Design and Assessment for Agile Auditing Model: The Case of ISO 9001 Traceability Requirements

Malik Qasaimeh and Alain Abran

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. More specifically, these organizations need to demonstrate 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. Agile software organizations which want to be ISO 9001 certified must therefore provide evidence of ISO 9001 conformity, and they need to develop their own procedures, tools, and methodologies to do so.**

This paper proposes an auditing model for ISO 9001 traceability requirements that is applicable in agile (XP) environments. The design of this auditing model is based on the 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 requirements management and traceability. This auditing model has been assessed based on case studies selected from the literature.

Index Terms—**Agile software certification, extreme programming, software process improvement, ISO 9001.**

I. Introduction

Implementing ISO 9001 in software organizations 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 have to develop myriad tools and techniques to demonstrate that their software processes are in conformity with the quality standard. A strategy is to set up 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 certification team must also assess the development teams to demonstrate that the software products are being developed according to ISO 9001 requirements. In essence, this means providing documented evidence that clarifies, for instance, how and when a particular design decision has been implemented. The collection of evidence constitutes a very important foundation on which the information system (IS) auditors base their audit results and conclusion. The ISO 9001 certification requirements are not technology related but are business requirements: there is ample evidence that many small organizations have achieved ISO 9001 certification and reap business benefits from such certification [1], [2].

This paper addresses this specific business issue of requirements for ISO 9001 in contexts where the software teams develop software products in an agile mode. More specifically, this paper looks at what agile teams must put in place to meet this business need.

The literature reports that some authors have initiated research work to address this business need. For instance, Vitoria [3] has studied the ISO 9001 and TickIT standards and analyzed how they have 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 agile-XP project, 24% could be partially applied, 20% could be applied in full, while 23% were not relevant to the scope of the projects.

Wright [4] describes a successful certification evidence for an agile-XP organization. This author describes how the organization managed the large team through the practice of agile-XP and highlights the tools used to support the project team to handle the ISO 9001 requirements: this author's focus is only on some selected ISO 9001 requirements and then highlights their corresponding XP support activities.

Extreme programming (agile-XP) has been selected in the research reported here for improvement as a candidate agile process. This selection was based on the literature indicating with different case studies from both academia and industry a higher adoption of agile-XP over other agile software processes [5], [6].

The traceability of the user requirements during the development process is among the important auditing challenges reported in the agile literature [7], [8]. 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."

In agile development, verifying that the requirements have been implemented, designed, and tested in the final product depends mainly on lightweight artifacts, such as test cases and user accepted tests, without documented evidence 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 [9] a manager cannot track progress in agile projects in the same way as in plan-driven projects, where a manager simply asks whether or not the necessary documents have been

Manuscript received March 23, 2014; revised August 4, 2014.

Malik Qasaimeh is with the Princess Sumaya University for Technology, Jordan (e-mail: m.qasaimeh@psut.edu.jo).

Alain Abran is with University of Québec, Canada (e-mail: alain.abran@etsmtl.ca).

produced.

This paper 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 with 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. Our model is designed from an engineering perspective: it is based on evaluation theory and on the investigation of the principles of engineering design [10]-[12]. Several best practice software engineering models such as CMMI-DEV and SWEBOK will be used to design the traceability auditing yardsticks. Finally, This paper assess the applicability of the proposed auditing model based on five case studies by extracting auditing evidences that support the conformity of ISO 9001 traceability requirements and extends the evaluation that has been proposed in [13].

This paper is organized as follows. Section II presents the concepts of auditing and certification in agile environments. Section III presents an analysis of traceability requirements in ISO 9001 and their potential advantages in software organizations. Section IV presents the methodology of auditing model design. Section V presents the formulation of the auditing criteria and the yardsticks. Section VI presents a case study for each of five agile-XP traceability approaches, and discusses the auditing evidence collected for software process traceability. Finally, Section VII presents the conclusion of the paper.

II. The Concept of Auditing and Certification in Agile Environment

Auditing consists of 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 . The activities may be carried out at various levels, such as: organization, system, process, project, or product.

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 objectives. 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.

Developers in agile environment can adopt agile modeling (AM) for the modeling and documentation for the software development processes: agile modeling (AM) is a collection of practices, guided by values and principles for application in a day to day basis. AM include practices such as: active stakeholder participation, group work to create suitable models, verification, iterative modeling, parallel model creation, application of standards and documentation improvement. Agile modeling has some common values with existing agile processes, such as XP and SCRUM, like communication with team members, simplicity, and feedback. Ambler in [14] mentions that " What makes AM a catalyst for improvement aren't the modeling techniques themselves—such as use case models, class models, data models, or user interface models—but how to apply them". However, agile modeling does not come with detailed procedures on how to create a software documentation process; rather, agile modeling is closer to an overall high level understanding of the whole system.

Software development-related documents constitute valuable audit evidence IS auditors. However, this is not the only type of evidence that can be obtained by the auditors: the IT Standards, Guidelines, and Tools, and the Techniques for Audit and Assurance and Control Professionals [15] 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.

III. Traceability Requirements in ISO 9001

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."

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.

In terms of the relationships between software process improvement and traceability techniques, the SWEBOK Guide [11] 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.

IV. Methodology

In this section, we present our methodology for the design of an audit model for software process traceability, focusing on ISO 9001 and the agile software processes. The methodology for this design is based on the work of [16]: 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 [16] and [17]. The design of our audit model considers the steps of an evaluation procedure as described by [16]:

- 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.

Fig. 1 presents the main process for designing an audit model for IS0 9001 traceability.

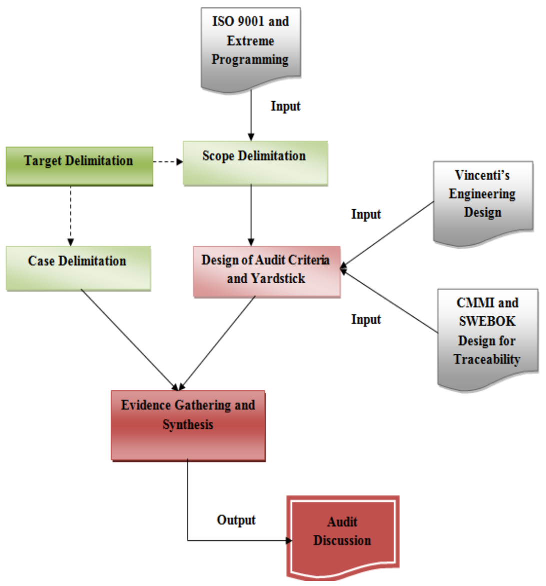

ISO 9001 and Extreme Programming
Input
Target Delimitation
Scope Delimitation
Vincenti's Engineering Design
Input
Case Delimitation
Design of Audit Criteria and Yardstick
Input
CMMI and SWEBOK Design for Traceability
Evidence Gathering and Synthesis
Output
Audit Discussion

Fig. 1. Design methodology for the audit model.

V. Design of the Auditing Model

A. Scope Delimitation

For agile software processes (e.g. agile-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 paper is to design an auditing model for traceability requirements 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. It can also be useful for auditing traditional software processes.

B. Design of the Audit Criteria and Yardsticks

As stated by [16], criteria can be elicited either using an obligatory 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 work here, the obligatory 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 [10].

The main auditing criteria and yardsticks are presented below:

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 [10], [11].

a) Design of the traceability method for agile

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.

The following are the audit yardsticks for these criteria:

- 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 [10]. 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 iterations, 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 iteration of the process life cycle; for example, the artifacts produced during the requirements elicitation process (e.g. the user stories in agile-XP).

2) *Management criteria*

The audit criteria presented below 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.
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 3, 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.

VI. Model Assessment

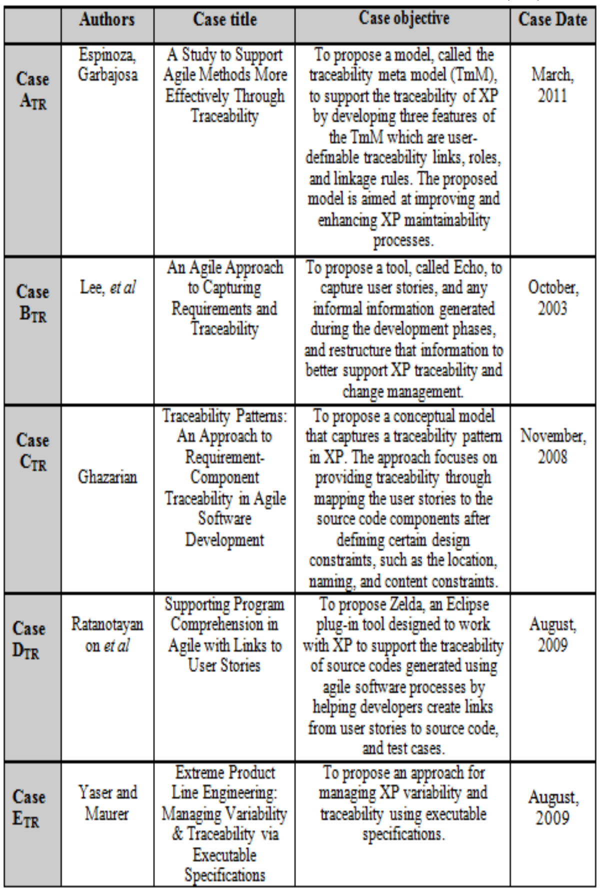
A. *Case Studies Setting*

The case studies presented in this paper are based on the proposals found in the literature for the traceability of agile software processes. The main limitations for assessing the proposed model 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 9001certification.
- **Time and budget constraint**: The assessment 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).

Five case studies have been selected from the literature to be compatible with the scope defined of the auditing model, seeing Table I.

TABLE I: Selected Case Studies for the Agile (XP)



	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, *et al*	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}	Ratanotayan on *et al*	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

B. *Detailed Auditing Example*

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 [18]. 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.

This section shows an auditing example for the five case studies based on yardstick TR #6 (Traceability item relationships). Due to the space limitation we are giving this example to illustrate the process of auditing using the proposed model. The same processes are also used to summarize the auditing evidences in the next section.

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 II 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 IV-C "prototype architecture and implementation" and in figure "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 IV "Traceability Patterns" and Fig. 1 "conceptual model of traceability patterns".

TABLE II: AUDIT BASED ON YARDSTICK $_{TR}$ #6

Audit for Yardstick TR #6 (Traceability item relationships)				
	Case B (Lee *et al.*)	Case C (Ghazarian)	Case D (Ratanotayanon *et al.*)	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

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.

C. Summary of Auditing Evidences

Table III 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, seeing Table III:

TABLE III: SUMMARY OF THE TRACEABILITY AUDITING

	Case A_{TR} (Espinoza, Garbajosa)	**Case B_{TR} (Lee *et al.*)**	**Case C_{TR} (Ghazarian)**	**Case D_{TR} (Ratanotayanon *et al.*)**	**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

- 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.

VII. Conclusion and Future Work

Software organizations that adopt lightweight documentation processes such as XP find it a challenge to demonstrate that they meet ISO 9001 requirements by providing such documentation [19], [20].

This paper proposes 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. 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 demonstrated full support for the auditing yardsticks.

This paper has focused solely on designing an auditing model for the traceability requirements of ISO 9001. Future work is required to extend this model to include other ISO 9001 requirements, such as the control of design and development changes, as well as measurement analysis and improvement.

References

[1] J. Griesemer, "A field study of the impact of ISO 9001 on software development in the United States," PhD thesis, Pace University, United State of America, 1999.

[2] G. K, Fuller, "Antecedents and consequences of certification of software engineering processes," PhD Thesis, University of British Columbia, Canada, 2006.

[3] D. Vitoria, "Aligning XP with ISO 9001:2000-TickIT guide 5.0," Master Thesis, Dept. Software Engineering, Blekinge Institute of Technology, Sweden, 2004.

[4] G. Wright, "Achieving ISO 9001 certification for an XP company," *Lecture Notes in Computer Science, Extreme programming and Agile Methods*, pp. 43-50, 2003.

[5] L. Vijayasarathy and D. Turk, "Agile software development: A survey of early adopters," *Journal of Information Technology Management*, vol. 19, no. 2, pp. 1-8, 2008.

[6] C. Schindler, "Agile software development methods and practices in Austrian it industry results of an empirical study," in *Proc. International Conferences on Computational Intelligence for Modeling, Control and Automation*, 2008, pp. 321-326.

[7] A. Ghazarian, "Traceability patterns: An approach to requirement component traceability in agile software development," in *Proc. 8th WSEAS International Conference on Applied Computer Science*, 2008, pp. 236-241.

[8] B. Ramesh and M. Jarke, "Towards reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58-93, 2011.

[9] M. Cohn and D. Ford, "Introducing an agile process to an organization," *IEEE Computer*, vol. 36, no. 6, pp.74-78, 2003.

[10] W. G. Vincenti, *What Engineers Know and How They Know It*, Baltimore, London: The John Hopkins University Press, 1990, pp. 1-326.

[11] A. Abran, P. Bourque, R. Dupuis, J. Moore, and L. Tripp, *Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society Press, 2004, pp. 1-228.

[12] K. Meridji, "Analysis of software engineering principles from an engineering perspective," Ph.D. dissertation, Dept. software Eng, École de technologie supérieure, Montréal, Canada, 2010.

[13] M. Qasaimeh and A. Abran, "An audit model for ISO 9001 traceability requirements in agile (XP) environments," *Journal of Software*, vol. 8, no. 7, pp 1556-1567, 2013.

[14] S. Ambler, *Agile modeling: Effective Practices for eXtreme Programming and the Unified Process*, John Wiley & Sons, 1st edition, pp. 1-400, 2002.

[15] *Standards, Guidelines and Procedures for Information System Auditing*, ISACA, 2010,

[16] M. Lopez, "An evaluation theory perspective of the architecture tradeoff analysis method (ATAM)," CMU/SEI-20Q0-TR-012, Pittsburgh, PA, CMU/SEI, 2000.

[17] M. Lopez, "Application of an evaluation framework for analyzing the architecture tradeoff analysis method," *Journal of Systems and Software*, vol. 68, no. 3, pp. 233-241, 2003.

[18] *Framework for IT Governance and Control*, Cobit 4.1, 2008.

[19] M. Qasaimeh and A. Abran, "Investigation of the capability of XP to support the requirements of ISO 9001 software process certification," in *Proc. Eighth ACIS International Conference on Software Engineering Research Management and Applications*, 2010, pp. 239-247.

[20] M. Qasaimeh and A. 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 is an assistant professor of software engineering in Princess Sumaya University for Technology, Jordan. He received his PhD degree in software engineering form University of Quebec (Canada, 2012). He has a master's degree in information systems security from Concordia University (Canada, 2007), and a bachelor's degree in computer science from Jordan University of Science & Technology, (Jordan, 2003).

He has published several papers in reputed journals and international conferences. His research interests include agile software processes certification and compliance, software process and product improvement, software engineering ISO standards, software security and software engineering principles.

Alain Abran is a professor and the director of the Software Engineering Research Laboratory at Ecole de technologie superieure (ETS), University of Quebec (Montréal, Canada). He holds a Ph.D. in electrical and computer engineering (1994) from the École Polytechnique de Montréal (Canada) and master's degrees in management sciences (1974) and electrical engineering (1975) from the University of Ottawa (Canada).

He has over 15 years of experience in teaching in a university environment, and more than 20 years of industry experience in information systems development and software engineering. His research interests include software productivity and estimation models, software engineering foundations, software quality, software functional size measurement, software risk management, and software maintenance management. He has published over 300 peer-reviewed publications and he is the author of the book "Software Metrics and Software Metrology" and a co-author of the book "Software Maintenance Management" (Wiley Inter science, Ed., & IEEE-CS Press). Dr. Abran is co-editor of the 2004 Guide to the Software Engineering Body of Knowledge – SWEBOK, and he is the chairman of the Common Software Measurement International Consortium (COSMIC).