mechanisms may exist whether or not pertinent data have already been collected".

This yardstick investigates the existence of mechanisms for the data collection used during the measurement development. This can provide the IS auditors for agile-XP with evidences that a specific mechanism/method has been developed to collect the measurement data. Several measurement data collection methods can be inspected in this yardstick such as:

- ✓ Qualitative data collection methods
- ✓ Web based questionnaires
- ✓ Paper based questionnaires
- ✓ Computer based Interviewing
- ✓ Face to face interviewing

Modeling artifacts of measurement data is also a valuable resource of evidence, especially for the measurement design based on modeling measurement methods such as COSMIC – ISO 19761; an example of modeling artifact that is important for this yardstick is listed next:

- ✓ Identification of functional processes
- ✓ Data movement diagrams
- ✓ Identification of data groups
- ✓ Identification of measurement functions
- ✓ Activity diagrams
- ✓ Collaboration diagrams
- ✓ Data flow diagrams
- ✓ Decision tables and diagrams

Yardstick #6:

Measurement results: the focus of this auditing yardstick is the investigation of the existence of appropriate data analysis and presentation techniques for better understanding of measurement results. The SEWBOK Guide mentions that measurement users can also participate in reviewing the data to ensure the accuracy of data and that they can be presented in a reasonable manner. Examples of analysis and presentation techniques in CMMI for investigation of the measurement results of agile-XP are listed next.

- ✓ Presentation techniques (e.g., pie charts, bar charts, histograms, radar charts, line graphs, scatter plots, tables).
- ✓ Descriptive statistics (e.g., arithmetic mean, median, mode).

- ✓ The modeling techniques presented in yardstick#5 are also a valuable resource of auditing evidences.

- **Normal configuration of measurement development**

The normal configuration of a device means “the general shape and arrangement that are commonly agreed to best embody the operational principles” Vincenti (1990). In his thesis, Zarour (2009) adds that for production of any device or product, it should consist of a set of sub-devices or sub-products, to support the overall design of the product; and this is what concerns the normal configuration. In the context of this chapter, the normal configuration represents the supplementary techniques that have been designed to support the measurement development. Next are the auditing yardsticks which are associated with this auditing criterion.

Yardstick #7:

Traceability of measurement data: This yardstick focuses on the investigation of the existence of techniques for tracing the measurement data throughout the measurement process. The importance of measurement traceability techniques has been highlighted by CMMI model as a tool for identification of the measures that have been already addressed in the measurement plan.

Yardstick #8:

Prioritization of measurement data: This yardstick focus is the investigation of the existence of techniques for prioritizing the measurement data throughout the measurement process. CMMI mentioned that it is important to develop a prioritization technique throughout the measurement development, as to improve the accuracy of measurement data and enhance the measurement evaluation.

- **Normal technology of measurement development**

"The improvement of the accepted tradition or its application under new or more stringent conditions" is known as ‘normal technology’ Vincenti (1990). For measurement

development, auditing criteria is to investigate the measurement technology/methods used for the design of techniques or procedures for measurement development. Next are the auditing yardsticks which are associated with this auditing criterion.

Yardstick #9:

Identification of measurement design: This yardstick focuses in the investigation of the classification of measurement design that has been used by the measurement team. Vincenti (1990) has classified the engineering design into:

- ✓ Normal design: The main features that characterize the normal design is “evolutionary rather than revolutionary” and “The designer knows at the outset of how the device works and what its customary features are”. The measurement design will be considered to be a normal design if it is based on known measurement methods such as Common Software Measurement International Consortium (COSMIC), Constructive Cost Model (COCOMO) and Constructive Systems Engineering Cost Model (COSYSMO).
- ✓ Radical design: The main features which characterize the radical design are “The designer has never seen such a device before and cannot presume that it will succeed” and “the designer is not familiar with the device itself”. The measurement design will be considered to be a radical design if it does not follow the guidelines of any measurement model and the measurement methods are designed based on team experiences.

7.6.3 C: Auditing criteria for measurement management

The measurement management auditing criteria focus is to identify the auditing yardstick that can reveal evidences at the measurement management level. The measurement management team includes the project leaders’, and the project stockholders who impose important decisions to improve the measurement process, such as decisions on the selected measurement tools, tanning and maintenance. The measurement management auditing criteria is composed of the following yardsticks.

Yardstick #10:

Evaluation of measurement process and results: The measurement result is the output of the analysis and manipulation that has been made on measurement data to achieve results that are useful for project stakeholders. This yardstick is also important to investigate evaluation methodologies (i.e. criteria), for the measurement process and results. SWEBOK and the CMMI model have not specified any evaluation criteria. Recommendations for evaluation criteria have been found in ISO 15393 as the following:

- Timeliness
- Efficiency
- Defect containment
- Customer satisfaction
- Process compliance

For IS auditors of agile-XP, it is important to find evidences that the measurement process and results have been evaluated. Existence of such evaluation criteria can provide more evidences for important measurement activities such as customer involvement, team expertise and/or tools selection process.

Yardstick #11:

Result communication: This auditing yardstick is focused to investigate the existence of an appropriate scheduling and presentation procedures to discuss the measurement result with the project stakeholders. This will include any procedure that assists the project stakeholders to understand the measurement results and keep the project stakeholders informed about the measurement results periodically. CMMI indicates some information that can be beneficial for this yardstick such as:

- ✓ How and why measures were specified.
- ✓ How data were obtained.
- ✓ How to interpret results based on the data analysis methods.
- ✓ How results address information needs.
- ✓ Providing training on the appropriate use and understanding of measurement results.

7.6. Summary

This chapter has proposed an extension to the auditing model proposed in chapter 6. This model can help software organizations in their effort to achieve ISO 9001 certification and help software auditors to extract auditing evidence that demonstrates the ability of a software organization to implement the ISO 9001 measurement requirements. The design process for the auditing model extension is based on Vincenti engineering design, evaluation theory, CMMI_DEV and SWEBOK. The model consists of three major categories of auditing criteria: measurement plan criteria, measurement development criteria and measurement management criteria. Each auditing criterion consists of several auditing yardsticks which focus on the evidences that can be extracted to demonstrate process conformity with ISO 9001 measurement requirements.

CHAPTER 8

CASE STUDIES

8.1 Introduction

The International Standard of Auditing ISA (2009) defines audit evidence as "all the information used by the auditor in arriving at conclusions on which the audit opinion is based." The audit evidence is described by the ISA as "proofs, facts and information about something to convince the auditors that something is true, fair or false. It gives auditors reasonable assurance and not absolute assurance about something." This standard was designed for auditing financial systems and financial records, and examples of auditing evidence are: counting records, internal and external documents, and physical observations.

For information systems, auditors usually look for evidence of the existence of internal controls. The Control Objectives for Information and related Technology ISACA (2008) defines internal IT controls as specific activities performed by persons or systems designed to ensure that business objectives are met. As indicated by Control Objectives for Information and related Technology (COBIT), internal IT controls can be implemented at different levels (organization, process, and product) to support business objectives, such as process activity integrity, reliability, and compliance.

The case studies presented in this chapter are based on the proposals found in the literature for the traceability of agile software processes and the measurement of agile software processes. The main limitations for evaluating this research project based on industrial case studies are summarized next:

- **Geographical area interest:** The agile software processes are still new methodologies for software organizations in general, and it is difficult to find software organizations in Montréal area that had adopted the agile software process and which had got ISO 9001 certification.
- **Time and budget constraint:** The evaluation of the proposed auditing models requires a close collaboration and training for the development team in order to setup the basic requirements for the development of certified agile software process: this requires time availability for both the trainer (i.e. the research team) and the trainee (i.e. the development team). Furthermore, a complete evaluation may require more than one project to be audited as to cover all the auditing yardsticks that have been proposed for the traceability auditing model and the measurement auditing model.

The limitations above have been avoided by selecting the case studies based on the literature found for both agile measurement and agile traceability, such that the research team was not limited to any geographical area, and budget constraint.

This chapter presents two set of case studies:

1. The first set of case studies consists of five different agile traceability approaches.
2. The second set of case studies consists of four different agile measurement approaches.

Each approach that is related to agile traceability has been denoted as (Case_ alphabet_TR). The alphabet is used for sequence ordering and “TR” is to indicate that the case study belongs to agile traceability. On the other hand, each approach that is related to agile measurement has been denoted as (Case_ alphabet_MR). The alphabet is used for sequence ordering and “MR” is used to indicate that the case studies belong to agile measurement.

This chapter is organized as follows:

Section 8.2 presents classification of auditing evidences.

Section 8.3 presents the selected agile traceability approaches (five case studies) and presents the analysis of each case study based on the auditing model proposed in chapter 6.

Section 8.4 presents the selected agile measurement approaches (four case studies) and presents the analysis of each case study based on the auditing model proposed in chapter 7.

Section 8.5 presents the chapter summary.

8.2 Classification of auditing evidences

The auditing evidences can be in different forms, such as diagrams of UML, structured development artifacts, low level design artifacts and/ or simple text that provide details of the design planning activities, data collection management, analysis of collected data, etc. Therefore, the evidences will be classified into three main categories:

- **Textual evidence:** Can be described as supporting proof in certain text format. In this case, the designer is expected to provide reasons or explanations for their opinion. This provides readable information for the auditors but it is inheriting the disadvantages of natural language such as inconsistency and misunderstanding. Therefore, this type of evidence needs a certain degree of expertises from the auditors to clarify whether the evidences are clearly stated or not.
- **Modeling evidence:** Can be described as supporting proof in certain modeling format. In this case, the designer is expected to provide low level clarification or identification for their design component. This provides visual information for the auditors. In software engineering several modeling languages can be used by the designer to provide such type of evidences such as Unified Modeling Language (UML), Business Process Modeling Notation (BPMN), Service-Oriented Modeling Framework (SOMF), Extended Enterprise Modeling Language (EEML), mathematical modeling and notations.
- **Graphical evidence:** Can be described as supporting proof in certain visual formal. In this case, the designer is expected to provide low level clarification or identification for the outcome of their design. This will also provide the auditors with visual information. In software engineering several graphical representations can be used by the designer to

provide such type of evidences, such as bar graphs, line graphs, pie charts, scatter plots, photographs, and line charts. Burn down chart is an example of this type of evidence that is used in the context of agile software processes.

8.3 Case studies: agile traceability audit

8.3.1 Context and scope

To study the applicability of the auditing model proposed in chapter 6, five case studies have been selected to be compatible with the scope defined in chapter 6. The selection of these case studies is based on the following steps and criteria – see Table 8.1:

- ✓ A search was conducted of IEEE Xplore and ScienceDirect-Elsevier for any paper proposing a methodology, technique, or framework to enhance the traceability of an agile software process.
- ✓ The following terms were used in browsing the content of IEEE Xplore, ScienceDirect-Elsevier:
 - Agile AND traceability
 - XP AND traceability
 - Agile AND configuration management
 - XP AND configuration management
- ✓ A title and abstract analysis was performed to select the papers that discuss XP traceability. Note that some authors discuss the principles of XP, but refer to them as agile principles (i.e. they do not specify which agile process they are improving, and selected XP as a candidate process without referring to it by name).
- ✓ All the papers that discuss agile traceability in general, without proposing a methodology, technique, or framework for managing traceability, were discarded.
- ✓ As some authors discuss the same proposed traceability approach in a number of different research articles, only their most recently published article was selected.

XP was proposed by Kent Beck in 1999, and very few papers discuss agile software process traceability or XP traceability prior to 2003. As a result, the case studies that have been identified were published between 2003 and 2011.

The main characteristics of the five selected proposals for traceability agile process are described next- See table 8.1:

- **Case A_{TR}** (Espinoza, Garbajosa): Case A_{TR} was published in the Journal of Innovations in Systems and Software Engineering. The proposal is part of a PhD thesis in software engineering which aims to improve the traceability of agile software processes. The title of the PhD thesis is “An advanced traceability schema to improve supporting life cycle process”. The study has been held in Technical University of Madrid under the supervision of Prof. Juan Garbajosa. The proposed model has been evaluated in the context of hardware drive unit testing, agile software process and software product line engineering. Case A_{TR} has been cited five times since 2011 based on Google scholar.
- **Case B_{TR}** (Lee, *et al*): Case B_{TR} was published in the proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering. The tool proposed by Case B_{TR} is part of a master thesis in computer science which aims to enhance the workflow of agile software processes. The title of the master thesis is “FLUID: knowledge creation for emergent workflows”. The study has been held in DePaul University under the supervision of Prof. Xiaoping Jia. Case B_{TR} has been cited thirteen times since 2003 based on Google scholar.
- **Case C_{TR}** (Ghazarian): Case C_{TR} was published in the proceedings of the 8th International Conference on Applied Computer Science. The proposal is part of a PhD thesis which aims to support the software maintenance through mechanism for establishing traceability relations between the system requirements and its code elements. The title of the PhD thesis is “A design rule based constructive approach to building traceable software”. The study has been held at Toronto University under the supervision of Prof. Dave Wortman. Case C_{TR} has been cited six times since 2008 based on Google scholar.

- **Case D_{TR}** (Ratanotayanon, *et al*): Case C_{TR} was published in the proceedings of the 2009 Agile Conference. Case C_{TR} is part of a project developed at the University of California called Zelda. The objective of Zelda is to design and implement a plug-in for Eclipse to create, visualize and maintain links between concerns and their implementation across software artifacts. Case D_{TR} has been cited three times since 2009 based on Google scholar.
- **Case E_{TR}** (Yaser, Maurer): Case C_{TR} was published in the proceedings of the 2009 Agile Conference. The proposal is part of a PhD thesis which aims to make it possible for agile organizations to address variability in a product line without affecting agility values and principles. The title of this research project is “Agile Product Line Engineering”. The project has been developed under the supervision of Prof. Frank Maurer at University of Calgary.

The audit of agile traceability case studies refers to the set of mechanisms, techniques, approaches and/or documentations that are implemented to support the traceability method pertaining to an internal control for agile software process traceability. For all the five case studies in this section, the evidences were gathered using the information system audit procedure described by the standards, guidelines and procedures for information system auditing ISACA (2010), as follows:

- Observation of traceability processes and the existence of the components of the traceability method.
- Documentary audit evidence, such as results of the traceability method execution, and records of the method performance.
- Representations of the method, such as written analyses, and descriptions of the traceability method and traceability method flowcharts.

Table 8.1 Selected case studies for the agile (XP) traceability audit

	Authors	Case title	Case objective	Case Date
Case A_{TR}	Espinoza, Garbajosa	A Study to Support Agile Methods More Effectively Through Traceability	To propose a model, called the traceability meta model (TmM), to support the traceability of XP by developing three features of the TmM which are user-definable traceability links, roles, and linkage rules. The proposed model is aimed at improving and enhancing XP maintainability processes.	March, 2011
Case B_{TR}	Lee, <i>et al</i>	An Agile Approach to Capturing Requirements and Traceability	To propose a tool, called Echo, to capture user stories, and any informal information generated during the development phases, and restructure that information to better support XP traceability and change management.	October, 2003
Case C_{TR}	Ghazarian	Traceability Patterns: An Approach to Requirement-Component Traceability in Agile Software Development	To propose a conceptual model that captures a traceability pattern in XP. The approach focuses on providing traceability through mapping the user stories to the source code components after defining certain design constraints, such as the location, naming, and content constraints.	November, 2008
Case D_{TR}	Ratanotayanon <i>et al</i>	Supporting Program Comprehension in Agile with Links to User Stories	To propose Zelda, an Eclipse plug-in tool designed to work with XP to support the traceability of source codes generated using agile software processes by helping developers create links from user stories to source code, and test cases.	August, 2009
Case E_{TR}	Yaser and Maurer	Extreme Product Line Engineering: Managing Variability & Traceability via Executable Specifications	To propose an approach for managing XP variability and traceability using executable specifications.	August, 2009

8.3.2 Traceability audit of Case A_{TR} (Espinoza, Garbajosa)

The identification of evidences existence in Case A_{TR} (Espinoza, Garbajosa) are presented in Table 8.2. The process focuses on providing links between the traceability yardsticks to its supporting type and location of evidences found in Case A_{TR}.

- **Yardstick_{TR} #1 (Identification of design intellectual concepts):** This yardstick investigates the intellectual concepts of the traceability design. The design will be classified as intellectual concepts if it represents the ideas people have in mind: chapter 6 presents the details about this yardstick. Case A_{TR} supports for textual and modeling evidences for this yardstick. More precisely this evidence is found in section 4.1 “TmM extended from SEMDM” and figure 4 "Traceability methodology, represented with the Traceability Conglomerate class".
- **Yardstick_{TR} #2 (Identification of design mathematical models):** This yardstick investigates the mathematical models of the traceability design. The design will be classified as mathematical modeling if it describes the traceability design using the mathematical concepts and notations. Case A_{TR} has no evidence for this yardstick.
- **Yardstick_{TR} #3 (Full operational principles):** The traceability design will be considered to support full operational principles if the design covers the artifact traceability for different life cycle phases. Case A_{TR} supports this yardstick with textual and modeling evidences. More precisely this evidence is found in section 4.4 “TmM usage process” and in figure 5 "TmM core (part I): Twp".
- **Yardstick_{TR} #4 (Partial operational principles):** The traceability design will be considered to support full operational principles if the design focuses on the relationships between entities developed in the same phase. Case A_{TR} supports this yardstick with modeling evidences. More precisely this evidence is found in figure 5 "TmM core (part I): Twu".
- **Yardstick_{TR} #5 (Traceability item identification):** This yardstick investigates the identification activities for the traceability items (i.e. artifacts). Case A_{TR} supports this yardstick with textual and modeling evidences. More precisely these evidences are found

in section 4.3 “TmM core: templates and resources”, in figure 5 “TmM core (part I): Twp” and in figure 5 “TmM core (part I): Twu”.

- **Yardstick_{TR} #6 (Traceability item relationships):** This yardstick investigates for the identified dependencies between the traceability items within a specific development phase or within the entire software life cycle. Case A_{TR} supports this yardstick with textual and modeling evidences found in section 4.3 TmM core: templates and resources and in figure 4 traceability methodology, represented with the traceability conglomerate class.
- **Yardstick_{TR} #7 (Traceability role identification):** This yardstick investigates the privileges assigned for the project stakeholders to access or modify the traceability items. Case A_{TR} supports this yardstick with modeling evidences. More precisely these evidences are found in figure 5 “TmM core (part I): Tp” and in figure 7 "TmM usage process".
- **Yardstick_{TR} #8 (Traceability documentation):** This yardstick investigates the documentation produced to support the traceability design. The traceability documentation provides the project stakeholders with useful information regarding project status. Case A_{TR} supports this yardstick with textual and modeling evidences. More precisely these evidences are found in 4.3 "TmM core templates and resources" and in figure 6 "TmM core (part II): traceability resources represented with classes".
- **Yardstick_{TR} #9 (Documentation access):** This yardstick investigates the design of documentation access method. This includes the storage and retrieval mechanisms and the right of access that has been granted based on the stakeholders role. Case A_{TR} supports this yardstick with modeling evidences. More precisely these evidences are found in figure 4 “Traceability methodology, represented with the traceability conglomerate class” and in figure 7 “TmM usage process”.

Table 8.2 Traceability audit of case A_{TR}

Case A_{TR} (Espinoza, Garbajosa)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{TR} # 1 (Identification of design intellectual concepts)	Evidence exists	Modeling and textual evidences	Section 4.1 and figure 4	Page # 67
Yardstick_{TR} # 2 (Identification of design mathematical models)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #3 (Full operational principles)	Evidence exists	Modeling and textual evidences	Section 4.4 and figure 5	Page # 61
Yardstick_{TR} #4 (Partial operational principles)	Evidence exists	Modeling evidences	Figure 5	Page # 60
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Modeling and textual evidences	Section 4.3 and figure 5	Page # 60 Page #61
Yardstick_{TR} #6 (Traceability item relationships)	Evidence exists	Modeling and textual evidences	Section 4.3 and figure 4	Page # 59 Page # 60
Yardstick_{TR} #7 (Traceability role identification)	Evidence exists	Modeling evidence	Figure 5 and figure 7	Page # 61 Page # 62
Yardstick_{TR} #8 (Traceability documentation)	Evidence Exists	Modeling and textual evidences	Section 4.3 and figure 6	Page # 59 Page # 61
Yardstick_{TR} #9 (Documentation access)	Evidence exists	Modeling evidences	Figure 4 and figure 7	Page # 59 Page # 62

Case B_{TR} fully supports five traceability auditing yardsticks out of nine. The fully supported traceability auditing yardsticks are yardstick_{TR}#1 (Identification of design intellectual concepts), yardstick_{TR} #4 (Partial operational principles), yardstick_{TR}#5 (Traceability item identification), yardstick_{TR}#6 (Traceability item relationships) and yardstick_{TR}#8 (Traceability documentation). Case B_{TR} partially supports yardstick_{TR}#9 (Documentation

access). No auditing evidence was found to support yardstick _{TR}# 2 (Identification of design mathematical models), yardstick _{TR} #3 (Full operational principles) and yardstick _{TR} #7 (Traceability role identification). Table 8.3 shows the auditing evidences type and location found in Case B_{TR}.

Case C_{TR} fully supports five traceability auditing yardsticks out of nine. The fully supported traceability auditing yardsticks are yardstick _{TR}#1 (Identification of design intellectual concepts), yardstick _{TR}#4 (Partial operational principles), yardstick _{TR}#5 (Traceability item identification) yardstick _{TR}#6 (Traceability item relationships), and yardstick _{TR}#8 (Traceability documentation). No auditing evidence was found to support yardstick _{TR}#2 (Identification of design mathematical models), yardstick _{TR}#3 (Full operational principles), yardstick _{TR}#7 (Traceability role identification) and yardstick _{TR}#9 (Documentation access). Table 8.4 shows the auditing evidences type and location found in Case C_{TR}.

Case D_{TR} fully supports five traceability auditing yardsticks out of nine. The fully supported traceability auditing yardsticks are yardstick _{TR} #1 (Identification of design intellectual concepts), yardstick _{TR} #3 (Full operational principles), Yardstick _{TR} #5 (Traceability item identification), Yardstick _{TR}#6 (Traceability item relationships), Yardstick _{TR} #8 (Traceability documentation). Case D_{TR} partially supports yardstick _{TR}#9 (Documentation access). No auditing evidences was found to supports yardstick _{TR}#2 (Identification of design mathematical models), yardstick _{TR}#4 (Partial operational principles), and yardstick _{TR}#7 (Traceability role identification). Table 8.5 shows the auditing evidences type and location found in Case D_{TR}.

Case E_{TR} fully supports three traceability auditing yardsticks out of nine. The fully supported traceability auditing yardsticks are yardstick _{TR} #1 (Identification of design intellectual concepts), yardstick _{TR}#4 (Partial operational principles) and yardstick _{TR} #5 (Traceability item identification). No auditing evidence was found to support yardstick _{TR}#2 (Identification of design mathematical models), yardstick _{TR} #3 (Full operational principles), Yardstick _{TR}#6 (Traceability item relationships), yardstick _{TR}#7 (Traceability role identification),

yardstick_{TR}#8 (Traceability documentation) and yardstick_{TR}#9 (Documentation access).

Table 8.6 shows the auditing evidences type and location found in Case E_{TR}.

Table 8.3 Traceability audit of case B_{TR}

Case B _{TR} (Lee <i>et al.</i>)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{TR} #1 (Identification of design intellectual concepts)	Evidence exists	Textual evidence	Section 4.3 “Prototype Architecture and Implementation” and figure 4.	page #4 page #5
Yardstick_{TR} #2 (Identification of design mathematical models)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #3 (Full operational principles)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #4 (Partial operational principles)	Evidence exists	Textual evidence and modeling	"Prototype Architecture and Implementation" and section 5 "Future work"	page #5 page #7
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Modeling and textual evidences	Section 4.3 “Prototype Architecture and Implementation” and figure 4.	page #4 page #5
Yardstick_{TR} #6 (Traceability item relationships)	Evidence exists	Modeling and textual evidences	Section 4.3 “Prototype Architecture and Implementation” and figure 4.	page #4 page #5
Yardstick_{TR} #7 (Traceability role identification)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #8 (Traceability documentation)	Evidence exists	Modeling and textual evidences	Figure 6 Figure 7 Figure 8	page #5 page #9
Yardstick_{TR} #9 (Documentation access)	Evidence partially exists	Modeling and textual evidences	Section 4.3 “Prototype Architecture and Implementation” and figure 4.	page #4 page #5

Table 8.4 Traceability audit of case C_{TR}

Case C_{TR} (Ghazarian)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{TR} # 1 (Identification of design intellectual concepts)	Evidence exists	Textual evidence	Section 4 "Traceability patterns"	page #238
Yardstick_{TR} # 2 (Identification of design mathematical models)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #3 (Full operational principles)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #4 (Partial operational principles)	Evidence exists	Modeling evidences	Figure 1	page #239
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Modeling and textual evidences	Section 4 "Traceability patterns" and figure 1	page #238 page #239
Yardstick_{TR} #6 (Traceability item relationships)	Evidence exists	Modeling and textual evidences	Section 4 "Traceability patterns" and figure 1	page #238 page #239
Yardstick_{TR} #7 (Traceability role identification)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #8 (Traceability documentation)	Evidence exists	Modeling and textual evidences	Section 4 "Traceability patterns" and figure 2	page #238 page #240
Yardstick_{TR} #9 (Documentation access)	Evidence does not exist	Not applicable	Not applicable	Not applicable

Table 8.5 Traceability audit of case D_{TR}

Case D_{TR} (Ratanotayanon <i>et al.</i>)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{TR} # 1 (Identification of design intellectual concepts)	Evidence exists	Textual and modeling evidences	Section IV “Zelda” and figure 3	page #28 page #29
Yardstick_{TR} # 2 (Identification of design mathematical models)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #3 (Full operational principles)	Evidence exist	Textual evidences	Section II “Supporting program comprehension with links to user stories”	page #27
Yardstick_{TR} #4 (Partial operational principles)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Modeling and textual evidences	Section III-B "Updating link locations" and figure 2	page #27 page #28
Yardstick_{TR} #6 (Traceability item relationships)	Evidence exists	Modeling and textual evidences	Section III-B "Link Recording" figure 1	page #27 page #28
Yardstick_{TR} #7 (Traceability role identification)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #8 (Traceability documentation)	Evidence exists	Modeling and textual evidences	Section III-B "Link Recording" figure 1	page #27 page #28
Yardstick_{TR} #9 (Documentation access)	Evidence partiality exist	Modeling and textual evidences	Section IV “Zelda” and figure 3	page #28 page #29

Table 8.6 Traceability audit of case E_{TR}

Case E_{TR} (Ghanam, Maurer)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{TR} # 1 (Identification of design intellectual concepts)	Evidence exists	Textual evidences	Section 2.1 "Organizing test artifacts"	Page #42
Yardstick_{TR} # 2 (Identification of design mathematical models)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #3 (Full operational principles)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #4 (Partial operational principles)	Evidence exists	Textual evidences	Section 2.2 "Introducing variability"	Page #43
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Modeling and textual evidences	Section 2.2 "Introducing variability" and figure 1	Page #43
Yardstick_{TR} #6 (Traceability item relationships)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #7 (Traceability role identification)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #8 (Traceability documentation)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{TR} #9 (Documentation access)	Evidence does not exist	Not applicable	Not applicable	Not applicable

8.3.3 Auditing for the five case studies based on yardstick_{TR} #6 (Traceability item relationships)

This yardstick investigates the existence of techniques developed for the identification of the relationships and dependencies between the traceability items. In this section the process of identifying the existence of evidences for the yardstick_{TR} #6 is clarified. This process focuses on providing links between the yardstick_{TR} #6 to its supporting type and location of evidences found in Case B_{TR}, Case C_{TR}, Case D_{TR} and Case E_{TR}. Yardstick_{TR} #6 for Case A_{TR} has been analyzed in the previous section. Table 8.7 describes the auditing results of traceability case studies for yardstick_{TR} #6.

Case B_{TR} supports Yardstick_{TR} #6 with both modeling and textual evidences. Case B_{TR} developed a navigation technique to provide various accesses to traceability items. The Navigation technique is based on annotation mechanisms to build access rules between traceability items. The textual and modeling evidences are found in section 4.3 "prototype architecture and implementation" and in figure 4 "prototype architecture".

Case C_{TR} supports Yardstick_{TR} #6 with both modeling and textual evidences. Case C_{TR} developed a clustering technique to identify the relationships and dependences between the traceability items. This clustering technique is based on "location constraint", "naming constraint" and "content constraints". The textual and modeling evidences are found in section 4 "Traceability Patterns" and figure 1 "conceptual model of traceability patterns".

Case D_{TR} supports Yardstick_{TR} #6 with both modeling and textual evidences. Case D_{TR} created a tool to identify and store links by integrating together different parts of traceability items. To facilitate in recording links, the tool provides an interface to communicate to a user story management tool. The textual and modeling evidences are found in 3.1 "link recording" and in figure 1 "model of links data".

Case E_{TR} does not have evidence to support Yardstick_{TR} #6.

Table 8.7 Audit based on yardstick $_{TR}$ #6

Audit for Yardstick TR #6 (Traceability item relationships)				
	Case B (Lee <i>et al.</i>)	Case C (Ghazarian)	Case D (Ratanotayanon <i>et al.</i>)	Case E (Ghanam, Maurer)
Evidence Status	Evidence exists	Evidence exists	Evidence exists	Evidence does not exist
Evidence Type	Modeling and textual evidences	Modeling and textual evidences	Modeling and textual evidences	Not applicable
Evidence Location	Section 4.3 and figure 4	Section 4 and figure 1	Section 3.1 figure 1	Not applicable
Page number	page #4 page #5	page #238 page #239	Page # 27 Page # 28	Not applicable

The auditing criteria in table 8.8 consists first of engineering criteria that are decomposed into design of the traceability method and coverage of the traceability method. Second, the management criteria are decomposed into identification of the traceability method and monitoring of the traceability method.

Table 8.8 shows a summary of the traceability auditing results for five the case studies based on the auditing model proposed in chapter 6 to determine whether or not they can provide evidence of the implementation of the audit yardsticks.

The following comments can be made based on the evidence gathered- See table 8.8:

- The traceability method in Case A_{TR} implements a meta model for agile process traceability based on the ISO-24744:2007 meta model, which was designed based on the UML architecture and notation. Case B was also designed based on the UML architecture and notation. Both cases A_{TR} and Case B_{TR} therefore provide evidence of intellectual concept design rather than mathematical model design; similarly for Cases C_{TR}, D_{TR}, and E_{TR}.
- Case B_{TR} shows partial evidence of operational principles, as the traceability approach only covers the requirements phase; similarly for Case C_{TR}, since it shows support for

traceability for the requirements, design, and coding phases. No evidence was found to the traceability in the planning, testing, validation, and verification phases for both Case B_{TR} and Case C_{TR}.

- For Case B_{TR}, there is partial support for the documentation access audit yardstick, since a mechanism was implemented in this case for accessing and retrieving the traceability items produced during the requirements phase, but there is no evidence of right of access mechanisms. The same is true for Case D_{TR}.
- No evidence was found for traceability role identification in Case B_{TR}, and the project stakeholders have the same right to access, modify, and retrieve the traceability items. The same is true for Case D_{TR}.
- Little evidence was found of support for the audit model in Case E_{TR}. The approach presented in Case E_{TR} was implemented to support traceability between the coding and testing phases in XP.
- For Case E_{TR}, no evidence was found for traceability item relationship identification or traceability role identification. Nor was evidence found of traceability documentation, such as traceability logs, the history of traceability items, and the relationships among traceability items, and so on. No evidence was found supporting documentation access or access rights either.

Table 8.8 Summary of evidences in the selected case studies

	Case A _{TR} (Espinoza, Garbajosa)	Case B _{TR} (Lee <i>et al.</i>)	Case C _{TR} (Ghazarian)	Case D _{TR} (Ratanotayano n <i>et al.</i>)	Case E _{TR} (Ghanam, Maurer)
Engineering criteria					
	Design of the traceability method				
Yardstick_{TR} # 1 (Identification of design intellectual concepts)	Evidence exists	Evidence exists	Evidence exists	Evidence exists	Evidence exists
Yardstick_{TR} # 2 (Identification of design mathematical models)	Evidence does not exist	Evidence does not exist	Evidence does not exist	Evidence does not exist	Evidence does not exist
	Coverage of the traceability method				
Yardstick_{TR} #3 (Full operational principles)	Evidence exists	Evidence does not exist	Evidence does not exist	Evidence exists	Evidence does not exist
Yardstick_{TR} #4 (Partial operational principles)	Evidence exists	Evidence exists	Evidence exists	Evidence does not exist	Evidence exists
Management criteria					
	Identification of the traceability method				
Yardstick_{TR} #5 (Traceability item identification)	Evidence exists	Evidence exists	Evidence exists	Evidence exists	Evidence exists
Yardstick_{TR} #6 (Traceability item relationships)	Evidence exists	Evidence exists	Evidence exists	Evidence exists	Evidence does not exist
Yardstick_{TR} #7 (Traceability role identification)	Evidence exists	Evidence does not exist	Evidence does not exist	Evidence does not exist	Evidence does not exist
	Monitoring of the traceability method				
Yardstick_{TR} #8 (Traceability documentation)	Evidence exists	Evidence exists	Evidence exists	Evidence exists	Evidence does not exist
Yardstick_{TR} #9 (Documentation access)	Evidence exists	Evidence partially exists	Evidence does not exist	Evidence partially exists	Evidence does not exist

8.4 Case studies: agile measurement audit

8.4.1 Context and scope

To study the applicability of the auditing model proposed in chapter 7, four case studies have been selected to be compatible with the scope defined in chapter 7. See Table 8.9. The selection of these case studies is based on the following steps and criteria:

- ✓ A search was conducted of IEEE Xplore and ScienceDirect-Elsevier for any paper proposing a methodology, technique, or framework to enhance the measurements of an agile software process.
- ✓ The objective is to select an agile measurement proposal that is mature enough to be considered as a measurement program for the agile process, such that: the proposal will be given a priority for selection if the following conditions are found.
 - It covers the activities for measurement planning
 - It covers activities for measurement design
 - It covers activities for measurement management
- ✓ The following terms were used in browsing the content of IEEE Xplore, Science Direct-Elsevier:
 - Agile AND measurement program
 - XP AND measurement program
 - Agile AND measurement techniques OR method.
 - XP AND measurement techniques OR method.
- ✓ A title and abstract analysis was performed to select the papers that discuss agile measurement. Note that some authors discuss different agile processes such as XP and Scum, but refer to them as agile process (i.e. they do not specify which agile process they are improving, and selected XP as a candidate process without referring to it by name).
- ✓ It has been observed that authors discuss the principle of Scrum as a candidate process for the purpose of agile measurement. A detailed analysis of the proposal reveals that the authors mixed the principles of Scrum with authors' agile software processes such as XP.

- ✓ All the papers that discuss agile measurement in general, without proposing a methodology, technique, or framework for managing the measurement process, were discarded.
- ✓ As some authors discuss the same proposed agile measurement approach in a number of different research articles, only their most recently published article was selected.
- ✓ Any paper focus to measure the agility of the process has been discarded. On the other hand there was no restriction on the focus or the goal of the agile measurement technique (i.e. No restriction on which process attributes are intended to be measured or which methodology has been used to achieve the proposed measurement goal).

For the four case studies in this section, the evidences were gathered using the information system audit procedure described by the standards, guidelines and procedures for information system auditing (ISACA,2010), as follows:

- Observation of measurement processes and the existence of the components of the measurement method.
- Documentary audit evidence, such as results of the measurement method execution, and records of the method evaluation.
- Representations of the measurement results, such as written analyses, and descriptions of the measurement method flowcharts.

Table 8.9 Selected case studies for the audit of agile (XP) measurement

	Authors	Case title	Case objective	Case Date
Case A MR	Ktata, Lévesque	Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?	To propose a measurement program for agile software process in software organization. The goal of this project is to allow teams and individuals to improve their development process and to provide better product quality and control over the project. The Goal Question Metric approach (GQM) has been used to identify a set of indicators. The indicators have been prioritized based on the improvement area that all stakeholders agree upon.	May 2010
Case B MR	Mahnic, Zabkar	Introducing CMMI Measurement and Practices into Scrum-based Software Development Process	To develop a measurement program for agile software process based on CMMI measurement and analysis key process area (KPA). The agile software process has been modeled using the entity-relationship notation. After the authors have established the measurement objective a collection of "base" and "derived" measure has been selected. A procedure for data collection, analysis and storage is also identified.	June, 2007
Case C MR	Gustafsson	Model of Agile Software Measurement : A Case Study	To propose a measurement model for performance measurement and process optimization measurement in agile software process environment. The model is also evaluated on a web game software organization in Sweden. The author identifies a set of process indicators and then proposes a set of metrics to be applied in the agile software process environment.	June, 2011
Case D MR	Fehlmann	Six Sigma for Agile Teams	To propose a measurement technique for agile software processes based on the methods used in Six Sigma and COSMIC for Software Functional Size Measurement (FSM). The author mentions that the proposed approach has been inspired by the work presented in IWSM / MetriKon / Mensura conference in Stuttgart 2010.	March, 2011

The main characteristics of the four selected proposals for measurement agile processes are described next:

- **Case A_{MR}** (Ktata, Lévesque): Case A_{MR} was published in the proceedings of the third C* Conference on Computer Science and Software Engineering. Case A_{MR} has been developed at University of Quebec under the supervision of Prof. Ghislain Lévesque. The author also has an industrial experience in agile organizations such as Pyxis Technologies and Agile Tour Montreal. Case A_{MR} has been validated based on an industrial experiment and cited 3 times since 2010.
- **Case B_{MR}** (Mahnic, Zabkar): Case B_{MR} has been published in the International journal of Mathematics and Computers in Simulation. Case B_{MR} has been developed as a research collaboration between Prof. Viljan Mahnic and Prof. Natasa Zabkar at University of Ljubljana, Ljubljana, Slovenia. Both authors have several publications in the area of agile software process improvement.
- **Case C_{MR}** (Gustafsson): Case C_{MR} is a master thesis at Chalmers University of Technology, Sweden and defended in June 2011 under the supervision of prof. Robert Feldt. Case C_{MR} is part of a project called “Agile and Lean Software Practices” established by Prof. Robert Feldt which aims to develop techniques to improve the agile processes in the context of software engineering practices. The group consists of several researchers at the PhD and master levels. Case C_{MR} has been also evaluated based on industrial experiment.
- **Case D_{MR}** (Fehlmann): Case D_{MR} has been published in the International Conference on the Modern art of Software. Case D_{MR} is authored by Dr. Thomas Fehlmann who is the founder of euro project office. The mission of this office is to provide the software organizations with several assistance including software measurement and agile training. Since 2005 several publications has been authored by Dr. Thomas Fehlmann in the area of agile software processes improvement.

8.4.2 Preliminary considerations

- During the search process for the case studies applicable to this research project the priority was given to research proposals that discuss a design of process for measurement program that is aligned with the principles of agile software process. A Measurement program is defined by ISO 15939 as the process for establishing, planning, performing and evaluating measurement within an overall project, enterprise or organizational measurement structure. The second priority was given to research proposals that discuss the design of a measurement method, measurement function and/or measurement procedure.
- Generally, the papers that have proposed solution for agile measurement have been published between the years of 2006 to 2011. This reflects the awareness since 2006 to integrate solutions into agile software practices and to investigate opportunities for improvement, manage workloads, reduce overtime, and improve communications in agile environment.
- In his book Hazzan and Dubinsky (2008) have claimed that techniques that have been proposed for measurements in traditional software process are also valid in agile software development “after adjustment and refinement”. The search process for this case study is also confirming this claim. Most of the measurement solutions found for agile software process are based on measurement models that have been proposed for traditional software, such as COSMIC (ISO 19761), ISO 15939 and CMMI.
- It should be noted that the objective of this set of four case studies it is to check for the existence of auditing evidences that support the auditing criteria and auditing yardsticks that have been developed in chapter 7. Evaluation of the performance and/or maturity of the proposed solution are considered out scope of this section. As well, any information that is not covered by the auditing criteria and auditing yardsticks will not be considered. "Auditors are not expected to address all the information that may exist" (ISA, 500).

8.4.3 Measurement audit of Case B_{MR} (Mahnic, Zabkar)

This section presents the process for identifying the existence of each evidence for the Case B_{MR}. This process focuses on providing links between the measurement yardsticks to its supporting type and location of evidences found in Case B_{MR}. Table 8.10 describes the auditing results of Case B_{MR} (Mahnic, Zabkar).

- **Yardstick_{MR} #1 (Identification of the measurement context):** This yardstick focus is to investigate the existence of evidences that identify the context, goal and/or objective of the measurement process. Case B_{MR} supports this yardstick with textual evidence. More precisely this evidence was found in section III-A "establish measurement objectives" of Case B_{MR}. More clarification and details of this evidence is found in table I "stakeholders goals" in Case B_{MR}.
- **Yardstick_{MR} #2 (Role assignment):** This yardstick focuses on identifying the measurement team roles and the responsibilities of each team member. Case B_{MR} supports this yardstick with modeling evidence. More precisely, this evidence was found in section V "Database tables" in Case B_{MR}. More clarification and details of this evidence was found in table IV "Project tables". These rational tables are designed to clarify the roles and relationships among projects, teams and employees, and include recording of administrative days and absence type. For each administrative day the number of hours the employee was not at work is recorded.
- **Yardstick_{MR} #3 (Resources and budget constraints):** This yardstick focuses on identifying the major project milestones, the schedule assumptions, and task dependencies and constraint. Case B_{MR} supports this yardstick with modeling evidence. More precisely, this evidence is found in section V "Database tables" in Case B_{MR}. More clarification and details of this evidence was found in table V "Release tables" and table VI "Measurement tables". These rational tables are designed to identify several release attributes such as release ID#, release description, sprint description, sprint begin date, sprint end date, sprint length, sprint estimated date, team ID, project ID and so on. Table VI "Measurement tables" also consists of several measurement attributes such as measure ID,

measure name, measure description, release ID, measure ID, date, measurement result, task ID, measure ID#, date#, measurement result, and so on.

- **Yardstick_{MR} #4 (Identification of measures):** This auditing yardstick investigates the classification of measures that have been specified for the agile measurement design. Case B_{MR} supports this yardstick with textual evidences. More precisely this evidence was found in section III-A "Establish measurement objectives" of Case B_{MR}. More clarification and details of this evidence was found in table I "Base measures" in Case B_{MR}.
- **Yardstick_{MR} #5 (Measurement data and models):** This yardstick investigates the existence of mechanisms for the data collection and modeling used during the measurement development. Case B_{MR} supports this yardstick with textual and modeling evidences. More precisely the textual evidence was found in section III-D "Specify analysis procedures" of Case B_{MR}. The objective of section III-D "Specify analysis procedures" is to classify the measurement data into derived measures or "indicators" help for analyzing software process performance in comparison to target values set by the software organization. The modeling evidence was found in section III-E "measuring earned value" of Case B_{MR}. Several mathematical equations have been proposed to compute the schedule performance and cost performance.
- **Yardstick_{MR} #6 (Measurement results):** The focus of this auditing yardstick is the investigation of the existence of appropriate data analysis and presentation techniques for better understanding of the measurement technique and results. Case B_{MR} supports this yardstick with modeling evidence. This modeling evidence is found in figure 2 "Measurement repository design" of section IV "Repository design". On the other hand no evidence has been found to indicate the presentation method for the measurement results.
- **Yardstick_{MR} #7 (Traceability of measurement data):** This yardstick focuses on the investigation of the existence of techniques for tracing the measurement data throughout the measurement process. Case B_{MR} supports this yardstick with modeling evidence to clarify the traceability of measurement data. This modeling evidence is found in table V "Release tables" and table VI "Measurement tables". These rational tables have been

designed to clarify the relations between the measurement data obtained for each specific project release.

- **Yardstick_{MR} #8 (Prioritization of measurement data):** This yardstick focus is the investigation of the existence of techniques for prioritizing the measurement data throughout the measurement process. No evidence has been found in Case B_{MR} to support this yardstick.
- **Yardstick_{MR} #9 (Identification of measurement design):** This yardstick focuses on the investigation of the classification of measurement design. Textual evidences are found in Case B_{MR} to support this yardstick. The design of Case B_{MR} is based on known software methodology for designing techniques for measurement processes (i.e. CMMI). These evidences are found in section III-A “Establish measurement objectives”, section III-B “Specify measures” and section III-C “Specify data collection and storage procedures”.
- **Yardstick_{MR} #10 (Evaluation of measurement process and results):** This yardstick is to investigate the evaluation methodologies (i.e. criteria), used to evaluate for the measurement process and results. No evidence is found in Case B_{MR} to support this yardstick.
- **Yardstick_{MR} #11 (Result communication):** This auditing yardstick investigates the existence of an appropriate scheduling and presentation procedures to discuss the measurement result with the project stakeholders. Modeling evidences are found in Case B_{MR} to support this yardstick. More clarification and details of this evidence are found in table V “Release tables” and table VI “Measurement tables”. These tables can help the project stakeholders to be informed about the measurement results periodically and to assess them to understand the measurement results.

Table 8.10 Measurement audit of case B_{MR}

Case B_{MR} (Mahnic, Zabkar)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{MR} # 1 (Identification of the measurement context)	Evidence exists	Textual evidence	Section III-A Table I	Page # 67
Yardstick_{MR} # 2 (Role assignment)	Evidence exists	Modeling evidence	Section V Table IV	Page # 71
Yardstick_{MR} #3 (Resources and budget constraints)	Evidence exists	Modeling evidence	Section V Table V and Table VI	Page # 71
Yardstick_{MR} #4 (Identification of measures)	Evidence exists	Textual evidence	Section III-A Table II Table II	Page # 67
Yardstick_{MR} #5 (Measurement data and models)	Evidence exists	Modeling and Textual evidences	Section III-D and Section III-E	Page # 68 69
Yardstick_{MR} #6 (Measurement results)	Evidence partially exists	Modeling evidence	Section IV Figure 2	Page # 70
Yardstick_{MR} #7 (Traceability of measurement data)	Evidence exists	Modeling evidence	Section V Table V and Table VI	Page # 71
Yardstick_{MR} #8 (Prioritization of measurement data)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #9 (Identification of measurement design)	Evidence exists	Textual evidence	Section III-A, section III-B and Section III- C	Page # 67 68
Yardstick_{MR} #10 (Evaluation of measurement process and results)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #11 (Result communication)	Evidence exist	Modeling evidence	Table and table VI	Page # 71

Case A_{MR} fully supports five measurement auditing yardsticks out of eleven. The fully supported measurement auditing yardsticks are yardstick_{MR}#1 (Identification of the measurement context), yardstick_{MR}#4 (Identification of measures), yardstick_{MR}#5 (Measurement data and models), yardstick_{MR} #8 (Prioritization of measurement data), and

yardstick_{MR}#9 (Identification of measurement design). Case A_{MR} partially supports yardstick_{MR}#2 (Role assignment). No auditing evidence was found to support yardstick_{MR}#3 (Resources and budget constraints), yardstick_{MR}#6 (Measurement results), yardstick_{MR}#7 (Traceability of measurement data), yardstick_{MR}#10 (Evaluation of measurement process and results), and yardstick_{MR}#11 (Result communication). Table 8.11 shows the auditing evidences type and location found in Case A_{MR}.

Case C_{MR} fully supports six measurement auditing yardsticks out of eleven. The fully supported measurement auditing yardsticks are yardstick_{MR}#1 (Identification of the measurement context), yardstick_{MR}#4 (Identification of measures), yardstick_{MR}#5 (Measurement data and models), yardstick_{MR}#6 (Measurement results), yardstick_{MR}#9 (Identification of measurement design), and yardstick_{MR}#10 (Evaluation of measurement process and results). Case C_{MR} partially supports yardstick_{MR}#2 (Role assignment). No auditing evidence was found to support yardstick_{MR}#3 (Resources and budget constraints), yardstick_{MR}#7 (Traceability of measurement data), yardstick_{MR}#8 (Prioritization of measurement data) and yardstick_{MR}#11 (Result communication). Table 8.12 shows the auditing evidences type and location found in Case C_{MR}.

Case D_{MR} fully supports six measurement auditing yardsticks out of eleven. The fully supported measurement auditing yardsticks are yardstick_{MR}#3 (Resources and budget constraints), yardstick_{MR}#4 (Identification of measures), yardstick_{MR}#5 (Measurement data and models), yardstick_{MR}#6 (Measurement results), yardstick_{MR}#8 (Prioritization of measurement data) and yardstick_{MR}#9 (Identification of measurement design). No auditing evidence was found to support yardstick_{MR}#1 (Identification of the measurement context), yardstick_{MR}#2 (Role assignment), yardstick_{MR}#7 (Traceability of measurement data), yardstick_{MR}#10 (Evaluation of measurement process and results), and yardstick_{MR}#11 (Result communication). Table 8.13 shows the auditing evidences type and location found in Case D_{MR}.

Table 8.11 Measurement audit of case A_{MR}

Case A _{MR} (Ktata, Lévesque)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{MR} # 1 (Identification of the measurement context)	Evidence exists	Textual evidence	Section 3.4 "Measurement program approach", 4.2 "Looking for the managing indicators needed"	page #103 page #104
Yardstick_{MR} # 2 (Role assignment)	Evidence partially exists	Textual evidence	Section 2.1" Introduction to agility and Scrum"	Page # 102
Yardstick_{MR} #3 (Resources and budget constraints)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #4 (Identification of measures)	Evidence exists	Textual evidence And modeling	Section 3.4 "Measurement program approach", 4.2 "Looking for the managing indicators needed"	page #103 page #104
Yardstick_{MR} #5 (Measurement data and models)	Evidence exists	Modeling and Textual evidences	Section 4.2 "Looking for the managing indicators needed", Table 1,2,3,4,5, and 6	page #104 page #105
Yardstick_{MR} #6 (Measurement results)	Evidence does not exist	Not applicable	Not applicable"	Not applicable
Yardstick_{MR} #7 (Traceability of measurement data)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #8 (Prioritization of measurement data)	Evidence Exists	Modeling and Textual evidences	Section 4.2 "Looking for the managing indicators needed", Table 1,2,3,4,5, and 6	page #104 page #105
Yardstick_{MR} #9 (Identification of measurement design)	Evidence exists	Textual evidence	Section 3.3 "GQM considerations"	Page # 103
Yardstick_{MR} #10 (Evaluation of measurement process and results)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #11 (Result communication)	Evidence does not exist	Not applicable	Not applicable	Not applicable

Table 8.12 Measurement audit of Case C_{MR}

Case C_{MR} (Gustafsson)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{MR} # 1 (Identification of the measurement context)	Evidence exists	Textual evidence and graphical evidence	Section III "Research methodology", figure 1 and 2	page #11
Yardstick_{MR} # 2 (Role assignment)	Evidence partially exists	Textual evidence	phase 1 and 2 of section IV "Results and analysis"	Page # 12 Page # 13
Yardstick_{MR} #3 (Resources and budget constraints)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #4 (Identification of measures)	Evidence exists	Textual evidence	phase 5 of section IV "Results and analysis"	page #17
Yardstick_{MR} #5 (Measurement data and models)	Evidence exists	Textual evidences	Section 5 "Data collection and analysis"	page #19
Yardstick_{MR} #6 (Measurement results)	Evidence exists	graphical evidence	Figure 7 and Figure 8	page #19
Yardstick_{MR} #7 (Traceability of measurement data)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #8 (Prioritization of measurement data)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #9 (Identification of measurement design)	Evidence exists	Textual evidence	Section III "research methodology"	Page # 11
Yardstick_{MR} #10 (Evaluation of measurement process and results)	Evidence Exists	Textual evidence	Phase 3 of section III "research methodology"	Page # 11
Yardstick_{MR} #11 (Result communication)	Evidence does not exist	Not applicable	Not applicable	Not applicable

Table 8.13 Measurement audit of Case D_{MR}

Case D_{MR} (Fehlmann)				
	Evidence Status	Evidence Type	Evidence Location	Page number
Yardstick_{MR} # 1 (Identification of the measurement context)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} # 2 (Role assignment)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #3 (Resources and budget constraints)	Evidence exists	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #4 (Identification of measures)	Evidence exists	Textual evidence	Section 3.3 "estimate effort needed for Implementing user stories"	page #3
Yardstick_{MR} #5 (Measurement data and models)	Evidence exists	Textual evidences and modeling evidences	Section 5 "blending six sigma with agile" and figure 1	page #5 page#6
Yardstick_{MR} #6 (Measurement results)	Evidence exists	Textual evidences and modeling evidences	Section 5 "detailed story items" and figure 2	page# 8 and page #9
Yardstick_{MR} #7 (Traceability of measurement data)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #8 (Prioritization of measurement data)	Evidence exists	Textual evidences	Section 5.1 "using QFD for user story prioritisation"	page #7
Yardstick_{MR} #9 (Identification of measurement design)	Evidence exists	Textual evidences	Section 4 "measuring non-functional requirements"	Page # 3
Yardstick_{MR} #10 (Evaluation of measurement process and results)	Evidence does not exist	Not applicable	Not applicable	Not applicable
Yardstick_{MR} #11 (Result communication)	Evidence exists	Textual and modeling evidences	Section 5.3 "using QFD as a communication tool" and figure 2.	Page #7 Page # 8

8.4.4 Auditing of measurement case studies for yardstick_{MR} #5 (Measurement data and models)

This yardstick investigates the existence of mechanisms for the data collection used during the measurement development. In this section the process of identifying the existence of evidences for the yardstick_{MR} #5 are clarified. The process focuses on providing links between the yardstick_{MR} #5 to its supporting type and location of evidences found in Case A_{MR}, Case C_{MR} and Case D_{MR}. Yardstick_{MR} #5 for Case B_{MR} has been analysed in the previous section. Table 8.14 describe the analysis results of yardstick_{MR} #5.

Case A_{MR} supports Yardstick_{MR} #5 with both modeling and textual evidences. Case A_{MR} developed “semi-structure interviews” based on GQM template for gathering and modeling the measurement data. The evidences are found in section 3.4 “measurement program approach” and section 4.2 “looking for the managing indicators needed”. The modeling evidences are found by using the GQM template for modeling the measurement data. This is also described in section 3.4 “measurement program approach” and section 4.2 “looking for the managing indicators needed”.

Case C_{MR} supports yardstick_{MR} #5 with textual and graphical evidences. In Case C_{MR} the data were collected by letting the developers fill out the required fields in an Excel sheet located on a shared server. The textual and graphical evidences are found in section 5 "Data collection and analysis", figure 7 "Sampling the root cause" and figure 8 "preventive actions that would have found the bug".

Case D_{MR} for textual and modeling evidences for yardstick_{MR} #5. In Case D_{MR} measurement data has been specified using the approach of COSMIC. Sequence diagrams have been created to provide common understanding among developers and sponsors. The textual and modeling evidences have been found in section 4 "measuring non-functional requirements" and Figure1 "Buglione-Trudel matrix".

Table 8.14 Audit based on Yardstick_{MR} #5

Yardstick_{MR} #5 (Measurement data and models)			
	Case A_{MR} (Ktata, Lévesque)	Case C_{MR} (Gustafsson)	Case D_{MR} (Fehlmann)
Evidence Status	Evidence exists	Evidence exists	Evidence exists
Evidence Type	modeling and textual evidences	textual and graphical evidences	modeling and textual evidences
Evidence Location	Section 3.4 Section 4.2	Section 5 Figure 7 Figure 8	Section 4 Figure1
Page number	Page # 103 Page #105	page # 19	Page #3 Page#6

8.4.3 Auditing summary of agile measurement case studies

Case A_{MR} (Ktata, Lévesque): Case A_{MR} supports the auditing criteria for measurement plans by identifying the context of the measurement program using the Goal Question Measurement approach (GQM). The GQM has been used to state explicit measurement goals and to analyze the measurement data. Role Assignment is not fully supported since there is no evidence on how the measurement team has been managed (i.e. Measurement user, Measurement analyst and/or Measurement librarian) but it has been noted that the product owner has assigned a responsibly for integrating the measurement activities with regular project activities. No evidence has been identified on how the resources and budget for the measurement team has been managed.

The auditing criteria for measurement development have been supported mainly by identifying a set of measures which has been derived from the GQM. The measures are next classified into team dynamic indicators, process and project related indicators, Customer related improvement indicators and internal quality indicators. The measurement data have been collected through face to face interviews with the project stakeholders. The

measurement data has been presented using decision tables. There is no evidence of presenting or manipulation for the measurement result but a cause-effect diagram has been proposed to trigger conversations that would result in a team sharing their concerns and issues and coming to a shared view. No evidence has been found on traceability of the measurement data.

The normal technology of measurement development is a normal design since the measurement method is mainly based on GQM. No evidence has been found on how the measurement results were evaluated, including the presentation procedures to the project stakeholders- See table 8.15.

Case B_{MR} (Mahnica, Zabkar): Case B_{MR} supports the auditing criteria for measurement plans by identifying the goal and context of the measurement program using the guidelines of the CMMI model. The measurement program context is set into: timely information on project performance, quality improvement, job satisfaction, and efficient impediments resolution and customer satisfaction. The role assignment of the team was managed using relational tables. These tables describe the relationships among projects, teams and employees. Each team has an ID and each employee as an ID and an employee description.

The dependencies between the measurement tasks were defined using release tables. Each release table identifies the release, task, and task status and task type. This practice is a partial support for auditing criteria for measurement plans.

The auditing criteria for measurement development have been supported mainly by identifying a set of measures based on the guidelines of CMMI model. Examples of the identified measures are: the number of errors reported by the user in a fixed period after release, the number of tasks completed during the iterations, the number of errors found during the iterations review meeting.

The measurement data was obtained during the planning meeting such that the following basic information was identified: the iteration length, the number of the team members,

percentage of each team member's engagement in the project, and costs of each team member's engineering hour.

Evidence has been found to support the traceability of measurement data by the existence of the following measurement tables: task measurement results, release measurement results and sprint measurement results. And each table is associated with Measure ID#, Date# and Measurement Result. No evidence has been found on how the measurement data was prioritized.

The normal technology of measurement development is a normal design since the measurement method is mainly based on CMMI model. No evidence has been found on how the measurement result was evaluated - See table 8.15.

Case C_{MR} (Gustafsson): Case C_{MR} supports the auditing criteria for measurement plans by identifying the goal and context of the measurement program using a set of identified key process indicator areas (KPIA). The context of Case C_{MR} is to improve the following organizational practices: product quality, product delivery, development cost, service level and product planning. The measurement user for Case C_{MR} was Bwin Games: a web game software company based in Sweden. No evidence has been found on how the measurement responsibilities were assigned to the measurement team. No evidence has been found on how the resources and budget of the measurement cost was managed.

The Auditing criteria for measurement development have been supported mainly by identifying a set of measures based on the interviews with the top management and based on the identified measurement context. Examples of the identified measures are: the financial measures, production measures, lean measure. No evidence has been found of traceability of the measurement data and no evidence has been found for management data prioritization.

No evidence has been found on how the measurement result was evaluated or the presentation procedures to the project stakeholders. Evidence of evaluation of measurement

results and communication has been found as a part of the design of the measurement model
- See table 8.15.

Case D_{MR} (Fehlmann): Case D_{MR} presents measurement methods for “making agile development processes lean and measurable”. The objective of Case D_{MR} was not to propose a measurement program for agile software process but to focus on a measurement improvement method for the agile software process. The objectives of Case D_{MR} make it less convenient to support for auditing criteria for measurement plans. The auditing criteria for measurement development have been supported mainly by identifying the data movements based on the guidelines of COSMIC model. The goal of the identified data movements is to estimate effort needed for implementing user stories. The measurement data was modeled using data movement diagrams and prioritized using QFD Quality Function Deployment (QFD). The measurement result was presented using "Buglione-Trudel Matrix". No evidence has been found for traceability of measurement data.

The normal technology of measurement development is a normal design since the measurement method is mainly based COSMIC and Six Sigma. No evidence has been found for evaluation of the measurement process and result- See table 8.15.

Table 8.15 consist of three main auditing criteria which are auditing criteria for measurement plan, auditing criteria for measurement development, and auditing criteria for measurement management. Table 8.15 illustrates the auditing results for the case studies based on proposed auditing model in chapter 7.

Table 8.15 Existence of evidence in the selected case studies

	Case A_{MR} (Ktata, Lévesque)	Case B_{MR} (Mahnic, Zabkar)	Case C_{MR} (Gustafsson)	Case D_{MR} (Fehlmann)
Auditing criteria for measurement plan				
Yardstick_{MR} # 1 (Identification of the measurement context)	Evidence exists	Evidence exists	Evidence exists	Evidence does not exist
Yardstick_{MR} # 2 (Role assignment)	Evidence partially exists	Evidence exists	Evidence partially exists	Evidence does not exist
Yardstick_{MR} #3 (Resources and budget constraints)	Evidence does not exist	Evidence exists	Evidence does not exist	Evidence does not exist
Auditing criteria for measurement development				
	Operational principles of measurement development			
Yardstick_{MR} #4 (Identification of measures)	Evidence exists	Evidence exists	Evidence exists	Evidence exists
Yardstick_{MR} #5 (Measurement data and models)	Evidence exists	Evidence exists	Evidence exists	Evidence exists
Yardstick_{MR} #6 (Measurement results)	Evidence does not exist	Evidence partially exists	Evidence exists	Evidence exists
	Normal technology of measurement development			
Yardstick_{MR} #7 (Traceability of measurement data)	Evidence does not exist	Evidence exists	Evidence does not exist	Evidence exists
Yardstick_{MR} #8 (Prioritization of measurement data)	Evidence exists	Evidence does not exist	Evidence does not exist	Evidence exists
	Normal configuration of measurement development			
Yardstick_{MR} #9 (Identification of measurement design)	Evidence exists	Evidence exists	Evidence exists	Evidence exists
Auditing criteria for measurement management				
Yardstick_{MR} #10 (Evaluation of measurement process and results)	Evidence does not exist	Evidence does not exist	Evidence exists	Evidence does not exist
Yardstick_{MR} #11 (Result communication)	Evidence does not exist	Evidence exists	Evidence does not exist	Evidence exists

8.5 Summary

Five different case studies have been audited based on the proposed model to investigate whether or not they conform to the ISO 9001 traceability requirements. The evidence gathered shows at least partial support for the requirements in each case study; however no case study has been demonstrated as supporting fully the auditing yardsticks.

Four different case studies have been audited based on the proposed model to investigate whether or not they conform to the ISO 9001 measurement requirements. The evidence gathered shows at least partial support for the requirements in each case study; however no case study has been demonstrated as supporting fully the auditing yardsticks.

CONCLUSION

The goal of this research project was to improve the agile-XP process in supporting the auditing requirements of ISO 9001, as well as to help agile software organizations in their effort to become ISO 9001 certified.

To achieve this goal, the following research objectives were specified:

- Identify gaps between agile-XP and ISO 9001, by highlighting the main strengths and weaknesses of agile-XP in handling the ISO 9001 requirements.
- Propose several sub processes to enhance the early planning activities of agile-XP according to ISO 9001 requirements.
- Design an auditing model that covers the measurement and traceability requirements of ISO 9001, and is capable of providing IS auditors with auditing evidence that the software projects developed using an agile-XP process have fulfilled the requirements of ISO 9001.
- Verify the applicability of the auditing model based on agile traceability and the agile measurement approaches.

The research objectives were achieved using several engineering models and frameworks: ISO 9001:2008, ISO 90003:2003, Vincenti's engineering design Vincenti (1990), SWEBOK guide Abran, Moore *et al.* (2004), ISO 12207:2008, CMMI for development, version 1.2, standards, guidelines and procedures for information system auditing, ISACA (2010), and the evaluation theory perspective of the Architecture Trade-off Analysis Method – ATAM López (2000). Defining the various concepts of the evaluation theory and Vincenti's engineering principles while developing the proposed auditing model helped us produce a rigorous and comprehensive research methodology.

ISO 12207 and CMMI, which integrate software engineering and systems engineering into product engineering best practices, were valuable tools for interpreting the ISO 9001 requirements from an engineering and software engineering perspective, and allowed us to

focus on proposing an auditing model for agile software processes (e.g. agile-XP) that is aligned with software engineering best practices.

Research Contributions

The research contributions of this study are classified into seven categories:

1. **Literature of the agile software process:** Analysis of several studies and surveys related to the topic of the agile software process and its implementation in the context of software organizations. This research also analyzed the characteristics of many agile processes and compared them based on key requirements for a software development project. This comparison can help project managers and software engineers select the agile process that best suits the requirements of their software projects.
2. **ISO 9001 and related auditing principles:** Analysis of several studies and surveys that report on the implementation of ISO 9001 in software organizations. This work also outlined the potential advantages of ISO 9001 certification for software organizations. The auditing principles for ISO 9001 certification were clarified.
3. **Analysis of agile-XP from the ISO 9001 and ISO 90003 perspectives:** This involves extracting the requirements related to the ISO 9001 product realization process and identifying the strengths and weaknesses of agile-XP in handling those requirements. The extraction of ISO 9001 requirements and ISO 90003 guidelines was based on ISO 12207 terminologies.
4. **Extension to agile-XP user stories:** Proposed extensions to the user story, based on four sub processes related to the CMMI-DEV model, are the following: 1) identification of the source of the user story; 2) categorization of the non functional requirements; 3) identification of the user story relationships; and 4) prioritization of the user stories. These sub processes are aligned with the agile-XP release planning phase, and enhance the ability of user stories to accumulate the information that is mandatory for achieving ISO 9001 certification. However, the usefulness of the proposed sub processes has not been validated based on industrial or controlled experiment.

5. **An audit model for ISO 9001 traceability requirements:** Development of an auditing model for ISO 9001 traceability requirements that is applicable in agile software process environments (e.g. agile-XP). The design of the auditing model is based on evaluation theory, and includes the use of several auditing “yardsticks” derived from the principles of engineering design, the SWEBOK Guide, and the CMMI-DEV guidelines for requirement management and traceability. The objective of this model is help software organizations in their effort to achieve ISO 9001 certification and help software auditors extract auditing evidence that demonstrates the ability of a software organization to implement the ISO 9001 traceability requirements.
6. **An audit model for ISO 9001 measurement requirements:** Development of an auditing model for ISO 9001 measurement requirements that is applicable in agile software process environments. Several engineering models have been analyzed to develop auditing criteria that support the ISO 9001 measurement requirements, such as Vincenti’s engineering design, evaluation theory, CMMI-DEV, and the SWEBOK. The model consists of three major categories of auditing criteria: measurement plan criteria, measurement development criteria and measurement management criteria. Each auditing criterion consists of several auditing yardsticks which focus on the evidence that can be extracted to demonstrate process conformity to ISO 9001 measurement requirements.
7. **Audit of case studies:** Demonstration for the applicability of the proposed auditing models based on five case studies from the literature related to agile software traceability and four case studies from the literature related to agile software measurement. This includes the development of criteria to search and select papers proposing a methodology, technique, or framework to enhance the agile traceability and agile measurement process. For all the cases studied in this research, the auditing evidence was gathered using the information system audit procedure described by the standards, guidelines, and procedures for information system auditing ISACA, (2010).

A number of outcomes of this thesis have been published/submitted in the following journals or conferences and workshops:

- ✓ Malik Qasaimeh, and Alain Abran, "Extending Extreme Programming User Stories to Meet ISO 9001 Formality Requirements," *Journal of Software Engineering and Applications*, vol. 4, no.11, pp. 626-638, 2011.
- ✓ Malik Qasaimeh, Alain Abran "An Audit Model for ISO 9001 Traceability Requirements in Agile (XP) Environments," Submitted to the *Journal of Software*, (JSW, ISSN 1796-217) (Submitted-2012).
- ✓ Malik Qasaimeh, Alain Abran, "Investigation of the Capability of XP to Support The Requirements of ISO 9001 Software Process Certification," 8th ACIS International Conference on Software Engineering Research, Management and Applications, pp. 239-247, Montreal, 26-29 May, 2010.
- ✓ Malik Qasaimeh, Alain Abran, "Agile process Support for Certification Requirement: The case of XP and ISO 9001," (invited talk) at the International Summer Research Symposium on Software Engineering Measurement, Montreal, Canada, August 2010.
- ✓ Malik Qasaimeh, Hossein Mehrfard, Abdelwahab Hamou-Lhadj, "Comparing Agile Software Processes Based on the Software Development Project Requirements," *IEEE International Conference on Innovation in Software Engineering*, Vienna, 10-12 December, 2008.

Future work

The research presented in this thesis can lead to further work to improve our understanding of the ISO certification process and our experience of the process in the context of agile software organizations. In this thesis, several engineering models and frameworks have been investigated to enhance the early practices of agile-XP to accommodate important information for ISO 9001 auditors. For instance, auditing models for ISO 9001 traceability and measurement have been developed to help ISO 9001 auditors find auditing evidence in the context of agile software processes.

Accordingly, future work can be pursued based on the results and the methodologies used in this thesis:

- **Engineering perspective of agile software processes:** Agile software processes are relatively new software process life cycle models in the domain of software engineering. The agile software process was introduced in February 2001, after a group of software developers in Snowbird (Utah) met to discuss lightweight development methods for software construction. The result of this meeting was the publication of the “Manifesto for Agile Software Development” to define the approach, now known as the agile software process. The literature provides evidence of the success of agile software processes (e.g. agile-XP) in the development of small and medium-sized software projects, but little is known about how the activities of the agile software process are aligned with engineering principles. Possible topics for future research work include:
 - ✓ Identification of the engineering principles for software process development in the context of the SWEBOK Guide Abran, Moore *et al.* (2004).
 - ✓ Identification of the engineering principles for product development in the context of Vincenti’s engineering design Vincenti (1990).
 - ✓ Development of engineering criteria to evaluate the software process life cycle within the software engineering domain.
 - ✓ Identification of the strengths and/or weaknesses of agile software process to support the engineering criteria.
 - ✓ Revisions with additional experts could be conducted to further strengthen the development of the engineering criteria and to sustain the identification of the strength and/or weaknesses of agile software processes.
- **Domain extension:** The auditing models developed in this research focus on providing auditing criteria for ISO 9001, clause 7, "Product realization," and clause 8, "Measurement, analysis and improvement." Clauses 7 and 8 have been found to have a direct impact on software process development. However, this research has not directly considered other ISO 9001 clauses that impact the organization’s business processes, but rather focuses on the certification of the organization’s processes for developing software system in the agile environment. Possible topics for future research work include:

- ✓ Analysis and extraction of ISO 9001 requirements that impact the organization's business process.
- ✓ Extending the auditing models using evaluation theory and other software engineering models, such as CMMI, to cover the business processes of agile software organizations. This extension could allow ISO 9001 auditors to extract auditing evidence to assess the conformity of business processes in agile software organizations.
- ✓ **Alignment with ISO 29110-3:** ISO 29110-3:2009 "Software Engineering – Lifecycle Profiles for Very Small Entities (VSE) – Part 3: Assessment guide" identifies a set of guidelines for VSE to address the requirements for performing assessment in VSE. The ISO defines a VSE as an entity (enterprise, organization, department, or project) having up to 25 employees. ISO 29110-3:2009 states that VSE are often cut off from some economic activities because of the difficulty of relating ISO standards to their business needs and of justifying the application of the standards to their business practices. ISO 29110-3:2009 also states that ISO standards do not address the needs of VSE, and that conforming to these standards is difficult. Accordingly, the auditing models proposed in this research work can be aligned with the guidelines of ISO 29110-3, which is to provide the agile-based VSE with a tool to assess their conformity to ISO 9001.
- ✓ **Validation subjectivity:** Forer (1949) highlights the problem of subjective validation in scientific research as that of a bias according to which researchers will consider the information correct if it has personal meaning or significant to them. The validation process for the proposed auditing models has been carried out to simulate an auditing process in software organizations where the auditors develop the audit conclusions based on the existence of evidence that supports the scope and objectives of the audit. Auditors should use their expertise and understanding of the organization's processes to judge whether or not the auditing evidence is sufficient. "The evidence can be considered sufficient if it supports all the material questions the audit objective and scope" ISACA (2010). From a scientific research perspective, this may lead to subjectivity in the

validation process. To reduce the level of subjectivity, the following further research steps could be conducted with more time and resources:

- ✓ **Validation to the extended used story:** This thesis has proposed four sup processes to be aligned with the early practise of agile-XP. A controlled experiment based on an industrial practise in agile organizations is recommended to validate the usefulness of the proposed sub process.
- ✓ **Validity threat:** The case studies that have been selected to evaluate the proposed auditing models have been carried out by the same researchers who designed the auditing models. This might introduce biases in the process for the evaluation of the proposed auditing models. To cope with this problem, the validation of the auditing models could be performed by external experts to assess the usability of the proposed auditing models in the area of ISO 9001 auditing and agile software processes. In this situation, the same case studies can be audited by bodies external to the research team.
- ✓ **Usability of the auditing models:** To evaluate the usability of the proposed auditing models in the context of agile software organization that wish to become an ISO 9001 certified. The auditing models could be validated by designing experiments for software products developed using agile software processes in an organization aiming to become ISO 9001 certified. This will also allow for both agile software developers and IS-auditors to assess the usability of the proposed auditing models.

BIBLIOGRAPHY

- Abrahamson, P., O. Salo, J. Ronkainen, J. Warsta. (2002). *"Agile Software Development Methods, Review and Analysis"*, VTT Publications 478. Espo, Finland, P. 107.
- Abran, A., J. Moore. (2004). *"Guide to the software engineering body of knowledge"*, IEEE Computer Society Press, pp. 1-228.
- Abran A., K.T. Al-Sarayreh, J. J. Cuadrado-Gallego. (2010). *"Measurement Model of Software Requirements Derived from System Maintainability Requirements"*, Twenty-Second International Conference on Software Engineering and Knowledge Engineering, San Francisco, pp.153-158.
- Abran, A., R.E. Al-Qutaish, J. J. Cuadrado-Gallego. (2004). *"Analysis of the ISO 9126 on Software Product Quality Evaluation from the Metrology and ISO 15939 Perspectives"*, WSEAS Transactions on Computers, World Scientific and Engineering Academy and Society, Athens, Greece, 5(11), pp.2778-2786.
- Alain A., R. E. Al-Qutaish, J. Desharnais, N. Habra. (2005). *"An Information Model for Software Quality Measurement with ISO Standards"*, International Conference on Software Development, Reykjavik, Iceland, pp. 104-116.
- Alali A.I. and Issa A.A. (2011). *"Towards Well Documented and Designed Agile Software Development"*, Proceeding of Modeling and Simulation, Calgary, AB, Canada.
- Alegria J. and Bastarrica M. (2006). *"Implementing CMMI Using a Combination of Agile Methods"*. CLEI Electric Journal, 7(1), pp. 20-35.
- Al-Qutaish, R. (2007). *"SPQ^{MM}: A Software Product Quality Maturity Model Using ISO/IEEE Standards, Metrology, and Sigma Concepts"*, PhD thesis, École de Technologie Supérieure, Université du Québec, Montréal, Canada.
- Al-Sarayreh K.T and Abran A. (2010). *"A Generic Model for the Specification of Software Interface Requirements and Measurement of their Functional Size"*, Eighth ACIS International Conference on Software Engineering Research Management and Applications, Montreal, Canada, 2010, pp. 217-222.
- Ambler S. (2006). *"Agile survey results summary"*, <http://trailridgeconsulting.com/surveys/agile-adoption-rates-2006>. Html, last access 24/02/2010.
- Angelina, E. and G. Juan (2011). *"Study to support agile methods more effectively through traceability"*, Computer Science Innovations in Systems and Software Engineering, 7(1), pp. 53-69.

- Baskerville, R., L. Levine, J. Pries-Heje, B. Ramesh, S. Slaughter. (2001). "*How Internet Software Companies Negotiate Quality*", *IEEE Computer*, 34(5), pp. 51-57.
- Bebensee, Th., I. Weerd, S. Brinkkemper.(2010). "*Binary Priority List for Prioritizing Software Requirements*", 6th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 67-78.
- Beck, K. (1999). "*Extreme programming explained: Embrace change*", Addison Wesley Professional, 1st edition, pp. 1- 224.
- Béгноche, L., A. Abran, L. Buglione.(2007). "*A Measurement Approach Integrating ISO 15939, CMMI and ISBSG*", 4th Software Measurement European Forum", Rome, Italy, pp. 26-34.
- Berander, P. and Andrews, A. (2005). "*Requirements Prioritization in Engineering and Managing Software Requirements*", Springer Verlag, Berlin, Germany, pp. 69-94.
- Boehm, B. and Turner, R. (2003). "*Using Risk to Balance Agile and Plan-Driven Methods. IEEE Computer*", 36(6), pp.57-66.
- Canfora, V. and A. Cimitile. (2005). "*Empirical Study on The Productivity of the Pair Programming*". Lecture Notes in Computer Science. Springer-Verlag, Berlin, pp. 92-99.
- Cao, D. B. (2006). "*An Empirical Investigation of Critical Success Factors in Agile Software Development Projects*". PhD thesis, Capella University, USA.
- Misra, C.S., V. Kumar, U. Kuma.(2009). "*Identifying Some Important Success Factors In Adopting Agile Software Development Practices*", *Journal of Systems and Software*, 82(11), P.1869-1890.
- Chambers, A. and G. Rand, (2010). "*Operational Auditing: Auditing Business and IT Processes*", Wiley, 2nd Edition, 2010.
- Chorafas, N. (2008). "*IT auditing and Sarbanes-Oxley Compliance: Key Strategies for Business Improvement*", Auerbach Publications, 1st Edition.
- Chow, T. and D. Cao. (2008). "*A survey study of critical success factors in agile software projects*", *Journal of Systems and Software*, 81(6), pp. 961-971.
- Cockburn, A. and L. Williams. (2001). "*The Costs and Benefits of Pair Programming in Extreme Programming Examined*", Addison Wesley, Boston, MA, pp. 223–243.

- Cohn, M. (2005). *"Agile Estimating and Planning"*, Prentice Hall PTR, Upper Saddle River, NJ. 1st Edition. pp.398.
- Cohn, M., and D. Ford. (2003). *"Introducing an Agile Process to an Organization"*, IEEE Computer, 36(6), pp.74-78.
- Conboy, K., and B. Fitzgerald. (2004). *"Toward a Conceptual Framework of Agile Methods: A study of Agility in Different Disciplines"*, Proceedings of the 2004 ACM workshop on Interdisciplinary Software Engineering Research, Newport Beach, USA, pp. 37-44.
- COSMIC (2011). *"Guideline for the use of COSMIC FSM to manage agile projects, V 1.0"*, The Common Software Measurement International Consortium. Last accessed 22/5/2012: http://www.cosmicon.com/portal/public/COSMIC_Agile_Projects_Guideline_v10.pdf.
- Diaz J., J. Pérez, PP. Alarcón, J. Garbajosa. (2011). *"Agile Product Line Engineering—A Systematic Literature Review"*, Journal of Software Maintenance and Evolution: Research and Practice, 41(8), pp. 921-941.
- Dolores, R. and R. Fujii. (1989). *"Software Verification and Validation: An Overview"*, IEEE Software, 6(3), pp.10-17.
- Dyba, T. and T. Dingsøy. (2008). *"Empirical Studies of Agile Software Development: A Systematic Review"*, Information and Software Technology, 50 (9), pp. 833-859.
- Ellis, T. and Y. Levy. (2008). *"Framework of Problem-Based Research: A Guide for Novice Researchers on the Development of a Research-Worthy Problem"*, International Journal of an Emerging Transdiscipline, Volume 11, pp. 17-33.
- Espinoza, A. and J. Garbajosa. (2011). *"Study to Support Agile Methods More Effectively through Traceability"*, Computer Science Innovations in Systems and Software Engineering, 7(1), pp. 53-69.
- Fehlmann, TH. (2011). *"Six Sigma for Agile Teams"*, International Conference on the Modern Art of Software, Zurich, Switzerland, pp. 1-11.
- Ferreira, A., G. Santos , G. Santos, R.Cerqueira, M. Montoni, A. Barreto, A. Oliveira, S. Barreto, A. Rocha. (2007). *"Applying ISO 9001:2000, MPS, BR and CMMI to Achieve Software Process Maturity: BL informatica's pathway"*. 29th Int. Conference on Software Engineering, Minneapolis, USA, pp. 642-651.
- Forer, B.R. (1949). *"The Fallacy of Personal Validation: A classroom Demonstration of Gullibility"*, Journal of Abnormal Psychology, 44, pp. 118-121.

- Forman, E. and Selly, M. A. (1996). "*Decision by Objectives*", 1996, George Washington University, Washington, DC, USA.
- Frank, F. and O. Karam. (2006). "*Essentials of Software Engineering*", Jones and Bartlett Publishers, Canada, pp. 1-320.
- Fricker, S. and P. Grünbacher. (2008). "*Negotiation Constellations: Method Selection Framework for Requirements Negotiation*", Intl. Working Conference on Requirements Engineering: Foundation for Software Quality, Montpellier, France.
- Fuller, G. K. (2006). "*Antecedents and Consequences of Certification of Software Engineering Processes*", PhD Thesis, University of British Columbia, Canada.
- Ghanam, .Y and F. Maurer. (2009). "*Extreme Product Line Engineering: Managing Variability & Traceability via Executable Specifications*", Agile Conference, Chicago, IL, pp. 41 - 48.
- Ghazarian, A. (2008). "*Traceability Patterns: An Approach to Requirement Component Traceability in Agile Software Development*", 8th WSEAS International Conference on Applied Computer Science, Venice, Italy, pp. 236-241.
- Glinz, M. and Wieringa R. (2007). "*Stakeholders in Requirements Engineering*", IEEE Software, 24(2), 2007, pp.18-21.
- Grandzol, J. (2005). "*Improving the Faculty Selection Process in Higher Education: A Case for the Analytic Hierarchy*", Process Association for Institutional Research, 6(1), pp. 1-13.
- Grenning, J. (April 2002). "*Planning Poker*", Renaissance Software Consulting, Retrieved 2008-08-31.
- Griesemer, J. (1999). "*A Field Study of the Impact of ISO 9001 on Software Development in the United States*", PhD thesis, Pace University, United State of America.
- Hass A. M., J. Johansen, J. Pries-Heje. (1998). "*Does ISO 9001 increase software development maturity*", 24th EUROMICRO Conference, Vasteras, Sweden, pp. 860-866.
- Hazzan, O. and Y. Dubinsky. (2008). "*Agile Software Engineering*", Springer, 1st Edition, P.303.
- Highsmith, J. (2000). "*Adaptive software development: A collaborative approach to managing complex systems*", Dorset House, New York, 2000, pp. 1-392.

- IEEE (1990). "*Standard Glossary of Software Engineering Terminology*", IEEE Std. 610.12 1990, New York (NY), USA: the Institute of Electrical and Electronics Engineers, pp. 1-83.
- ISA (2009). "*International Standard on Auditing 500-Audit Evidence*", International Federation of Accountants, New York, USA.
- ISACA (2008). "*Control Objectives for Information and related Technology*", Information Systems Audit and Control Association, Rolling Meadows, IL, USA.
- ISACA (2010). "*IT Standards, Guidelines, and Tools and Techniques for Audit and Assurance and Control Professionals*", Information Systems Audit and Control Association, Rolling Meadows, IL, USA.
- ISO-19761 (2011). "*Software Engineering - COSMIC v 3.0 - A Functional Size Measurement Method*", International Organization for Standardization, Geneva (Switzerland).
- ISO-12207 (2008). "*Systems and Software Engineering – Software Life Cycles Processes*", International Organization for Standardization/International Electrotechnical Commission, Geneva (Switzerland).
- ISO-15288 (2008). "*Systems and software engineering – System life cycles processes*", International Organization for Standardization/International Electrotechnical Commission, Geneva (Switzerland).
- ISO-9126 (2004). "*Software Engineering - Product Quality - Part 1: Quality Model 9126-1*", International Organization for Standardization, Geneva (Switzerland).
- ISO 90003 (2004). "*Software Engineering - Guidelines for the Application of ISO 9001:2000 to Computer Software. ISO/IEC 90003*", International Organization for Standardization, Geneva, Switzerland, pp. 1-54.
- ISO 15939 (2002). "*Software Engineering - Software Measurement Process*". ISO/IEC 15939, International Organization for Standardization, Geneva, Switzerland, pp.1-37.
- Jalali, S. and C. Wohlin. (2011). "*Global Software Engineering and Agile Practices: a Systematic Review*", Journal of Software Maintenance and Evolution: Research and Practice, DOI: 10.1002/smr.561.
- Josyleuda M., M. Oliveira, A. Belchior. (2005). "*Measurement Process: A Mapping Among CMMI-SW, ISO/IEC 15939, IEEE Std 1061, Six Sigma and PSM*", International Conference on Service Systems and Service Management, Troyes, France, pp. 810-815.

- Karlesky, M. and M. Vander. (2008). "*Agile Project Management (or Burning your Gantt Charts)*", Embedded Systems Conference Boston (Boston, Massachusetts), P. 247-267.
- Karlsson, L., B. Regnell, P. Berander, C. Wohlin. (2008). "*Requirements Prioritization: An Experiment on Exhaustive Pair-Wise Comparison versus Planning Game Partitioning*", Empirical Assessment in Software Engineering Conference, Keele, UK, 2008, P. 122-131.
- Kitchenham, B. (2007). "*Guidelines for Performing Systematic Literature Reviews in Software Engineering*", Joint Technical Report, Keele University, EBSE-2007-01, July 2007.
- Kevin, R. (2003). "*ISO 9001:2000 A Practical Quality Manual Explained*", ASQ Quality Press, USA, pp. 1-296.
- Kowalczykiewicz, K. and D. Weiss. (2002). "*Traceability: Taming Uncontrolled Change in Software Development*", 4th National Software Engineering Conference, Tarnowo Podgorne, Poland.
- Ktata, O and G. Lévesqu. (2010). "*Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?*", Proceedings of the Third C* Conference on Computer Science and Software Engineering, ACM, New York, NY, USA, pp.101-107.
- Lee, C., L. Guadagno, X. Jia. (2003). "*An Agile Approach to Capturing Requirements Traceability*", 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, Canada, pp.104-110.
- Lehtola, L. and M. Kauppinen. (2004). "*Requirements Prioritization Challenges in Practice*", Springer-Verlag, Berlin Heidelberg, pp.497-508.
- Lindstrom, L. and R. Jeffries. (2004). "*Extreme Programming and Agile Software Development Methodologies*", Information Systems Management, pp.41-52.
- Lopez, M. (2000). "*An Evaluation Theory Perspective of the Architecture Tradeoff Analysis Method (ATAM)*", CMU/SEI-20Q0-TR-012, Pittsburgh, PA.
- Lopez, M. (2003). "*Application of an Evaluation Framework for Analyzing the Architecture Tradeoff Analysis Method*", Journal of Systems and Software, 68(3), pp.233-241.
- Mahnic, V and N. Zabkar. (2007). "*Introducing CMMI Measurement and Analysis Practices into Scrum-Based Software Development Process*", International Journal of Mathematics and Computers in Simulation, 1(1), pp. 65-72.

- Makdee, B. and P. Praneetpolgrang. (2005). "*Roadmap in Development of Quality Model for Thai Software*". 3rd International Conference on Information and Communications Technology, Egypt, Cairo, pp.829-836.
- Maurer, F. and S. Martel. (2002). "*Extreme programming: Rapid Development for Web-based Application*", IEEE Internet Computing, 6(1), pp. 86-90.
- Meridji, K.(2010). "*Analysis of Software Engineering Principles from an Engineering Perspective*", Ph.D. dissertation, École de technologie supérieure, Montréal (Canada), 2010.
- Nerur S., R. Mahapatra, G. Mangalaraj. (2005). "*Challenges of migrating to agile methodologies*". Communications of the ACM, 48(5), P. 72-78.
- Qasaimeh, M. and A. Abran. (2011). "*Extending Extreme Programming User Stories to Meet ISO 9001 Formality Requirements*", Journal of Software Engineering and Applications, Vol.4, No.11, pp.626-638.
- Qasaimeh, M. and A. Abran. (2010). "*Investigation of the capability of XP to support the requirements of ISO 9001 software process certification*", Eighth ACIS International Conference on Software Engineering Research Management and Applications, Montreal, Canada, pp. 239-247.
- Ramesh, B. and M. Jarke. (2001). "*Towards Reference Models for Requirements Traceability*", IEEE Transactions on Software Engineering, 27(1), pp. 58-93.
- Rumpe, B. and A. Schröder. (2002). "*Quantitative Survey on Extreme Programming Projects*". Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, Alghero, Italy, pp. 95-100.
- Salo, O. and P. Abrahamsson. (2008). "*Agile Methods in European Embedded Software development Organizations: A Survey on the Actual Use and Usefulness of Extreme Programming and Scrum*", Institution of Engineering and Technology Software, 2(1), pp. 58-64.
- Schindler, C. (2008). "*Agile Software Development Methods and Practices in Austrian it Industry Results of an Empirical Study*", International Conferences on Computational Intelligence for Modeling, Control and Automation, Vienna, Austria, pp. 321-326.
- Statz, J. (2005), "*Measurement for process improvement v1.0*", Technical Report, Practical Software and Systems Measurement Center, New Jersey, USA.
- Stelzer, D., W. Melli, et al. (1996). "*Software Process Improvement via ISO 9000: Results of Two Surveys Among European Software Houses*". Proceedings of the 29th Hawaii International Conference on System Sciences, Maui, Hawaii, pp. 704-710.

- Tracy, H., B. Sarah, J. Verner, D. Wilson. (2008). "*The Impact of Staff Turnover on Software Projects: The Importance of Understanding What Makes Software Practitioners Tick*", ACM conference on Computer personnel doctoral consortium and research, New York, NY, USA, pp. 30-39.
- Turk, D., R. France, B. Rumpe. (2004). "*Assumptions Underlying Agile Software-Development Processes*". Journal of Database Management, 16(4), pp. 62-87.
- Verbo E., I. Caballero, R. Perez, C. Calero, M. Piattini. (2009). "*MEPLAMECAL: A Methodology Based on ISO/IEC 15939 to Elaborate Data Quality Measurement Plans*", IEEE Latin America Transactions, 7(3), pp.361-368.
- Vijayasarathy, L. and Turk D. (2008). "*Agile Software Development: A Survey of Early Adopters*". Journal of Information Technology Management, 19(2), pp. 1-8.
- Vincenti W. G. (1990), "*What Engineers Know and How They Know It*", The John Hopkins University Press, Baltimore, London, pp.336.
- Vitoria, D. (2004). "*Aligning XP with ISO 9001:2000-TickIT guide 5.0*", Master Thesis Software Engineering", Blekinge Institute of Technology, Sweden.
- Vriens, C.H. (2003). "*Certifying for CMM level 2 and ISO9001 with XP@Scrum*", Conference on Agile Development, IEEE Computer Society, Washington DC, USA, pp. 120-124.
- Wang, X., M. Lane , K. Conboy, M. Pikkarainen. (2009). "*Where agile research goes: starting from a 7-year retrospective (report on agile research workshop at XP2009)*", ACM SIGSOFT Software Engineering Notes, 34(5), pp. 28-30.
- Wieggers, K. (2002). "*Peer Reviews in Software: A Practical Guide*", Johns Hopkins University Press, Boston, pp. 1-256.
- Wright, G. (2003). "*Achieving ISO 9001 Certification for an XP Company*", Lecture Notes in Computer Science, Extreme programming and Agile Methods, agile universe, New Orleans, pp. 43-50.
- Yahaya, J., A. Deraman, F. Baharom, A. Hamdan. (2009). "*Software Certification from Process and Product Perspectives*". International Journal of Computer Science and Network Security, 9(3), pp. 222-231.
- Yang, H.-G and B. Vandenbosch. (1998). "*Visibility as the Basis of a Framework for Identifying Strategic Information Systems*", Journal of Information Technology Management, 9(2), pp. 31-42.

Zarour, M. (2009). “*Methods to Evaluate Lightweight Software Process Assessment Methods Based on Evaluation Theory and Engineering Design Principles*”, PhD thesis, École de Technologie Supérieure, Université du Québec, Montréal, Canada.

Zelkowitz, M. V. and D.Wallace. (1997). “*Eexperimental validation in software technology*”, Information and Software Technology, 39(11), pp.735-744.