2016 IEEE 24th International Requirements Engineering Conference

TraceBoK: Toward a Software Requirements Traceability Body of Knowledge

Ana Marcia Debiasi Duarte
Universidade do Oeste de Santa Catarina
Chapecó, SC - Brazil
anamarciadd@gmail.com

Denio Duarte
Universidade Federal da Fronteira Sul
Chapecó, SC - Brazil
duarte@uffs.edu.br

Marcello Thiry
Universidade do Vale do Itajaí
Itajaí, SC - Brazil
thiry@univali.br

***Abstract*—This paper introduces the idea of building a body of knowledge (BoK) to gather the better practices in software requirements traceability that could bring major benefits for analyzing, managing, and implementing software changes impact. The implementation of traceability in the organizations is still a challenge even many studies have been conducted on the subject. The aim of this work is to evaluate traceability approaches focusing on software product traceability by a systematic review. We conducted a series of interviews with practitioners to complement the evaluation. Based on this study, we categorize the approaches, and we propose a body of knowledge (Bok) on traceability requirements *TraceBoK*. This BoK intends to assist software engineers to decide how to define an implementation of traceability in the development and maintenance of software products. Experts in traceability have evaluated this BoK, and we notice that it meets practitioners needs. However, some improvements must be done in the future versions, and we have made *TraceBoK* available at `tracebok.org`. We hope that the proposed BoK helps practitioners to apply better and understand traceability in both academic and industry areas.**

***Index Terms*—traceability, body of knowledge, software product**

I. Introduction

Software requirements traceability (SRT) is used to ensure consistency among the artifacts created during the development and maintenance of software products. An effective traceability approach depends on several factors such as architecture, technical modeling tools, among others. The implementation of traceability in the organizations is still a challenge even many studies have been conducted on the subject [1]. Literature has shown some reasons for that: cost of implementation, different stakeholders viewpoint, difficulties of maintaining updated information about requirements, and integrating all generated data from software development lifecycle. Furthermore, the lack of commercial tools for implementing methods and techniques for integrated traceability to the development process [2], [3], [4]. Moreover, as requirements evolve, to keep track of changes is also difficult.

A well-accepted definition of traceability (found in [1]) is "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction, *i.e.* from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases". From that, we can conclude that requirements traceability (RT) supports software management, software evolution, and validation.

In software engineering, traceability describes always requirements links. Requirements express needs and constraints of a software product, and traceability allows to describe and follow requirements steps [5], [6]. As a side effect, traceability makes easier to verify and validate requirements. Therefore, traceability supports software management, software evolution, and validation. When changes are made in a software product, traceability is fundamental to analyze changes impact. Besides, it helps the understanding, capturing, tracking, and verification of software artifacts, their relationships, and dependencies with other artifacts during the software lifecycle [7].

As point out in [1], requirements traceability has two directions from requirement: from a requirement to its artifacts or elements (built from it) and from a requirement to its sources.

Traceability may be classified considering the relationship level among the traced artifacts. It is called two-dimensional traceability. The horizontal (or inter traceability) refers to relationships between different levels (or models) of abstraction: from requirements to implementation going through design. In this classification, for example, trace links are created between requirements and source codes. The vertical (or intra-traceability) refers to relationships between the same levels (or model) of abstraction: between software components, related requirements, among others.

Definitions of traceability, traceability directions, and dimensionality underline the basis of traceability.

RT may be applied to different tasks of software development. To apply traceability, Mader and Gotel[8] proposed an SRT life cycle. It is composed of three generic processes: (i) defining traceability, (ii) create and maintaining traceability, and (iii) using traceability. A generic traceability process model shows the essential activities used in the software traceability. This model is essential to guide the organizational traceability strategy. A model proposed by [9] describes the four key activities of the generic traceability process model: traceability strategy, traceability creation, traceability maintenance, and traceability use. Figure 1 describes the four key activities of the generic traceability process model: traceability strategy, traceability creation, traceability maintenance, trace-

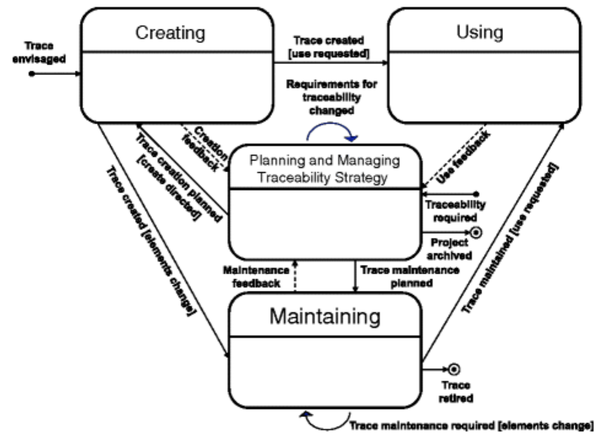

Creating
Using
Trace envisaged
Trace created [use requested]
Requirements for traceability changed
Creation feedback
Trace creation planned [create directed]
Trace created [elements change]
Planning and Managing Traceability Strategy
Use feedback
Traceability required
Project archived
Maintenance feedback
Trace maintenance planned
Trace maintained [use requested]
Maintaining
Trace retired
Trace maintenance required [elements change]

Fig. 1. Generic Traceability Process Model [9].

ability use [9].

Traceability has been considered an essential software development topic since 1968, and Bohem, in 1976 [9], published the first traceability survey. One of the main tasks of the requirements engineering (RE) is to assure SRT [10]. As reported by [4], traceability can bring important benefits in the areas like project management, process visibility, verification and validation, and maintenance. The improvement of readability, identification of reusable components, and impact analysis of changes are also benefited by the use of traceability [11].

Requirements fall into two general categories: functional and non-functional. In a nutshell, the former represents those related to what the application should do, and the latter represents those related to how the application should work [12]. Non-functional requirements are harder to trace since they can be ambiguous and non-measurable and, thus, we pay special attention to non-functional requirements in the proposed body knowledge. For example, non-functional requirements are more difficult to express in a quantifiable manner. They are also hard to be verified against individuals components, therefore, tracing non-functional requirements specifications toward the designing and vice-versa requires much attention of software engineers.

According to Gotel and Finkelstein[1], RT remains a widely reported problem in the software industry. The Cost of implementation, management of changes, different stakeholders viewpoint, and poor tool support are some causes of the misusing of RT [2], [3], [4]. Therefore, SRT is still challenging to implement since requirements can be, by nature, ambiguous, and some of them are hard to measure or implement.

In this context, we argue that a body of knowledge (BoK) that gathers and organizes approaches to managing SRT could help practitioners to understand and apply traceability in their daily software development tasks. Hence, software engineers can apply this BoK as a guideline for incorporating SRT in their products. Professional licensing and training programs will also find it useful.

In this work, we consider SRT in the context of software product development since software product development is wider than a software project.

We conducted a systematic review on software product RT to identify some characteristics that could be useful to build a BoK[1]. Based on a systematic review on software product RT, we propose an SRT organization to group approaches and tools to serve as a guide to implementing SRT in organizations. We present each approach based on the following topics: descriptions, restriction of using, process support tools, and how to apply the approach, the latter is divided into creation and use. Figure 2 presents how we organize TraceBoK:

- **Life Cycle** is the first knowledge area (KA), and it represents the base for traceability concepts and the other KA. Furthermore, this KA aims to elucidate (i) the main traceability concepts, and (ii) the organization of the process and its part of traceability life cycle. All approaches presented in this BoK follow the concepts in Traceability Life Cycle KA.
- **Functional KA** encompasses approaches for functional requirements traceability (FRT). Most of the traceability approaches deal with this kind of requirements.
- **Non-functional KA** gathers approaches that deal with non-functional requirements (NFRT). This KA is important since non-functional requirements have specific characteristics for traceability, such that: non-functional requirements come in many different shapes and sizes, and there is, therefore, no single traceability technique that can be applied in every case [13].
- **Product Line KA** address approaches that tackle product line traceability (SPLT) approaches.

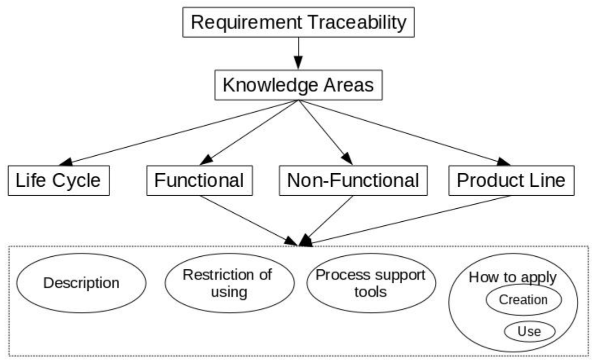


Fig. 2. TraceBok organization

As expected, KA provide a guide to some of the major traceability purpose and practices. The proposed knowledge areas were defined from reviews of the most using traceability approaches found in the literature and organizations.

[1]The systematic review was performed by one of the authors and validated by the other ones.

TraceBoK is then organized to improve the software traceability comprehension. Different groups of users may benefit from this guide. Software developers may use as a source for software development traceability approaches studies and as a guide for helping to choose approaches that fit their needs. Project managers may use for helping establish strategies for the use of traceability in organizations. Academic people may use to organize concepts and contents.

The reminder of the paper is organized as follows: next section presents how TraceBoK was conceived, Section III introduces TraceBoK, and, finally, Section IV concludes the paper with the summary of the research.

II. Our Methodology to Build TraceBoK

This section presents how TraceBoK was conceived. We based our process to building TraceBoK on the framework proposed in [14] because one of the authors has participated in building it. This framework, called PRO2PI-MFMOD[2], is a framework to modeling process reference model, and it is composed of seven sequential practices:

[P1] Initial decisions: corresponds to the initial decisions after the commitment for model development. In this step, the subject of interest was defined: software requirements traceability.

[P2] Sources analysis: corresponds to identify, gather, and analyze sources for building a model. To support this practice, besides analyzing the related work, we conduct a systematic review (Section II-A) and a survey (Section II-B).

[P3] Strategy for development: corresponds to the definition of the strategy to be used to develop a model. Here, we started with the preliminary results of our survey, and we analyzed some available bodies of knowledge (*e.g.*, SWEBoK [15], PMBoK [16], among others).

[P4] Model design: corresponds to design a model. After defining the strategy for development, we determined the processes and structures of TraceBoK (Figure 2 shows the result of this practice).

[P5] Draft model development: corresponds to build the first version of a model. In this step, we proposed the first version of TraceBoK, and we published it at `tracebok.org`.

[P6] Draft model validation: after publishing TraceBoK on the web, we conduct a survey with experts on requirements to validate our proposal (Section II-C).

[P7] Model consolidation: this practice is still under construction since TraceBoK intends to be an "alive" document.

In the following, we present more details of TraceBoK conception.

A. Traceability Aproach Identification

One of the definitions of body of knowledge proposed in [17] is *(i)* a structured knowledge that is used by members of a discipline to guide their practice or work, and *(ii)* the prescribed aggregation of knowledge in a particular area an individual is expected to have mastered to be considered or certified as a practitioner. We focus our attention on (i), and a systematic literature review (SLR) was conducted to structure the knowledge around software traceability approaches. We follow the method defined in [18] to achieve that. Notice that we conducted an SRL to help us to propose TraceBoK, and it is not the aim of this work.

This SLR proposed aims to answer three research questions:
a) *RQ1*: Which are SRT approaches proposed in the context of software development organizations?
b) *RQ2*: May traceability approaches to software product development be applied to software project development and *vice versa*?
c) *RQ3*: Are there traceability approaches that can be applied after software implementation?

We built a search string (SS) to help answer the above questions: *(traceability* **and** *requirement* **and** *software)* **and** *(tracing* **or** *product* **or** *approach* **or** *impact* **or** *technique* **or** *tool* **or** *method* **or** *methodology* **or** *process* **or** *"life cycle").* Our SS was used to query the main repositories, and we delimited publications from 2007 to 2014. We based our SS on the similar study presented in [10]. This study covered works from 1997 to 2007, and we extended that period. The queried repositories were IEEE Xplorer, ACM Digital Library, Spring Link and Science Direct.

In the first selection and deduplication, we got 560 studies. Then, we proceeded with the first analysis to identify whether or not the study was *right on the target*. Titles, abstracts, keywords, among others were analyzed. We retained 102 studies from this analysis. After a full-text reading, 26 relevant studies are kept to helping to build the proposed BoK. For the sake of simplicity, we call selected approaches from SLR (SA_{SLR}) those approaches resulted from the SLR.

The research question *RQ1* was answered from the selected studies.

Moreover, SA_{SLR} were classified according to their target domain: requirements in general (any requirements); software product line (requirements used only in software product line context); security requirements (deals with only security requirements); non-functional requirements (traces only NFR); function point (traces from function point to source code); goal-centric traceability (traces goals and assessment models); tests (specific for tracing requirements and test-cases); and airplane software (deals specifically with airborne software requirements). Table I shows approaches and target domain found in SA_{SLR}

References marked with asterisk show approach dealing only with software project and cannot be extended to the product ($RQ2$). However, in a product context traceability artifacts can be reused and enhanced traceability experience in other projects. To build TraceBok, we focus only on software product. Moreover, we discarded target domains with only one reference.

We also investigate whether SA_{SLR} may be applied after software implementation or not ($RQ3$). We found four approaches [41], [24], [26], and [7] that can be used after decisions about which artifacts will be produced during a

[2]Method Framework for Engineering Process Capability Models as an element of the Process Capability Profile to drive Process Improvement

TABLE I
PURPOSE TRACEABILITY APPROACHES.

Target	References
Requirements Traceability	[19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [7], [29], [30], [31]*, [32] *
Software Product Line	[33], [34], [35]
Security Requirements	[36], [37]
Non Functional Requirements	[38], [39]
Function Point	[11] *
Goal-Centric Traceability	[40]
Tests	[41]
Airplane Software	[42]

project. Remark that some target domains on Table I have only one reference. It is due since other references do not follow the proposed criteria: the work must propose an approach, and the approach must have empirical results.

The next step after SLR accomplished, we built a survey to identify how software companies apply SRT. Next section describes briefly how our survey was built.

B. Survey

We conducted a survey in software development companies[3] to identify how SRT is applied to managing the changes in requirements. (Table II presents some characteristics of surveyed companies). The companies are inquired about one of their software product. The survey was based on the features from SA$_{SLR}$. The main purpose of the survey is to identify which of those features companies are using. We apply the methodology proposed in [43] to conduct our survey: planning, execution, and analysis of results. Although interviewed companies are strictly Brazilian, their characteristics are similar to several companies around the world as we can see in Table II.

TABLE II
CHARACTERISTICS OF THE INTERVIEWED COMPANIES.

#developers	less than 10	40%
	between 10 and 30	45%
	more than 30	15%
Product Domain	Commerce	30%
	Industry	20%
	Others	50%
Product Age (years)	less than 5	48%
	less than 9	16%
	9 or more	36%
Maturity Model	ISO 9000	10%
	CMMI	5%
	MPS.BR	15%
	None	70%

Two goals were defined in planning task: (G_1) to find which features of SA$_{SLR}$ companies are using for developing their products and (G_2) to find which features of SRT are shared among companies. All needed information was organized based on Goal Question Metrics (GQM) [44].

The survey was organized in three parts: the first one is composed of seven question ($Q0.n$) and intends to gather information about the company (context information), the second one is composed of four questions ($Q1.n$) and intends to verify G_1, and the latter is composed of three questions ($Q2.n$) and intends to verify G_2:

Q0.1 In what state is your company located?
Q0.2 In what city is your company located?
Q0.3 How many developers does your company have?
Q0.4 What business category is your product designed for?
Q0.5 How long has your company maintained its product?
Q0.6 Does your company follow any maturity model (CMMI, ISO, etc)?
Q0.7 Does your company apply SRT on any level of its product requirements?
Q1.1 Does SRT used in your company meet the purpose presented in SA$_{SLR}$?
Q1.2 Are your traced artifacts suitable with artifacts proposed by some SA$_{SLR}$?
Q1.3 Are the modeling patterns used by your company compatible with the models required by some SA$_{SLR}$?
Q1.4 Are your STR tools compatible with the proposed tools in SA$_{SLR}$?
Q2.1 Are your approaches to trace artifacts applied to software product or software project?
Q2.2 Does your company use automated tools to support STR?
Q2.3 Does your company apply SRT during the product construction?

We built a first version, and we applied the survey to two software development companies to validate it. After that, some adaptations were made, and the survey was sent to other twenty companies. We followed and clarified some questions during the response process.

In the first part of our survey, we had the following results:

- Twenty company from Southern Brazil answered our survey and most of them design software to commerce management.
- The number of developers follows the range: > 30 (15%), ≥ 10 and < 30 (45%), and < 10 (40%).
- The age of the products varies between 3 to 9 years. 70% do not follow any maturity model.
- All companies apply SRT (fully or partially) to their projects/products.

In the second part, we had the following results:

- Most companies use SRT to trace requirements (63%). The other companies use SRT to trace customer requests.
- Most companies (87%) trace artifacts proposed by some of SA$_{SLR}$
- Most companies (73%) do not use pattern models compatible with SA$_{SLR}$.
- Most of tools used by companies (95%) are not compatible with SA$_{SLR}$.

Finally, the results from third part were:

- Companies apply SRT to software product sum up 53%, and software project 47%.
- Most companies use automated tools to trace artifacts (95%).

[3]This survey was conducted in Brazilian software companies.

- All companies planned SRT before constructing the product.

We can conclude from our survey that artifacts traced by companies are the same proposed in SA_{SLR}, however, model standards and tools are not those proposed in SA_{SLR}. Maybe, tools and models from SA_{SLR} represent the state-of-art of STR and companies prefer to apply a traditional approach to manage SRT. All companies apply SRT in some of their product life cycle; however, none of them apply SRT after finishing the product. Half of the companies apply SRT to software project context as well.

The systematic review showed us that there is a gap between the research and practice of RT. The survey, on the other hand, showed us that organizations apply RT but not in a systematic way. With this in mind, TraceBoK was conceived. To verify whether or not TraceBoK is useful to organizations, we conducted a survey with SRT experts to validate the draft version. Next section presents how the validation process was conducted.

C. Validation

We invited seven experts in software development and traceability to validate our BoK proposal. We chose experts with relevant domain background in SRT and maturity in software development. Moreover, we introduce TraceBok for each expert by a short explanation of how it is organized. To reduce the threats and anomalies in the validation process, we define some criteria to select experts: to have a degree in computer science area, to have managed software development team, and to have applied SRT in the last three years or to be lead appraiser for a maturity model.

Experts answered a survey with our supervision. Survey was composed of eight questions: (Q1) Is the TraceBoK presented (chapter, sections) in a comprehensive way? (Q2) Do you think it is easy to find information about traceability? (Q3) Does TraceBoK present some approach that you have known previously? (Q4) Do you know some SRT approach that TraceBoK does not describe? (Q5) Do you think TraceBoK presents at least one approach that can be applied to a company you work or know? (Q6) Do you think TraceBoK may help practitioners to apply or enhance SRT? (Q7) Do you think approaches proposed by TraceBoK are aligned with the quality models you know? (Q8) Would you use TraceBoK in your company to implement SRT?

The answers follow Likert scale [45] (1 to 5, the higher the score closer to yes is the answer, being 3 a neutral answer). Table III presents the result of experts survey.

TABLE III
INITIAL VALIDATION RESULTS IN % (ROUNDED).

Question	5	4	3	2	1
Q1	14	72	0	14	0
Q2	0	43	0	57	0
Q3	0	86	0	14	0
Q4	57	14	29	0	0
Q5	29	71	0	0	0
Q6	57	43	0	0	0
Q7	57	0	43	0	0
Q8	14	86	0	0	0
Total	28	52	9	11	0

Taking into account neutral answer (3), TraceBoK summed up 80% as positive answers. It means that TraceBoK first version may be applied and used in software companies to manage requirements traceability. From the survey, we can point out some improvements:

- Almost half of experts (*i.e.*, 3) did not have an opinion about quality models and approaches proposed in TraceBoK (Q7). We can conclude that some experts do not know quality models or they are not able to map approaches in TraceBoK and known quality models.
- In Q4, we notice that 71% of experts (*i.e.*, 5) know approaches that are not presented in TraceBok. In this case, TraceBoK may be presented specific approaches for specific domains, and we have to identify and fix it.
- About Q2 (more than 50% do not agree with it), we notice that we have to reorganize TraceBoK to make easier to find information about SRT.

D. Threats to validaty

This study represents an initial attempt to propose a BoK on requirements traceability. As we have selected experts by criteria presented above, we claim that the major limitation of TraceBoK validation, and therefore a threat to its validity is that how the experts tackled the survey. We mitigate this threat by following the experts while answering our survey. Thus, when an expert had a doubt concerning a question or an approach from TraceBoK, we immediately tried to explain and clarify the doubt.

Another threat could be our sample of experts: we have just selected Brazilian experts, but we believe that our criteria of selection helped to mitigate that threat.

From survey applied to experts, we conclude that, although there are several points to improve, TraceBok in its first version may be used by practitioners to improve SRT. Next section presents details of TraceBok.

III. TRACEBOK

Based on the systematic review presented previously and some experts interviews, we notice that organizations do not apply SRT systematically. Generally, they use traceability on source code not taking into account other traceable artifacts. However, traceability must be applied in every kind of requirements to guarantee the management of artifacts produced during development and maintenance of software products [46], [21]. We also notice that most of researches on SRT have not been applied in the industry since they are spread across several repositories. Hence, TraceBoK intends to contribute to the organizations in applying traceability by making available the knowledge about approaches in RT to practitioners who want to apply RT systematically in their products.

TraceBoK, as stated before, is organized in four KA and three of them aim to group the traceability approaches into categories oriented to user needs (Life Cycle KA intends to support the other KA). Therefore, approaches found for traceability are grouped as Functional Requirements, Non-Functional Requirements, and Product Line Traceability. Each approach is described in such way that practitioners can evaluate whether or not the approach fits traceability requirements for their application.

We propose a specific knowledge area for software product line (SPL) since SPL generates a large number and heterogeneity of documents during the development. This may cause difficulties to identify common and variable aspects among applications, and to reuse core assets that are available under the product line. The commonalities and variabilities in software product line increase the traceability difficulty. Approaches related to this knowledge area present alternative solutions for traceability in SPL development.

Notice that knowledge areas are not intended to represent phases in traceability. The objective is to organize available literature approaches according to common characteristics. Thus, this BoK gather approaches by traceability uses and objectives.

Approaches chosen from SLR were classified and are presented in TraceBoK as Funcional (FRT): [19], [21], [22], [23], [26], [24], [30], [27], [28], [7], [25], [20]; Non Funcional (NFRT): [38], [36], [37], [35], [20]; and Software Product Line Traceability (SPLT): [33], [34], [29].

Besides identifying the key areas for helping SRT implementation, it is necessary to organize the knowledge in each area. To achieve that, each approach is presented following the organization showed in Figure 2:

Description: we describe the approach briefly: definition, purpose, utilization, among others.

Restriction of using: we present some restrictions of using the approach. A restriction may be an environment where the approach can be applied. The purpose of showing the restriction is to help practitioners to choose the best approaches for their development environment.

Process support tools: we present which tools can be used to implement the approach. Tools can be plugins for an available integrated development environment (IDE) like Eclipse or implemented specifically for the approach or a methodology, or a model.

How to apply: we give an overview on how to prepare the environment to apply the approach (*e.g.*, methodologies, tools). *Creation* shows how to create traceability artifacts. It extends the explanation given in *How to apply* about create traces. *Use* presents how to use and manage traceability artifacts. Based on the approach, we may also give some hints to trace the requirements successfully.

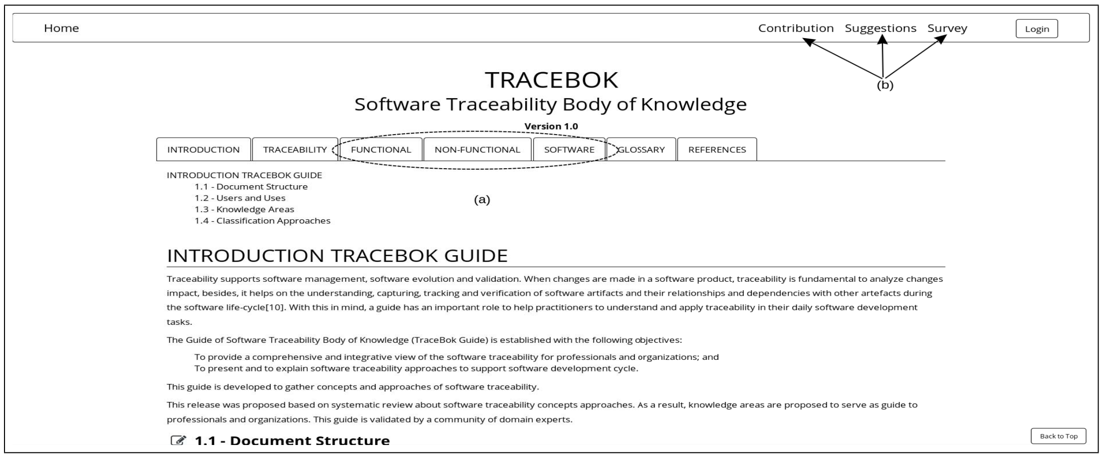
In TraceBoK, environments, tools, and frameworks are presented altogether with the approaches to help users to choose what approach (or approaches) fits better into their projects. Moreover, we try to show different perspectives to give an overview of the field of SRT. A successful traceability depends on several factors: tools, environment characteristics, team maturity, good planning, among others. TraceBoK does not intend to solve problems faced during the traceability implementation; instead, it intends to serve as a guide for practitioners to apply it in such way all requirements are traced.

Besides of the four KA, a glossary is available containing the most used terms in traceability and requirements [47]. Figure 3 presents a screenshot of TraceBoK's website. The highlighted area ((a)) shows the menu of the three main KA where users can browse and search approaches that meet their needs in a KA of interest. Remark that, from Figure 3(b), users and practitioners may also contribute to TraceBoK improvement by interacting through the available options. In the option *Survey*, users can answer the same questions presented in Section II-C and, thus, we can have a wider perception of TraceBoK. Interested readers who want to explore TraceBoK in details are referred to `tracebok.org`.

We conducted a systematic review of available BoK in the literature for software development to proposing TraceBoK structure: Business Analysis Body of Knowledge (BABoK) [48], Geographic Information Science and Technology Body of Knowledge GIS&T BoK [49], Body of Knowledge on Model Checking for Software Development (MCBoK) [50], Project Management Body of Knowledge (PMBoK), Personal Software Process Body of Knowledge (PSPBoK) [51], Requirements Engineering Body of Knowledge (REBoK) [52], Systems Engineering Body of Knowledge (SEBoK) [53], Software Process pErformance Analysis Knowledge-based EnviRonment (SPEAKER) [54], SWEBoK, and Team Software Process Body of Knowledge (TSPBoK) [55]. All studied BoK have a common feature: they gather what practitioners need to understand and what tasks must be performed in a defined area of interest. This grouping is called knowledge area. Table IV shows how we separate the organization of studied BoK comparing to TraceBoK. We separate as follows: whether or not the BoK presents an introduction (Introd.), basics of the subject (Basics), who and how a KA can be used (User and Uses), or a glossary (Glossary). Remark that, as expected, all BoK have an introduction and KA. Based on the result of the systematic review, we decided to organize TraceBoK to meet what we have considered the best practices of each BoK.

Although we have analyzed several BoK, TraceBoK is mainly based on the structure and model of PMBoK and SWEBoK since they are the most known and well-established BoK in the industry.

TraceBoK is in the first release, and we intend it become a live document. To accomplish that, we made TraceBoK available on the web (see Figure 3) so people can suggest updates and deletions over the time. Interested parties and subject matter experts are invited to contribute to the ongoing development and refinement of this body of knowledge. As shown previously, in the website `tracebok.org`, readers can find all details of TraceBoK and can help to improve it.



TRACEBOK

Software Traceability Body of Knowledge

INTRODUCTION TRACEBOK GUIDE
1.1 - Document Structure
1.2 - Users and Uses
1.3 - Knowledge Areas
1.4 - Classification Approaches

INTRODUCTION TRACEBOK GUIDE

Traceability supports software management, software evolution and validation. When changes are made in a software product, traceability is fundamental to analyze changes impact, besides, it helps on the understanding, capturing, tracking and verification of software artifacts and their relationships and dependencies with other artefacts during the software life-cycle[10]. With this in mind, a guide has an important role to help practitioners to understand and apply traceability in their daily software development tasks.

The Guide of Software Traceability Body of Knowledge (TraceBok Guide) is established with the following objectives:

To provide a comprehensive and integrative view of the software traceability for professionals and organizations; and
To present and to explain software traceability approaches to support software development cycle.

This guide is developed to gather concepts and approaches of software traceability.

This release was proposed based on systematic review about software traceability concepts approaches. As a result, knowledge areas are proposed to serve as guide to professionals and organizations. This guide is validated by a community of domain experts.

1.1 - Document Structure

Fig. 3. Screenshot from TraceBoK site (`tracebok.org`).

TABLE IV
THE ORGANIZATION OF STUDIED BOK AND TRACEBOK.

Bok	Introd.	Basics	KA	User and Uses	Glossary
BABoK	X		X		X
GIS&T BoK	X		X		
MCBoK	X		X		X
PSPBoK	X		X	X	
PMBoK	X	X	X		X
REBoK	X		X		
SEBoK	X		X	X	
SPEAKER	X		X	X	
SWEBoK	X	X	X		
TSPBoK	X		X		
TraceBoK	X	X	X	X	X

Next section presents an example of a use of TraceBoK.

A. *Example of Use*

In this section, we present how TraceBoK can help a software development organization in managing requirements traceability. This example is based on a real case.

For the sake of confidentiality, the name of the organization is omitted, and we name it *Sofho*. *Sofho* applies RT in its products but in *ad hoc* way, using *Git* and *Redmine*. For this study, *Sofho* wants to apply RT formally in the development of a new product (*NP*) for hotel accounting management . *Sofho* applies a mix of formal requirements specification and agile cycles as software development methodology.

A traceability team (TT) was formed, consisting of a process analyst, a project manager, a business analyst, a configuration analyst, and a programmer. TT started planning how RT would be carried out by reading the first part of TraceBoK, that is, Life Cycle KA. Figure 1 shows a generic traceability process model followed by TT (Section 2.2 of TraceBoK). The main objective of TT was to link product features of *NP* to its source code, focusing on functional requirements.

Firstly, the artifacts to be traced were defined: features, routines, stories, tasks and source codes. After artifacts definition, TT studied some approaches presented in Functional Requirements KA (FRT). They decided to explore #FRT01 (Requirement and Source Code Traceability [19]) and #FRT03 (End-to-end Software Traceability [21]). Both approaches were fully studied to find the best alternative. This study was accomplished by reading the papers cited in #FRT01 ad #FRT03. Those papers are referenced in *Reference* option in the TraceBok.

The team discarded #FRT03 since RT should follow a workflow implemented using MS SharePoint[4], and they did not want to buy another tool to implement traceability. Although #FRT01 approach is based on UNICASE tool[5], an Eclipse plugin, TT decided to apply it using only the proposed model (Traceability Information Model - TIM). In this case, the available tools for tracing *NP* would be customized to meet TIM. Figure 4 presents pictorially how TIM is organized: system model artifacts (feature, functional requirement), project model artifacts (sprint, work item, developer) and code model artifacts (code file, revision). This organization met the needs of TT to trace the artifacts of *NP*.

Since there are some differences regarding traceable artifacts, a customized model (based on TIM) was built (see Figure 5 - the attributes are omitted). In the system model, instead of using functional requirements, the team decided to trace routines. In the project model, work items were replaced by stories and tasks. Finally, the code model was kept unchanged.

A set of tools were defined to manage the traces during the development of *NP*: Redmine and Git (tools recommended by *Sofho*). Redmine was used as requirements repository and control of sprints. Thus, Redmine stored the objects features, routines, stories, sprints, tasks and their links. The traces between requirements artifacts and source codes were implemented using Git. Thus, in every commit into Git repository, the programmer registered the task description and id (this id is taken from Redmine repository).

[4]https://products.office.com/en-us/sharepoint/collaboration
[5]https://marketplace.eclipse.org/content/unicase

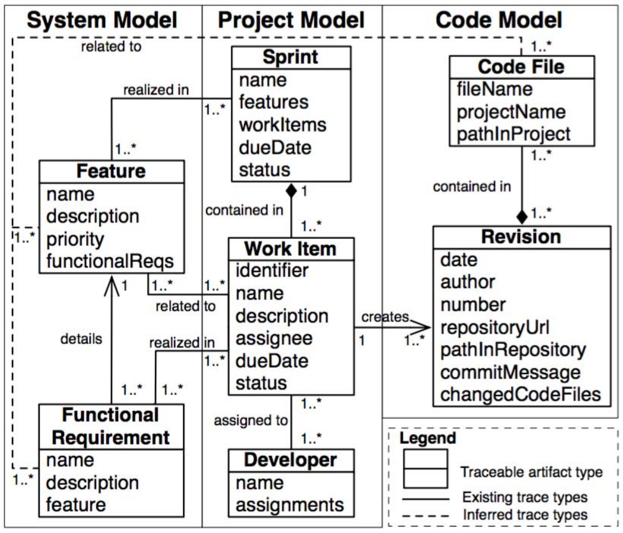


Fig. 4. Traceability Information Model.

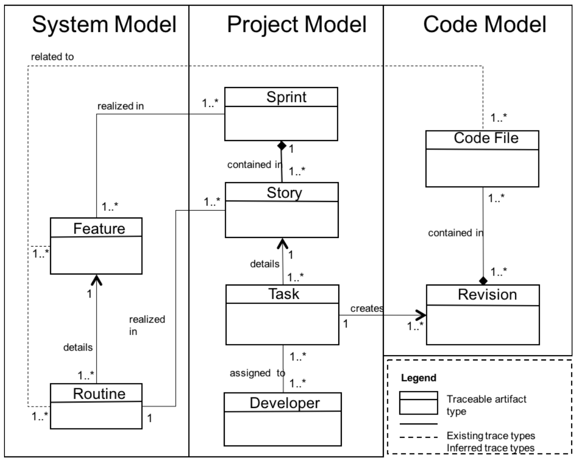


Fig. 5. The customized TIM.

Figure 6 shows a screenshot of how Redmine is used for controlling traceability among the traced artifacts. The leftmost column presents the artifacts id (Column #), and when developers click on one of id, the traces of the selected artifact are presented. We remark that Redmine and Git are configured to work together, that is, by clicking in one of the story id, a member of the team could see details and the set of tasks of that story. Tasks are related to the code revision (see Figure 5). In the screenshot, remark that the traced feature is *Accounting* (see *Feature Accounting*), and the rows present some artifacts of *Accounting*. Notice also that, in this screenshot, there are four traced routines (*504 - CREATE ACCOUNTING, 505 - ACCOUNTING MANAGEMENT, 506 - CLOSING ACCOUNTING, 507 - ACCOUNTING REPORT*). Each routine is composed of traced stories (see Column *Subject*), *e.g.*, *ACCOUNT MANAGEMENT* has one story: *512 - Accounting - Temporary Accounting Creation*.

The development of *NP* is still in progress, but TT is very satisfied with the way the RT is being managed. From this experience, we can claim that TraceBoK may be very useful for implementing traceability. We point out some reasons: (i) TraceBoK presents the basic concepts of RT, and that helps practitioners to plan the traceability strategies, (ii) practitioners have several choices of approaches to applying traceability, (iii) TraceBoK also presents general views of approaches, and practitioners are encouraged to explore each approach by selecting available references, and (iv) it encourages practitioners to have a standard view of RT, and it makes easier RT implementation in organizations.

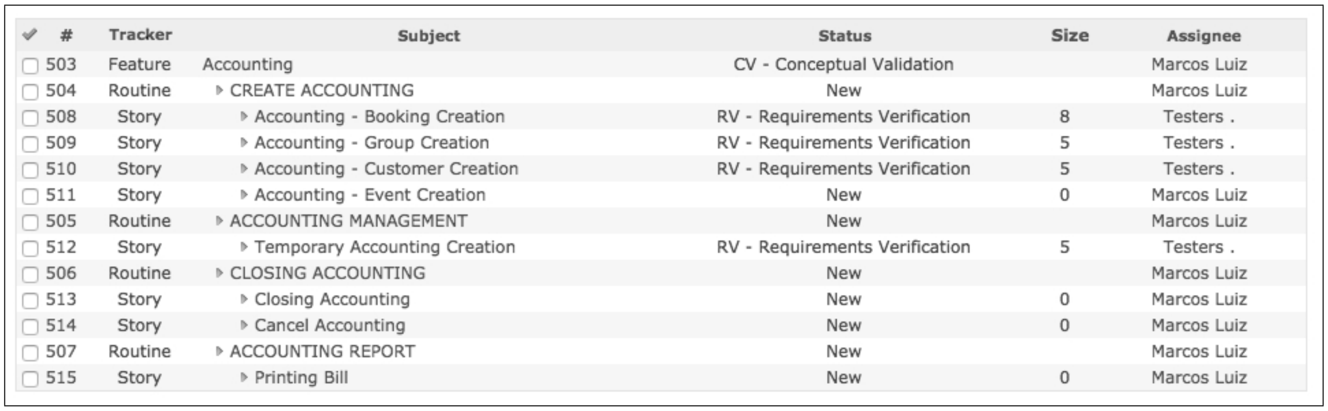
IV. Conclusions

A body of knowledge has a significant role to advancement of an area as knowledgeable practice. Define a BoK for a particular area means to present the basics and extent of the knowledge that would be expected of any professional within that field. In most all fields, practitioners are who apply the knowledge built from researches. However, to transfer the results of researches from academy to the industry is always a hard task [56], [57]. Bodies of knowledge may be an effective way to organize knowledge from research areas and make them available for practitioners and other researchers.

With this in mind, this paper presented a first version of a body of knowledge on software requirements traceability, TraceBoK. We propose an organization to traceability requirements (see Figure 2). Moreover, TraceBoK is organized in four knowledge areas based on specificities of requirements needs. One KA intends to support the other KA presenting the basis of traceability, named Life Cycle KA. The other KA are organized to cover traceability on functional and non-functional requirements besides requirements from software product line, named Functional, Non-functional and Product Line KA, respectively. These three KA are presented following a schema: *description*, *restriction of using*, *process support tools* and *how to apply*. This schema allows practitioners to understand how the approach works and, thus, decide which fits better their needs.

To build TraceBoK, we first conduct a systematic review of traceability requirements (as shown in Section II) to give us an overview of existing approaches in the research area. After selecting some approaches, we built the first version, and we conduct a survey of some experts in requirements management and requirements traceability. From the survey, we validate TraceBoK and make it available to any professional who is interested in querying, making suggestions, and corrections.

TraceBok aims to provide a complete high-level overview of the basics and approaches that comprise the knowledge required for a successful implementation of SRT as software engineering or software development manager. This document is not meant to provide detailed descriptions or in-depth explanations of the approaches and procedures of every component in RT. Rather, the purpose of this document is to provide an overview of the concepts, knowledge areas, and approaches



✓	#	Tracker	Subject	Status	Size	Assignee
☐	503	Feature	Accounting	CV - Conceptual Validation		Marcos Luiz
☐	504	Routine	▸ CREATE ACCOUNTING	New		Marcos Luiz
☐	508	Story	▸ Accounting - Booking Creation	RV - Requirements Verification	8	Testers .
☐	509	Story	▸ Accounting - Group Creation	RV - Requirements Verification	5	Testers .
☐	510	Story	▸ Accounting - Customer Creation	RV - Requirements Verification	5	Testers .
☐	511	Story	▸ Accounting - Event Creation	New	0	Marcos Luiz
☐	505	Routine	▸ ACCOUNTING MANAGEMENT	New		Marcos Luiz
☐	512	Story	▸ Temporary Accounting Creation	RV - Requirements Verification	5	Testers .
☐	506	Routine	▸ CLOSING ACCOUNTING	New		Marcos Luiz
☐	513	Story	▸ Closing Accounting	New	0	Marcos Luiz
☐	514	Story	▸ Cancel Accounting	New	0	Marcos Luiz
☐	507	Routine	▸ ACCOUNTING REPORT	New		Marcos Luiz
☐	515	Story	▸ Printing Bill	New	0	Marcos Luiz

Fig. 6. Screenshot of Redmine using customized TIM.

that constitute the essential knowledge, skills, and abilities of expert practitioners.

We are aware that this BoK is the first step to building a complete document for helping software engineering in tracing requirements. We are interested in TraceBoK working as a live document and improvement during its life cycle will lead a useful tool for SRT. So, we want to ensure that a growing, reliable, and accessible BoK remains available to all individuals, organizations, and communities worldwide.

References

[1] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of the First International Conference on Requeriments Engineering*. IEEE, 1994, pp. 94–101.

[2] G. Spanoudakis and A. Zisman, "Software traceability: a roadmap," *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, pp. 395–428, 2005.

[3] G. Regan, F. McCaffery, K. McDaid, and D. Flood, "The barriers to traceability and their potential solutions: Towards a reference framework," in *Software Engineering and Advanced Applications (SEAA)*, 2012, pp. 319–322.

[4] A. Kannenberg and H. Saiedian, "Why software requirements traceability remains a challenge," *CrossTalk The Journal of Defense Software Engineering*, vol. 22, no. 5, pp. 14–19, 2009.

[5] I. Sommerville and G. Kotonya, *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[6] S. Pfleeger and J. M. Atlee, *Software Engineering: Theory and Practice*, 4th ed. Prentice Hall, 2010.

[7] A. Goknil, I. Kurtev, and K. van den Berg, "Tool support for generation and validation of traces between requirements and architecture," in *Proceedings of the 6th ECMFA Traceability Workshop*. ACM, 2010, pp. 39–46.

[8] P. Mäder and O. Gotel, "Towards automated traceability maintenance," *Journal of Systems and Software*, vol. 85, no. 10, pp. 2205–2227, 2012.

[9] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and systems traceability*. Springer, 2012, vol. 2, no. 3.

[10] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, R. U. Akbar, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 3, pp. 385–433, 2012.

[11] P. J. A. Vianna Ferreira and M. d. O. Barros, "Traceability between function point and source code," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*. ACM, 2011.

[12] I. Sommerville, *Software Engineering*, 9th ed. Pearson Education Inc., 2010.

[13] J. Cleland-Huang and D. Schmelzer, "Dynamically tracing nonfunctional requirements through design pattern invariants," in *Workshop on Traceability in Emerging Forms of Software Engineering*, October 2003.

[14] C. F. Salviano, A. Zoucas, J. V. Silva, A. M. Alves, C. G. von Wangenheim, and M. Thiry, "A method framework for engineering process capability models," in *16th European Systems and Software Process Improvement and Innovation*, 2009.

[15] P. Bourque, R. E. Fairley *et al.*, *Guide to the Software Engineering Body of Knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[16] PMI, *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*, 5th ed. Project Management Institute, 2013.

[17] T. I. Oren, "Toward the body of knowledge of modeling and simulation," in *Interservice/Industry Training, Simulation, and Education Conference*, (I/ITSEC), Ed., 2005.

[18] P. Brereton, B. Kitchenham, D.Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571 – 583, 2007.

[19] A. Delater and B. Paech, "Analyzing the tracing of requirements and source code during software development," in *Proceedings of the 19th international conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2013.

[20] H. Dubois, M.-A. Peraldi-Frati, and F. Lakhal, "A model for requirements traceability in a heterogeneous model-based design process: application to automotive embedded systems," in *15th IEEE International Conference on Engineering of Complex Computer Systems*. IEEE, 2010.

[21] H. U. Asuncion, F. François, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM, 2007.

[22] J. Cleland-Huang, P. Mäder, M. Mirakhorli, and S. Amornborvornwong, "Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders," in *20th IEEE International Requirements Engineering Conference*. IEEE, 2012.

[23] G. Buchgeher and R. Weinreich, "Automatic tracing of decisions to architecture and implementation," in *9th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2011.

[24] A. Egyed, G. Binder, and P. Grunbacher, "STRADA: A tool for scenario-based feature-to-code trace detection and analysis," in *Companion to the proceedings of the 29th International Conference on Software Engineering*. IEEE, 2007.

[25] P. Mäder, O. Gotel, and I. Philippow, "Semi-automated traceability maintenance: An architectural overview of tracemaintainer," in *31st International Conference on Software Engineering-Companion Volume (ICSE)*. IEEE, 2009.

[26] Y. Yu, J. Jürjens, and J. Mylopoulos, "Traceability for the maintenance of secure software," in *IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2008.

[27] I. Malavolta, H. Muccini, and V. S. Rekha, "Supporting architectural

design decisions evolution through model driven engineering," in *3th SERENE*, 2011.

[28] K. Sousa, H. Mendonça, J. Vanderdonckt, and M. S. Pimenta, "Supporting requirements in a traceability approach between business process and user interfaces," in *Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems*. SBC, 2008.

[29] W. Jirapanthong and A. Zisman, "Xtraque: traceability for product line systems," *Software & Systems Modeling*, vol. 8, no. 1, pp. 117–144, 2009.

[30] H. Schwarz, J. Ebert, and A. Winter, "Graph-based traceability: a comprehensive approach," *Software & Systems Modeling*, vol. 9, no. 4, pp. 473–492, 2010.

[31] Y. Hong, M. Kim, and S.-W. Lee, "Requirements management tool with evolving traceability for heterogeneous artifacts in the entire life cycle," in *8th International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 2010.

[32] J. Cleland-Huang, J. H. Hayes, and J. Domel, "Model-based traceability," in *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE, 2009.

[33] M. Moon, H. S. Chae, T. Nam, and K. Yeom, "A metamodeling approach to tracing variability between requirements and architecture in software product lines," in *7th IEEE International Conference on Computer and Information Technology*. IEEE, 2007.

[34] T. K. Satyananda, D. Lee, S. Kang, and S. I. Hashmi, "Identifying traceability between feature model and software architecture in software product line using formal concept analysis," in *International Conference on Computational Science and its Applications*. IEEE, 2007.

[35] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq, "A sysml-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies," *Information and Software Technology*, vol. 54, no. 6, pp. 569–590, 2012.

[36] P. Sanchez, D. Alonso, F. Rosique, B. Alvarez, and J. Pastor, "Introducing safety requirements traceability support in model-driven development of robotic applications," *Computers, IEEE Transactions on*, vol. 60, no. 8, pp. 1059–1071, 2011.

[37] J.-S. Lee, V. Katta, E.-K. Jee, and C. Raspotnig, "Means-ends and whole-part traceability analysis of safety requirements," *Journal of Systems and Software*, vol. 83, no. 9, pp. 1612–1621, 2010.

[38] A. Yrjönen and J. Merilinna, "Tooling for the full traceability of non-functional requirements within model-driven development," in *6th ECMFA Traceability Workshop*, 2010.

[39] J. Merilinna, A. Yrjönen, and T. Räty, "NFR+ framework method to support bi-directional traceability of non-functional requirements," *Computer Science-Research and Development*, vol. 30, no. 1, pp. 35–49, 2015.

[40] J. Cleland-Huang, W. Marrero, and B. Berenbach, "Goal-centric traceability: Using virtual plumblines to maintain critical systemic qualities," *IEEE Trans. Softw. Eng.*, vol. 34, no. 5, pp. 685–699, Sep. 2008.

[41] C. Ziftci and I. Krueger, "Tracing requirements to tests with high precision and recall," in *26th IEEE/ACM ASE*, 2011.

[42] T. Levendovszky, D. Balasubramanian, K. Smyth, F. Shi, and G. Karsai, "A transformation instance-based approach to traceability," in *6th ECMFA Traceability Workshop*, 2010.

[43] R. M. Groves, F. J. Fowler Jr, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey methodology*. John Wiley & Sons, 2011, vol. 561.

[44] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Transactions on Software Engineering*, no. 6, pp. 728–738, 1984.

[45] R. A. Likert, "A technique for the measurement of attitudes," in *Archivés of Psychology*, vol. 140, 1932, pp. 1–55.

[46] S. Rochimah, S. Kadir, M. Wan, and A. Abdullah, "An evaluation of traceability approaches to support software evolution," in *Proceedings of the ICSEA*, 2007.

[47] O. C. Gotel and S. J. Morris, "Case-based stories for traceability education and training," in *IEEE 7th International Workshop on Requirements Engineering Education and Training*. IEEE, 2012.

[48] I. I. of Business Analysis IIBA, "Iiba," February 2014. [Online]. Available: http://www.iiba.org/

[49] D. DiBiase, M. DeMers, A. Johnson, K. Kemp, A. T. Luck, B. Plewe, and E. Wentz, "Introducing the first edition of geographic information science and technology body of knowledge," *Cartography and Geographic Information Science*, vol. 34, no. 2, pp. 113–120, 2007.

[50] K. Taguchi, H. Nishihara, T. Aoki, F. Kumeno, K. Hayamizu, and K. Shinozaki, "Building a body of knowledge on model checking for software development," in *IEEE 37th Annual COMPSAC*, 2013.

[51] M. Pomeroy-Huff, R. Cannon, T. A. Chick, J. Mullaney, and W. Nichols, "The personal software process (psp) body of knowledge (bok)," Software Engineering Institute, http://www.sei.cmu.edu, Special Report CMU/SEI-2009-SR-018, August 2009.

[52] M. Aoyama, T. Nakatani, S. Saito, M. Suzuki, K. Fujita, H. Nakazaki, and R. Suzuki, "A model and architecture of rebok (requirements engineering body of knowledge) and its evaluation," in *17th APSEC*, 2010.

[53] D. Henry, A. Pyster, D. H. Olwell, N. Hutchison, S. Enck, and J. F. Anthony, "Experiences from creating the guide to the systems engineering body of knowledge (SEBoK) v. 1.0," *Procedia Computer Science*, vol. 16, pp. 990–999, 2013.

[54] N. C. L. Schots, A. R. Rocha, and G. Santos, "A body of knowledge for executing performance analysis of software processes," in *The 26th International Conference on Software Engineering and Knowledge Engineering*, 2014.

[55] W. S. Humphrey, T. A. Chick, W. Nichols, and M. Pomeroy-Huff, "Team software process (tsp) body of knowledge (bok)," Software Engineering Institute, http://repository.cmu.edu/sei/12, TECHNICAL REPORT CMU/SEI-2010-TR-020 ESC-TR-2010-020, July 2010.

[56] A. Agrawal, "University-to-industry knowledge transfer: literature review and unanswered questions," *International Journal of Management Reviews*, vol. 3, no. 4, pp. 285–302, 2001.

[57] U. Lichtenthaler, "Open innovation: Past research, current debates, and future directions," *The Academy of Management Perspectives*, vol. 25, no. 1, pp. 75–93, 2011.