VNIVERSIDAD NACIONAL AVTONOMA DE MEXICO

NATIONAL AUTONOMOUS UNIVERSITY OF MEXICO

GRADUATE COURSES IN COMPUTER SCIENCE AND ENGINEERING

IDENTIFICATION AND FORMALIZATION OF SOFTWARE PROJECT COMMON CONCEPTS

THESIS
PRESENTED IN CANDIDACY FOR THE DEGREE OF:
PHD (COMPUTER SCIENCE)

BY:
MIGUEL EHÉCATL MORALES TRUJILLO

SUPERVISOR
PHD. HANNA JADWIGA OKTABA, SCIENCE FACULTY, UNAM

CO-SUPERVISORS:
PHD. MARIO PIATTINI VELTHUIS, UNIVERSITY OF CASTILLA – LA MANCHA
PHD. FRANCISCO HERNÁNDEZ QUIROZ, SCIENCE FACULTY, UNAM

MEXICO CITY, MEXICO. AUGUST 2015

Identification and Formalization of Software Project Common Concepts

by

Miguel Ehécatl Morales Trujillo

MSCS, National Autonomous University of Mexico (2010)

Dissertation presented in candidacy for the degree of

PhD (Computer Science)

to the

Graduate Courses in Computer Science and Engineering

National Autonomous University of Mexico

Ciudad Universitaria, Mexico City, Mexico. August, 2015

I dedicate this work to my wife, Полина, and our children, Миша и Даша.
I also dedicate this work to my parents, Miguel Romo and Luz Trujillo, and
my sisters, Bárbara and Tania.

Acknowledgments

In moments like this, when you are achieving a very important personal and academic goal and trying to put in paper the words of appreciation and gratutude, it is hard to keep feelings aside. Many people supported and encouraged me during these years of PhD, but there are also people from other years of my life to whom I am grateful for what I have been able to achieve, who by telling me a word, teaching me something new, or standing next to me in the right moment or place made this work possible.

I am deeply thankful to my “academic mothers”, PhD. Hanna Oktaba and M.Sc. Guadalupe Ibargüengoitia. I sincerely appreciate all the time, guidance and advice they gave me during these years. Words cannot express how grateful I am, and I have it clear that this work would not have been equaled without them.

I would also like to express my special thanks to PhD. Mario Piattini, who was a very important part of this effort. He believed in me and my work and always supported it without hesitation. During my research stays at Ciudad Real he empowered my research and, most importantly, he taught and improved my research habits.

I also thank PhD. Francisco Hernández Quiroz, who guided me in a very tricky part of my research, and semester after semester was involved and truly interested in this work.

Thanks to the members of the examining committee: PhD. Felipe Lara Rosano and PhD. Fernando Gamboa Rodríguez, for reading this work carefully and making invaluable comments and suggestions to improve it.

I express my warm thanks to PhD. Francisco J. Pino, who made me feel like home during my first stay at Ciudad Real and introduced me to the world of academic research. I still remember him saying *“el que no publica, no existe”*, thank you Pino for showing me the multifaceted world of the Software Engineering research.

My special thanks to Goyi Romero, who was always aware of and attentive to any logistics of my academic stays in Spain.

I am thankful to the Alarcos Research Group members who invariably welcomed visitors like me.

I would also like to thank Diana Arias, Amalia Arriaga, Cecilia Mandujano y Lulú González, who keep PCIC running and in order.

My sincere thanks to PhD. Leobardo Fernández Román, who taught me the amazing world of mathematics. He made me more resistant to the hard world of Geometry and at the same time motivated me not to drop out of the Science Faculty.

My special thanks go to Alex Pérez Meléndez, who invited me to volunteer for a summer camp, so I found myself at Ribes de Freser (Catalonia) and met a woman who changed the essence of my life.

In addition, I want to thank PhD. Javier García García, PhD. Jorge Ortega Arjona and PhD. Iván Hernández Serrano. During my bachelor's and master's studies I enjoyed their lectures and learned a lot. My special thanks to PhD. Javier García García, who encouraged me to read research papers and, moreover, to be critical of them.

I would like to thank Mario Cruz-Terán (Calculus), Reyes Mata-Franco (Chemistry) and José Castañeda (Logic) from high school; Felipe Radilla (Physics), Miguel Moreno (Biology) and Rosa María Abarca (Geography) from junior high school; and Maria Alessandri, María Luna Vela and Martha Barrios from elementary school. Their commitment and vocation for teaching were second to none.

A doctor or researcher in any science is not complete without being able to guide others in their research. My sincere thanks to Martha, Jess, Erick, Daniel, Pablo and Efraín who believed in my guidance as a thesis adviser.

On a more personal side, I would like to thank my grandmother Juanita and my aunts Teté, Mati† and my uncle Artur. I am also thankful to my Russian family: Lidia, Alexander and Evgeny, for always being curious about "whatever-Miguel-is-doing" and for keeping asking me when I will finish.

My warm thanks to my old friends Rodrigo Blando and Roberto Uribe, who are like brothers to me. You have made me laugh so many times and were always with me when in need.

At IIMAS I have met many people who became important to me: thank you David Velázquez, Ricardo Cruz and Mauricio Morgado for making the master's an enjoyable place.

At the Science Faculty, a thank-you goes to the most laureated soccer team: Silvestres FC. Martín Ponce, Marco Camacho, Amilcar Martínez, Arturo Palacios, Alonso Alvarado, Alejandro Piña and Bragi Pérez, with whom running after a ball got a different sense: being a team, being friends.

I also want to express my gratitude to Tere Ventura, Marcela Peñaloza, Hugo Reyes, Alma García, José Luis Aguirre, Iran Velázquez and Heidi Vera from DGTIC-DCV, who are an actual definition of a perfect work team.

I would like to thank the Consejo Nacional de Ciencia y Tecnología (CONACyT) for providing the necessary financial support for this work.

Last but not least, I want to express my gratitude and appreciation to the persons who have always been there with me. In the first place, thanks to my parents: Miguel and Luz; all I am and all I have got is because of you, you taught me to always do the right thing, to be happy with who I am, to never quit, to take on new challenges and, most importantly, to be thankful for what I have. You supported my every step and here we are. Secondly, I want to thank my sisters, Bárbara and Tania. Although we are very different, we share the principles of who we are and in times of difficulty we survived and succeeded together as a family. Since I can remember the five of us have been the happiest family ever.

Finally, I am deeply grateful to Polina, who gives me the courage to do whatever I want no matter how difficult it is. The confidence that you transmit to me is the cornerstone of everything. Thank you for jumping with me into "crazy" projects of our life, I love you. My dear thanks to Misha and Dasha, whose smiles are my endless source of energy and with whom I am invincible.

Miguel Ehécatl Morales Trujillo, Mexico City, Mexico.

Table of Contents

List of Figures

List of Tables

Identification and Formalization of Software Project Common Concepts

by

Miguel Ehécatl Morales Trujillo

Abstract

Software Engineering knowledge is obtained by practitioners during software engineering efforts and represents a valuable source of knowledge with which to create and formulate theories. This knowledge is applied in organizations by practitioners, and even those practitioners with the simplest way of working follow tacit practices in order to carry out their work. However, these ways of working are frequently neither expressed nor collected in order to reason about their characteristics and properties. Moreover, the explicit ways of working, which are comprised in process reference models that follow a prescriptive top-down approach, are not suitable for small software development organizations.

This thesis presents KUALI-BEH, a bottom-up metamodel that provides software engineering practitioners with an authoring framework with which to express, adapt and share their ways of working as a collection of methods and practices.

KUALI-BEH is composed of two views: static and operational. The static view defines the common concepts needed for the definition of the practitioners' diverse ways of working. The operational view is related to the software project execution. It provides mechanisms with which to enact a method and adapt its practices to the specific context and stakeholder needs.

After validating the metamodel with a collaborative workshop, three case studies and five reviews by the Object Management Group Task Force, it was found that KUALI-BEH permits small organizations to create an organizational method repository comprising their knowledge, and to gradually introduce them to the adoption of standards and reference models. Furthermore KUALI-BEH has been used in an educational context with beneficial results, thus bringing Software Engineering theory and practice closer to bachelor degree students.

It is concluded that KUALI-BEH is a valuable alternative as a first step for practitioners and small organizations to reduce the gap between Software Engineering theory and practice.

Part I

Generalities

Chapter 1

Introduction

"Seamos realistas, soñemos lo imposible."
"Let's be realistic, let's dream the impossible."

Ernesto *Che* Guevara. Marxist revolutionary, physician, writer, guerrilla leader, diplomat and military theorist (1928 – 1967).

This chapter presents the motivation of this thesis, its hypothesis, research objective, sponsored research projects and how it is organized.

1.1 Background and motivation

Software Engineering was first discussed as a discipline during a NATO meeting in 1968 (Naur and Randell, 1969). Friedrich Bauer later conceived the first definition of the discipline as the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines (Bauer, 1971).

Software Engineering is by definition part of a practical discipline (Broy, 2011). Engineering disciplines must be based on scientific practices and theory in order to justify their approaches and to provide scientific evidence as to why and where their methods work properly (Broy, 2011).

However, Software Engineering is currently controlled and guided by the practices used in industry and technological innovations (Wang, 2007), while its theory and foundations, which are essential in supporting its practices, are left aside (Johnson et al., 2012).

According to (Perry et al., 2000), all large software projects follow some underlying development process that includes stages such as requirements definition, functional design, unit implementation, integration, and so on; but the way in which these stages are conducted, the tools that are used to support them and the rationale for doing so vary widely.

Moreover, the many differences between individual software projects make comparison difficult (Basili et al., 1986). In consequence, formulizing theories to base on practices carried out during software projects has become a challenging task.

These concerns in the discipline have motivated the creation and putting forward of various proposals such as: metamodels to create software and method processes; Method Engineering and Situational Method Engineering, which were conceived to build methods from pre-existing pieces of methods; the collection of ISO/IEC Software Engineering standards; and efforts like Software Engineering Method and Theory (SEMAT), whose objective is to concentrate and define the theory needed to re-found the discipline. SEMAT initiative has later become a standardization process that is supported by the Object Management Group (OMG). As a whole, these efforts work on creating theories that are eagerly sought by Software Engineering.

Nevertheless, these proposals fit bigger organizations better and become an obstacle when the actual target audience is mainly composed of small organizations that do not follow models or standards and have low levels of maturity. According to (Oktaba and Piattini, 2008), 58% of organizations working on significant software products are small organizations. This reinforces the need for strategies for process improvement that are tailored to small companies' characteristics (Pino et al., 2010).

Some of the practices needed by the discipline are contained in standards and process reference models, but there is evidence that the majority of small software organizations are not adopting existing standards, as they perceive them to be oriented towards large organizations (Laporte et al., 2008). These standards per se are not suitable for small organizations (Pino et al., 2010).

In practice, small organizations predominantly acquire knowledge through observations, which are obtained during software engineering efforts such as projects, experiments or case studies. Practitioners then apply these observations to their ways of working. These practitioners consequently need an effective thinking framework that will bridge the gap between their current ways of working and any new ideas that they wish to adopt (Jacobson et al., 2012) within their organizations.

Seeking to resolve these issues, this PhD thesis proposes KUALI-BEH, a bottom-up framework for software engineering practitioners working in small organizations with which they can formulate, progress, adapt and improve their ways of working in the form of practices and methods.

The objective of KUALI-BEH is to become a starting point between practice and theory in Software Engineering, and its main focus is on practitioners. On the one hand, KUALI-BEH can be applied as a framework with which to author software engineers' tacit ways of working, and it supports practitioners in the reasoning about their characteristics and properties. This is a first step towards the incorporation of practitioners into theories and metamodels that sustain the Software Engineering discipline.

On the other hand, it permits small organizations to create an organizational method repository that comprises their knowledge, and to gradually introduce them to the adoption of standards and reference models.

Besides, this generated knowledge, once validated and approved by the software engineering community, could form a collection of methods and practices recommended for educational purposes in Software Engineering courses.

As (Basili et al., 1986) mentioned almost 30 years ago "we greatly need to improve our knowledge of how software is developed", and KUALI-BEH, presented in this thesis, is a proposal towards achieving this ambitious goal.

1.2 Hypothesis and objective

In the context of software developer organizations arises a question: Which model is pertinent in order to represent, adapt, share, compare and execute their ways of working? Guided by this research question the main hypothesis stated for this research is:

It is possible to support and improve the representation, adaptation, sharing, comparison and execution of the practitioners' ways of working through a kernel of common concepts.

According to this hypothesis, the main objective of our research is, therefore:

To define a kernel of common concepts/framework that supports the expression of practitioners' ways of working carried out during software projects.

The achievement of the main objective will be based on the achievement of the following Goals (Gs):

G1. To identify the universals of software projects carrying out an in-depth study in Method Engineering, Situational Method Engineering, Process Reference Models, Process Metamodels and related Standards.

G2. To define a kernel of common concepts which facilitates the transformation of practitioners' tacit knowledge into explicit knowledge.

G3. To formalize the description of methods as a composition of practices using the defined language of common concepts.

G4. To establish a conceptual framework that supports understanding between concepts, terms and relations around software projects.

G5. To validate the framework through case studies.

1.3 Sponsored research projects

This research has principally been developed within the framework of the following research projects:

- The national and mixed grants given by the Consejo Nacional de Ciencia y Tecnología (CONACyT) of Mexico.

- The Programa de Apoyo a Estudiantes de Posgrado (PAEP) financed by the National Autonomous University of Mexico (UNAM).

- The GEODAS-BC project financed by the Ministerio de Economía y Competitividad of Spain and the Fondo Europeo de Desarrollo Regional (FEDER) (TIN2012-37493-C03-01). This project was carried out by the University of Castilla – La Mancha under the direction and support of the Alarcos Research Group.

- The GLOBALIA project financed by the Consejería de Educación, Ciencia y Cultura (Junta de Comunidades de Castilla – La Mancha) of Spain and FEDER (PEII11-0291-5274). This project was carried out by the University of Castilla – La Mancha under the direction and support of the Alarcos Research Group.

- The SDGear project framed under the ITEA 2 Call 7, and co-funded by Ministerio de Industria, Energía y Turismo (Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2013-2016) and FEDER (TSI-100104-2014-4). This project was carried out by the University of Castilla – La Mancha under the direction and support of the Alarcos Research Group.

1.4 Thesis structure

This document consists of 5 parts, 10 chapters and 4 appendixes. The contents of the remainder of this document are as follows.

- Part I – Generalities: consists of introduction (Chapter 1), the research method which was used to achieve the objective and goals defined for this thesis (Chapter 2) and the state of the art of the research area (Chapter 3).

- Part II – KUALI-BEH Proposal: presents the process followed to identify the common concepts that belong to KUALI-BEH (Chapter 4). The full explanation of the basis of KUALI-BEH framework and its common concepts (Chapter 5) and the formalization of the kernel properties and operations using an ontology and Description Logic (Chapter 6).

- Part III – KUALI-BEH Validation: presents the evolution of the research through the application of Technical-Action-Research method in order to qualitatively demonstrate whether KUALI-BEH fulfills its objective and goals. The validation included a collaborative workshop (Chapter 7), a set of case studies (Chapter 8) and a standardization process related to OMG (Chapter 9).

- Part IV – Conclusions: presents an analysis of the research goals, the main contributions, the results and the future lines of the research (Chapter 10).

- Part V – Appendixes: consist of:
  - Appendix A: shows the KUALI-BEH Extension to ESSENCE formal specification.
  - Appendix B: shows the list of references and sources used to create the common concepts definitions.
  - Appendix C: shows the issues related with an alternative naming of common concepts, SPEM and MOF.
  - Appendix D: shows the list of acronyms used in this thesis.

Chapter 2

Research Method

“Учиться, учиться, учиться.”
“Learn, learn and learn.”

Vladimir Ilyich Lenin. Communist revolutionary, politician and political theorist (1870 – 1924).

This chapter presents the research methods that were used to achieve the objectives defined in this thesis. The methods applied were Technical-Action-Research and Case Study.

2.1 Research methods in Software Engineering

Software Engineering requires both theoretical and empirical research. The former focuses on foundations and basic theories of software engineering; whilst the latter concentrates on fundamental principles, tools/environments, and best practices (Wang, 2007).

As mentioned by (Harrison et al., 1999), Software Engineering should have strong foundations as a scientific and engineering discipline. This implies that validation processes must be mature enough to provide evidences that support its advances. An alternative to achieve that goal is the empirical software engineering, according to (Belady and Lehman, 1976) this type of research offers the opportunity to build and verify its theories.

Validation in Software Engineering must always involve scaling up to practice, which means that successive tests take place under increasingly realistic conditions (Wieringa, 2014). No

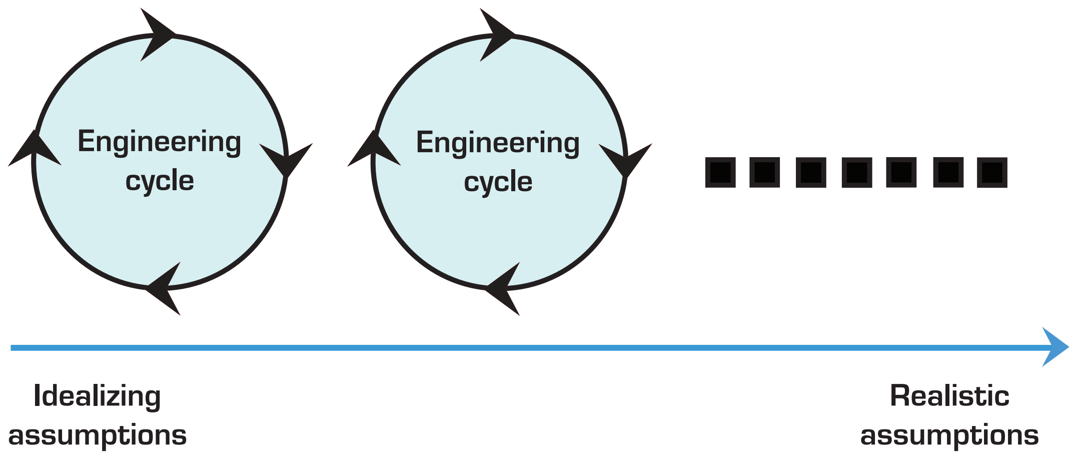


Figure 2-1: Iterating through the engineering cycle, adapted from (Wieringa and Moralı, 2012).

matter what its form is, the essence of an empirical study is the attempt to learn something useful by comparing theory to reality and to improve our theories as a result (Perry et al., 2000).

For the purposes of this thesis, the more relevant research methods are described in the next sections.

2.2 Technical-Action-Research method

Technical-Action-Research (TAR) is an approach to validate new artifacts under conditions of practice (Wieringa and Moralı, 2012). In TAR the researcher uses an artifact in a real-world project to help a client, or gives the artifact to others to use it in a real-world project (Engelsman and Wieringa, 2012).

TAR can be seen as a research method that starts from the opposite side of traditional research methods. TAR starts with an artifact, and then tests it under conditions of practice by solving concrete problems with it (Wieringa and Moralı, 2012).

In this method the researcher develops the artifact, tests it on an hypothetical situation inside laboratory and then scales it to be tested in real-world situations, from its idealized assumptions to the real ones, see Figure 2-1.

In order to clarify how TAR lifecycle is, it can be compared with Medicine, where vaccines testing process first occurs under artificial conditions in a laboratory, then they are applied to healthy volunteers and lastly to ill volunteers, proving if they solve the problem for which they had been created.

In technical disciplines, prototypes of artifacts are first tested in the idealized conditions of the laboratory; after each iteration these conditions are gradually relaxed, andl a realistic version of the artifact is tested in a realistic environment, allowing technical scientist to develop knowledge about the behavior of artifacts in practice (Wieringa and Moralı, 2012).

Those iterations are called *engineering cycles* (Wieringa and Moralı, 2012) or *regulative cycles* (Van Strien, 1997). In an engineering cycle an improvement problem is investigated, then a treatment to solve the problem is designed and validated, later its improved version is implemented, finally the implementation experience is evaluated, and the cycle is executed again (Wieringa and Moralı, 2012).

Each engineering cycle consists of four steps, see Figure 2-2:

1. **Problem Investigation:** in this step the stakeholders, their goals and the improvement problem are identified.

2. **Treatment Design:** in this step the researcher designs an artifact that will be used as a treatment for the problem. According to (Wieringa and Moralı, 2012) artifacts may consist of software, hardware or conceptual entities such as methods and techniques or business processes.

3. **Design Validation:** after the artifact was designed, the researcher proceeds with its validation, inserting it into the context. The researcher must state the expected effects of introducing the artifact and how those effects will satisfy the stakeholders needs. (Wieringa and Moralı, 2012) establishes that, before the implementation of the treatment, four questions must be asked:

   - What will be the effects of the artifact in a problem context? This question will determine the *expected effects.*

   - How well will these effects satisfy the criteria? This question will state the *expected value.*

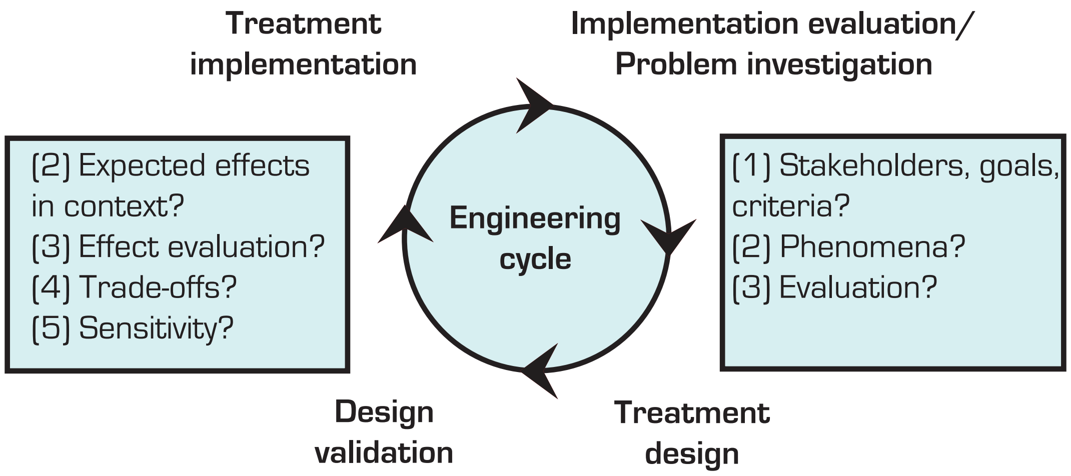


Figure 2-2: Engineering cycle, adapted from (Wieringa and Moralı, 2012).

- How does this treatment perform compared to other possible treatments? This question includes comparison between reduced versions of the treatment, as well as between existing treatments. It will point out the *trade-offs* between treatments.
- Would the treatment still be effective and useful if the problem changes? It will establish the *sensitivity*.

4. **Treatment Implementation and Evaluation:** this step consists in the insertion of the artifact into real world. At that moment the artifact can be validated using the defined above questions, and the researcher can compare obtained effects, value, trade-offs and sensitivity of the respective engineering cycle.

TAR is based on the assumption that what the researcher learns in this particular case will provide lessons learned that will be usable in the next case (Wieringa and Moralı, 2012). The expected result from TAR is the validation of the proposed artifact in a specified context, satisfying the stakeholders needs and giving value to it.

2.3 Case Study research method

A case study is an intensive investigation and analysis of a particular technology, project, organization, or environment based on information obtained from a variety of sources such as interviews, surveys, documents, test or trial results, and archival records (Wang, 2007). Wang establishes that case studies link a theory to practice, which allows conclusions to be drawn about the suitability of a given method on real-world problems in industrial scales (Wang, 2007).

Case studies, according to (Yin, 2009) is *an empirical inquiry that investigates a contemporary phenomenon in depth and within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.*

The essence of a case study, the central tendency among all types of case study, is that it tries to illuminate a decision or set of decisions: why they were taken, how they were implemented, and with what result (Schramm, 1971).

Case studies may be used to validate a theory or method by empirical tests. They are also useful for providing a counter instance for a generally accepted principle. However, the drawback of case studies as an empirical method in Software Engineering is the difficulties of data collection and the generalization of findings via limited cases, particularly when they are positive but non-exhaustive (Wang, 2007).

In addition, there are differences between a case study in Social Sciences and a case study in Software Engineering, which rely on the properties of the study objects present in each discipline. According to (Runeson et al., 2012) Software Engineering study objects have the following properties:

- They are private corporations or units of public agencies *developing* software rather than public agencies or private corporations *using* software systems.
- They are *project-oriented* rather than *function-oriented* organizations.
- The studied work is an *advanced* engineering work conducted by highly educated people, rather than a *routine* work.
- There is an aim to improve the engineering practices, which implies that there is a component of design research.

Based on the previously explained ideas, a case study in Software Engineering is *an empirical inquiry that draws on multiple sources of evidence to investigate one instance (or a small*

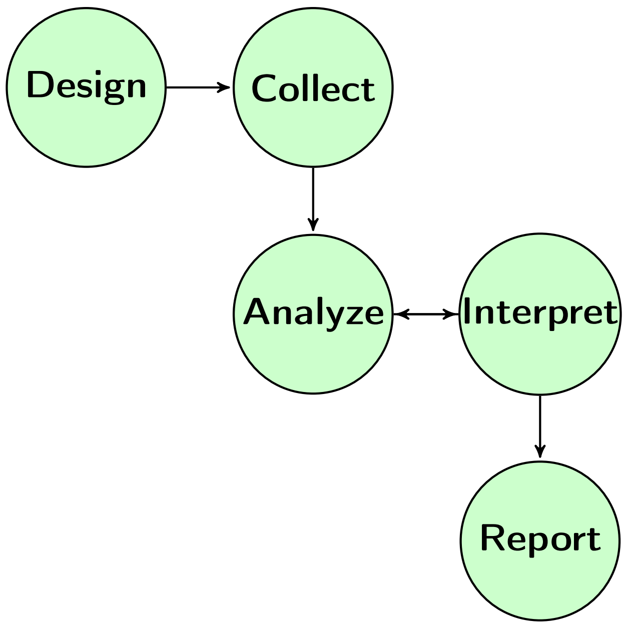


Figure 2-3: Case study phases, based on (Runeson et al., 2012).

number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified (Runeson et al., 2012).

The generic phases of case studies are: Design, Preparation, Data Collection, Data Analysis and Interpretation, and Sharing (or Reporting and Dissemination). Figure 2-3 shows the Software Engineering oriented approach described by (Runeson et al., 2012).

2.3.1 Designing the case study

During the Design phase of the study, the researcher describes the roadmap which will guide the process to prepare, collect, analyze and share the research.

2.3.2 Preparing the case study

During this phase, the researcher writes a protocol of the future study. According to (Yin, 2009) a case study protocol should have:

1. An overview of the case study.

2. Field procedures.
3. Case study questions.
4. A guide for the case study report.

2.3.3 Collecting the data

After the protocol of the study is approved, the study starts and the data can be collected. A first step is to identify the sources from which data can come, (Runeson et al., 2012) highlights six sources organized in three levels.

1. **First degree:** While using these methods the researcher is in direct contact with the participants, collecting data in real time.
    - Interviews.
    - Focus groups.
    - Delphi surveys.
    - Direct observations.
2. **Second degree:** Using these methods the researcher collects data in without interacting with participants during the process.
    - Software monitoring tools.
    - Video recording.
3. **Third degree:** Using these methods the researcher analyzes available work artifacts in an independently manner.
    - Work artifacts.

In order to maximize benefits from evidence sources, (Yin, 2009) establishes three principles to be followed, no matter what source will be used:

- **Principle 1:** Use multiple sources of evidence.
- **Principle 2:** Create a case study database.

- **Principle 3:** Maintain a chain of evidence.

Once the data is collected, it is time to analyze it.

2.3.4 Analyzing and interpreting the data

The data analysis must be supported by a strategy, and again (Yin, 2009) proposes four strategies to analyze the data collected during the case:

- Relying on theoretical strategies.
- Developing a case description.
- Using both qualitative and quantitative data.
- Examining rival explanations.

(Yin, 2009) and (Runeson et al., 2012) concur on the following techniques to support the data interpretation:

- Pattern matching.
- Explanation building.
- Time-series analysis.
- Logic models.
- Cross-case synthesis.

2.3.5 Reporting and disseminating the results

After having analyzed the evidence, the case study results must be reported to stakeholders. The first step is to identify the potential audience and the second step is to follow a structure to prepare a report, in this case: Linear-Analytic.

According to (Yin, 2009) the Linear-Analytic structure is defined as a sequence of subtopics that starts with the issue or problem being studied and a review of the relevant prior literature. The subtopics then proceed to cover the methods used, the findings from the data collected and analyzed, and the conclusions and implications from the findings.

In order to compose a solid report, the researcher should consider the checklist provided by (Yin, 2009) with such general characteristics of an exemplary case study as:

- The case study must be *significant*.
- The case study must be *complete*.
- The case study must consider *alternative perspectives*.
- The case study must display *sufficient evidence*.
- The case study must be composed in an *engaging manner*.

2.3.6 Threats to validity

The validity of a study denotes the trustworthiness of the results, and to what extent the results are not biased by the researchers' subjective point of view (Runeson et al., 2012). The validity evaluation of the case study must be done during the analysis phase; however, since the very beginning of the case study it must be monitored.

Four aspects of validity are distinguished by (Runeson et al., 2012):

- **Construct validity:** This aspect of validity reflects to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated according to the research questions.
- **Internal validity:** This aspect of validity is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor, there is a risk that the investigated factor is also affected by a third factor. If the researcher is not aware of the third factor and/or does not know to what extent it affects the investigated factor, there is a threat to the internal validity.
- **External validity:** This aspect of validity is concerned with to what extent it is possible to generalize findings, and to what extent the findings are of interest to other people outside the investigated case. During analysis of external validity, the researcher tries to analyze to what extent the findings are of relevance for other cases. The context description in the report can help the reader understand the relevance of the study for another situation.

- **Reliability:** This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. Hypothetically, if another researcher later on conducted the same study, the result should be the same.

2.3.6.1 Approaches to improve validity

According to (Runeson et al., 2012) the most important approaches in order to avoid serious threats are:

- **Prolonged involvement:** A long-term relation between the researchers and the organization could result in a trustful relation between the researchers and the organization. In consequence, the researcher can get a deeper understanding of participants intents and purposes. Also, the time slot that participants will provide data increases.

  In contrast, the researcher may lose independence or gets biased by being too involved in the organization.

- **Triangulation:** Involving more than one researcher in the data collection process reduces the risk of being biased. Also it allows researchers to analyze different data sources, which can improve the reliability.

- **Peer debriefing:** To work with a group of researchers in the case study instead of working alone can help to avoid bias and improves the understanding of the participants, since a group of researchers together can know more about the subject and a studied organization than what is possible for a single researcher.

  Another form of peer debriefing is to involve research colleagues in reviewing documents such as the research design.

- **Member checking:** The material generated during the case study can be given to participants so they can review it. They can also check that there is no misunderstanding or misinterpretation of what they have said or done. This approach could improve the reliability since the risk of having fault in the interpretations is lowered.

- **Negative case analysis:** Formulating alternative explanations and theories in the analysis can improve the analysis. This is a normal step in the analysis, which is valuable.

- **Audit trail:** It consists in keeping traceability of all data and material generated during the case study. Defining a version control strategy of all generated documents and work products and managing them adequately will provide researcher with a valuable chain of evidence.

2.4 Research strategy of this thesis

The research process in this thesis was guided by the integration of TAR and Case Study methods. TAR method was used as a framework in which the Case Study method was applied during engineering cycles to evaluate the artifact .

The research process was made up of five engineering cycles, see Figure 2-4. The validation of these engineering cycles was carried out through case studies primarily developed in real-world software development organizations. In addition the artifact was also inserted in contexts formed by practitioners and experts from academy and industry, such as a collaborative academy-industry workshop and the OMG Analysis and Design Task Force in charge of developing IT standards.

The first engineering cycle (from August 2011 through February 2012) consisted in the identification of common concepts used in the context of a software project and involved a literature review. The description of this review and its more relevant findings are presented in detail in chapter 3. Also, chapter 4 narrates the identification process of the common concepts.

During the second engineering cycle (from March through August 2012) an initial version of the artifact was released and validated through a collaborative workshop, whose participants were active practitioners of Software Engineering, master degree students and researchers of the discipline.

From September through December 2012 the third engineering cycle took place. A first case study was carried out during these months in which an improved version of the artifact was validated in a Mexican enterprise in charge of software development and hardware construction. Also the artifact was presented and evaluated by an OMG task force.

Later on, the fourth cycle started in January and finished in July 2013, with the constant improvement of the artifact in accordance with lessons learned from the previous cycles. The

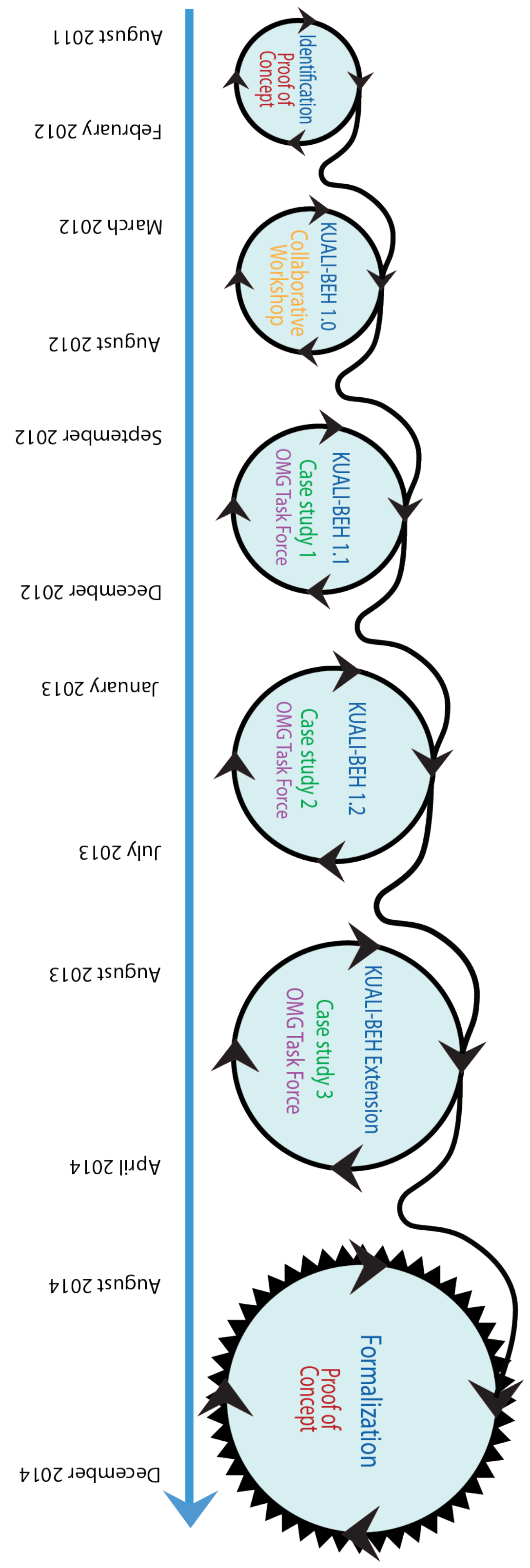


Figure 2-4: Research process followed in this thesis.

artifact got involved in a second case study, which in that occasion took place in an entity specialized in requirements specification and software projects design.

Then, a fifth engineering cycle took place between August 2013 and April 2014. During this cycle the artifact was enriched with an extension, which now belongs to a closely related OMG formal specification. At that point the artifact was newly validated by the OMG task force, and at the same time, a third case study was carried out in a very small entity in charge of software development.

A detailed and deep description of the final version of the artifact is presented in chapter 5, while Part III presents validation steps (the workshop, three case studies and the OMG process), resulted from the application of the integrated research method.

The final engineering cycle was designed with the aim to formalize the software project common concepts based on the final version of the artifact, and it took place from August to December 2014. It is important to mention that the approach of this engineering cycle was not TAR, since the artifact was not inserted into a real context. However, the generated during this last engineering cycle artifact was validated theoretically through a proof of concept developed by method engineers. A detailed description of the formalization process is presented in chapter 6.

2.5 Conclusions

The research strategy presented in this chapter is the result of the integration of TAR and Case Study methods. We took advantage of engineering cycles, which were used as the means to create and validate the artifact in a real context, allowing us to minimize disadvantages of a full empirical investigation that can be very disruptive and time-consuming for the company involved according to (Harrison et al., 1999).

On the one hand, making the artifact available to critical reference groups formed by theoreticians and practitioners provided us with fruitful feedback and a wide range of points of view. That knowledge became lessons learned, which were applied during the treatment design phase of each engineering cycle.

On the other hand, the application of case studies was meaningful in order to establish the sensitivity of the artifact and whether it satisfied the stakeholders' expected value and the hypothesis of the present research in a real world scenario.

This research strategy allowed us to generate KUALI-BEH framework, guided us through the validation process and helped us to achieve the objective and goals set for this thesis.

Chapter 3

State-of-the-Art

"Desconfíen vuestras mercedes de quién es lector de un solo libro."
"Never trust a man who reads only one book."

Diego Alatriste y Tenorio. Fictional soldier and swordsman created by Arturo Pérez-Reverte (1582 – 1643).

This chapter presents the general background of the state-of-the-art surrounding the description and modeling of software processes.

3.1 Software process metamodels

The importance of creating software process descriptions with which to guide key software process was highlighted by Osterweil in the well-known paper Software processes are software too (Osterweil, 1987) in the late 80's. Later, from the 90's on, there has been an increasing interest in developing proposals for process modeling. Among various proposals, there are several metamodels with similar purposes, such as Process Interchange Format (PIF) (Lee et al., 1998) and Process Specification Language (PSL) (Schlenoff et al., 1996), whose objective is to provide a standard exchange format in order to share process models between organizations.

Other existing metamodels are: Entry-Task-Validation-Exit method (ETVX) (Radice et al., 1999), Integration Definition for Function Modelling (IDEF0) (NIST, 1993), Core Plan Presentation (CPR) (Pease and Carrico, 1997), Shared Planning and Activity Representation (SPAR) (Tate, 1998), PROMENADE (Franch and Ribó, 1999) and SPEARMINT (Becker-Kornstaedt et al., 2000).

There are consequently many proposals with which to model software processes, and this implies a high degree of heterogeneity and requires greater standardization. With that goal in mind, two proposals (described below) can be highlighted.

The first proposal under consideration is the Software and Systems Process Engineering Meta-model (SPEM) (OMG, 2008b), defined in 2008 by OMG. SPEM is both a framework and an engineering process metamodel. This metamodel provides the concepts needed to model, document, manage, share and represent methods and processes. SPEM describes a language and a representation scheme that formally expresses methods and processes.

The drawbacks of the proposal are that the SPEM semiformal architecture makes the verification of a created statement extremely difficult (Krdžavac et al., 2009), and that SPEM is focused on the definition of a process, while aspects of execution such as sequence and the order of method components are sidelined.

The second proposal under consideration is the ISO/IEC 24744 standard (ISO/IEC, 2007b), aka the Software Engineering – Metamodel for Development Methodologies (SEMDM), which provides a framework for the definition and extension of methodologies in information-based domains, such as software development methodologies. Method engineers who define methodologies are the target audience of SEMDM, which attempts to facilitate communication between practitioners and engineers. SEMDM includes three main aspects: the process to follow, the products, used and generated, and the people involved.

Despite the advantages of formalism and structure of the SEMDM metamodelling approach, the concepts proposed are not familiar to practitioners and are distanced from their context. For example, the term *clabject* mixes the idea of class and object, leading to confusion and lack of use among practitioners.

Finally, SEMDM and SPEM are top-down approaches which are principally focused on method engineers and prescribe practitioners with ways of working. In contrast, the bottom-up approach of KUALI-BEH permits the description of ways of working and encourages synergy between practitioners and method engineers.

3.2 Method Engineering

Method Engineering is an information systems discipline that is in charge of constructing new methods from existing methods (Harmsen and Saeki, 1996). It focuses on the design, construction and evaluation of methods, techniques and support tools for information systems development (Brinkkemper, 1996).

This discipline describes a mechanism with which to build a method by starting from previously defined method fragments. In principle, any coherent product, activity, or tool that is part of an existing generic or situational method is a method fragment (Harmsen et al., 1994).

According to (Harmsen et al., 1994), Method Engineering involves the harmonization and standardization of methods. On the one hand, standardization relies on collecting proven and consensual knowledge within a community to which all parties must adhere, resulting in the definition of process reference models and standards.

On the other hand, the harmonization of methods consists in building up methods by using existing proven method fragments as building blocks. These methods are tuned according to the situation in which they are applied.

These building blocks are existing method fragments which are stored in an organizational repository. One drawback of this premise is that an in-depth means to create an initial set of method fragments is not provided.

KUALI-BEH can be used as a framework to identify and express method fragments as practices to be stored in a repository. Method Engineering can later make the most of these building blocks and construct methods.

3.3 Process reference models and standards

Process reference models or standards, such as CMMI (SEI, 2010), ISO/IEC 12207 (ISO/IEC, 2008a), ISO/IEC 15504 (ISO/IEC, 2004), and ISO/IEC 29110 (ISO/IEC, 2011) contain validated and approved knowledge to be applied by practitioners. However, this knowledge may undergo transformations when in use. According to (Osterweil, 1987) a static process description is constructed to specify a collection of dynamic processes. A process may consequently be performed in many ways, but not all of them are correct.

It is therefore crucial to enclose and express the process instances in a homogeneous manner, in order to analyze them and decide which are valid and which are not. Homogeneity between standards has not yet been achieved, and in (Rout, 1999) the need to align the large number of definitions stated in standards is reinforced, particularly in SC7 in which 198 of the 864 definitions were duplicated or contained overlapping terms (Henderson-Sellers et al., 2014).

All of the above makes the transition complex for both practitioners and the organization, thus increasing the effort required to understand, adopt and train a new model, which is a time- and cost-consuming task, particularly for small organizations.

The top-down strategy of standard adoption in organizations forces practitioners to adapt or replace their actual ways of working in order to conform to a standard or model correctly. The prescriptive nature of process reference models and standards lessens the expression of their acquired knowledge and halts their actual practices.

What is more, there is a growing concern in the Software Engineering community that SEI and ISO standards are not easily applicable to small firms because they require a huge investment (in time, money, and resources) (Pino et al., 2008).

The use of a bottom-up approach would therefore enable practitioners to author their own ways of working, thus reducing the aforementioned difficulties.

3.4 Software Engineering Method and Theory

SEMAT, a major new initiative for the purpose of generating a theoretical basis for software engineering, first appeared in 2009. SEMAT includes several influential members of the software engineering community.

SEMAT currently supports the process of re-founding software engineering on a solid theoretical basis with proven principles and better practices that will allow a common language to be conceived. This language will define the practices and patterns of which new methods will be composed, which could in the future be compared under a particular evaluation mechanism (Johnson et al., 2012; Jacobson et al., 2009, 2012, 2013).

The expected milestones were: to determine a set of definitions of the fundamental concepts needed by practices, to identify specific theories backed by examples of their successful application, to elucidate a set of universals and a kernel language with which to describe them, and to establish a set of metrics that can be used to assess software practices, products and people (Jacobson et al., 2010).

SEMAT has been endorsed and accepted by the OMG as a standard project: A Foundation for the Agile Creation and Enactment of Software Engineering Methods RFP (FACESEM) (OMG, 2011b). Since 2011, the OMG Task Force has been developing a proposal in response to the RFP. As a result, two proposals have been presented: ESSENCE – Kernel and Language for Software Engineering Methods (OMG, 2013) and KUALI-BEH: Software Project Common Concepts (Morales-Trujillo and Oktaba, 2012a).

After two OMG Task Force reviews, a joint submission was generated. In this joint submission, KUALI-BEH's main contributions are: an improved and refined definition of Method and Practice concepts as an alternative by which to characterize both concepts using the kernel and language elements, formal definition of method properties, operations of adaptation for method tailoring and a board-based approach with which to enact and manage projects using the proposal in small organizations.

In March of 2014 the joint submission was voted for and became an OMG formal specification.

3.4.1 ESSENCE – Kernel and Language for Software Engineering Methods

The standard project, initiated by SEMAT, resulted in ESSENCE, a four layer approach that defines a kernel and a language to build software engineering methods. Figure 3-1 shows the architecture of ESSENCE.

The ESSENCE kernel is a set of indispensable components involved in software engineering efforts, paired with a language to express them through the syntax and semantics defined.

The path that guided the definition of ESSENCE was the separation of concerns, and it therefore defines three areas of concern: Customer, Solution and Endeavor. Each area of concern defines an activity space that is a place in which practitioners can model the state of a particular software project with help of a set of Abstract-Level Project Health Attributes (Alphas; see

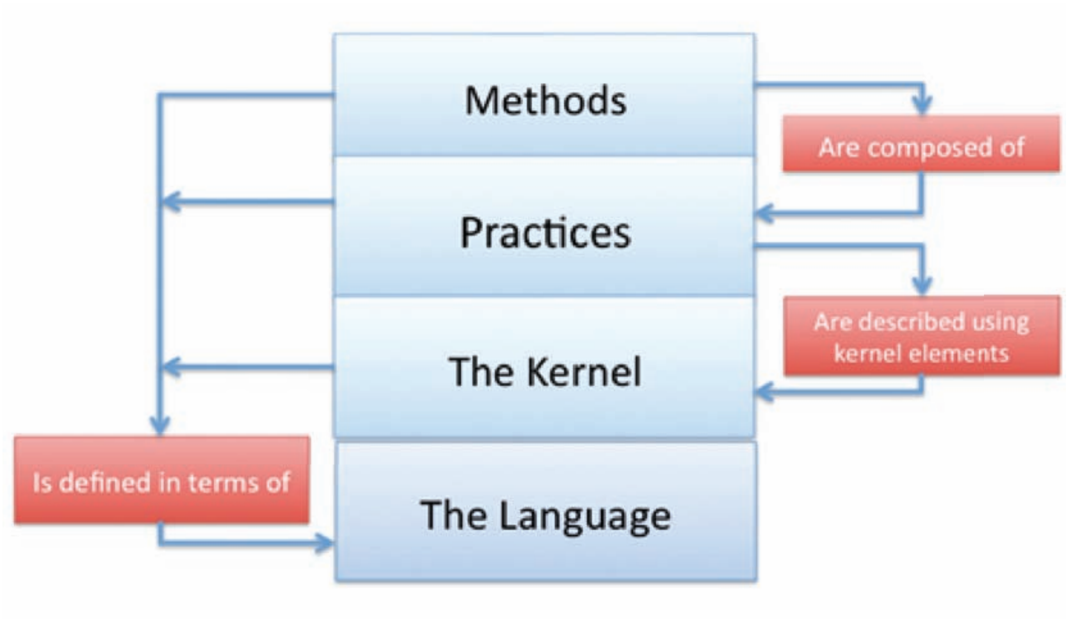


Figure 3-1: ESSENCE architecture (OMG, 2013).

Figure 3-2.

For the purposes of this thesis, the Endeavor area of concern will be relevant, for it contains elements related to how a team carries out its work according to its own way of working. The Alphas associated with the Endeavor area of concern are defined as follows (OMG, 2013):

- **Work:** An activity involving mental or physical effort exerted in order to achieve a result. In the context of software engineering, work is everything that the team does to meet the goals of producing a software system matching the requirements, and addressing the opportunity, presented by the customer. The work is guided by the practices that make up the team's way-of-working.

- **Way-of-Working:** The tailored set of practices and tools used by a team to guide and support their work. The team members evolve their way of working alongside their understanding of their mission and their working environment. As their work proceeds, they continually reflect on their way of working and adapt it as necessary to their current context.

- **Team:** A group of people actively engaged in the development, maintenance, delivery

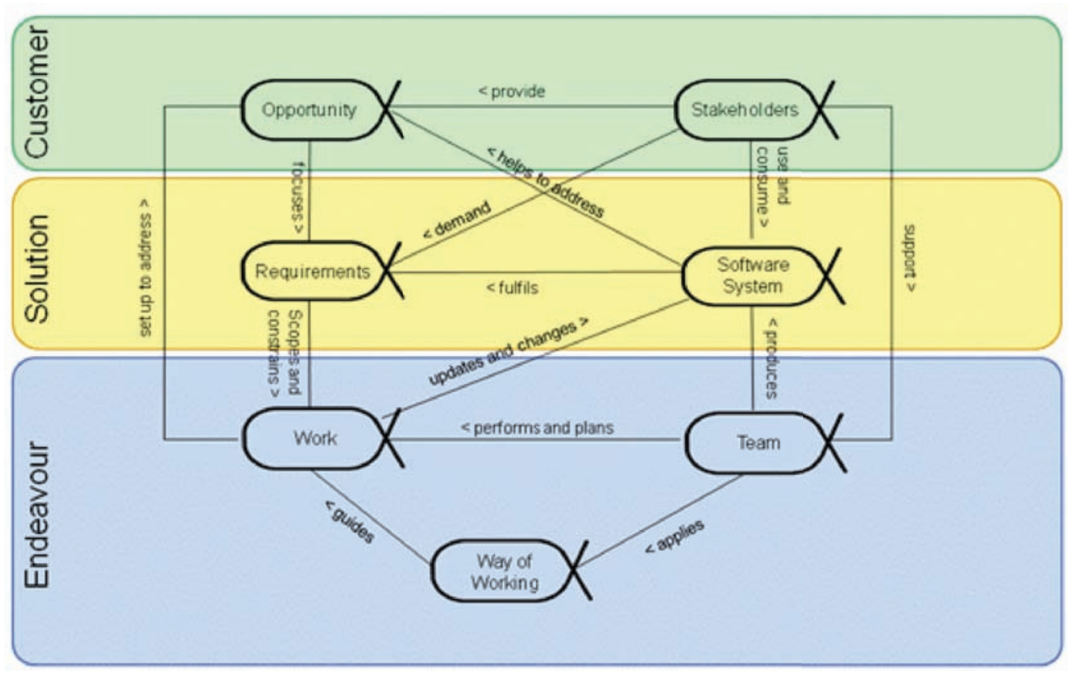


Figure 3-2: ESSENCE Areas of Concern and its Alphas (OMG, 2013).

or support of a specific software system. One or more teams plan and perform the work needed to create, update and/or change the software system.

In particular, the Endeavor area of concern comprises the practitioners' practical knowledge, but it does not denote how to express it.

3.5 Conclusions

This chapter analyzed and classified proposals that are suitable to model software processes. The obtained results showed a high degree of heterogeneity between proposals and affirmed that standardization is needed but is a hard goal to achieve.

The identified drawbacks are distributed along the whole modeling process: definition, evaluation, execution and improvement. On the one hand, in some cases process definition becomes complicated since the very beginning because the proposed concepts are not familiar to practitioners, so they are forced to do an extra effort to understand them and then adapt them to the context. On the other hand, there are proposals that sideline aspects of evaluation and/or execution of the modeled process, yet these tasks are essential for practitioners. These factors may affect the decision of using or abandoning such proposals. Moreover, the top-down approaches prescribe practitioners not only with ways of working, but also with new terms and definitions that vary between standards, leading to confusion and reluctance to use.

We can conclude that the top-down strategy followed by the majority of the proposals, such as standards, metamodels and process reference models, forces practitioners to adapt or replace their actual ways of working in order to follow the new standard. This prescriptive approach affects directly the spreading, sharing and improvement of actual ways of working obstructing such a valuable source of knowledge as the lessons learned by Software Engineering practitioners.

These findings motivated the origin of KUALI-BEH, a metamodel composed of two views: static and operational. The static view describes the common concepts that practitioners need in order to define their ways of working. These ways of working are first expressed through practices and then compiled into methods. The operational view is related to the software project execution and helps work teams to enact a method and to adapt its practices to a specific context. The target audience of this proposal is Software Engineering practitioners who can accumulate and share their knowledge with the help of KUALI-BEH's bottom-up approach.

Part II

KUALI-BEH Proposal

Chapter 4

Identification

"Whenever you find yourself on the side of the majority, it is time to pause and reflect."

Mark Twain. Writer (1835 – 1910).

This chapter details the Identification phase, which corresponds to the first engineering cycle of the research. This phase presents how KUALI-BEH common concepts were identified, defined and modeled.

4.1 Creating the KUALI-BEH proposal

In 2010 KUALI-KAANS, our research group, took the SEMAT Call for Action seriously and created a first effort called Software Development Projects Kernel (SDPK) (Oktaba and Dávila, 2011). SDPK defined work units, which are atomic elements used to describe practices that a work team executes in a software endeavor. This work was presented in 2011 at the Latin American Symposium of Software Engineering (LASES'11) in Medellin, Colombia. LASES'11 was also the event at which the establishment of SEMAT's Latin American chapter was announced by Ivar Jacobson.

As a parallel effort, after OMG FACESEM RFP was developed and made publicly available in 2011, the development of KUALI-BEH began. In contrast to SDPK which responded to the SEMAT call, the KUALI-BEH project aimed to respond to the FACESEM RFP requirements.

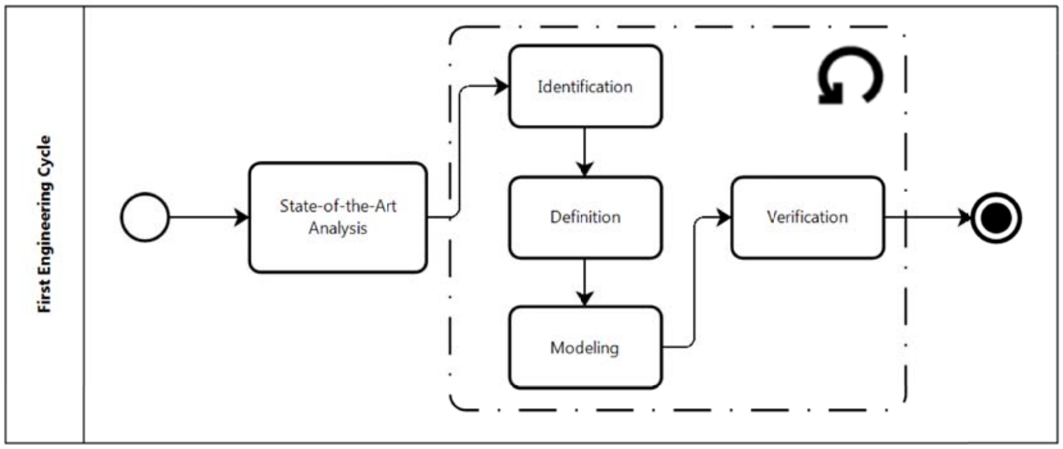


Figure 4-1: Steps developed during the first engineering cycle of the research.

The goal was to identify a set of common concepts that are involved in software projects, and later use them to express and structure Software Engineering methods. In order to carry out the project, a first engineering cycle was executed, see Figure 4-1.

This first cycle started with a review of the state-of-the-art, and the review's outcome served as starting point for the identification, definition and modeling of common concepts. Those three activities were developed under an iterative approach together with a theoretical verification of the results. Each activity is detailed in the next subsections.

4.1.1 State-of-the-Art analysis

In order to attain an in-depth knowledge of and analyze existing efforts surrounding the topic, the first step was to study the state-of-the-art, starting with the RFP itself. As part of all RFPs, the OMG requests that a relationship be maintained with the existing OMG specifications and activities, and FACESEM RFP particularly highlighted four of them. The first was an already adopted specification, the Software and System Process Engineering Metamodel (OMG, 2008b).

SPEM guided us to the Eclipse Process Framework (EPF) (Eclipse, 2013), which is a tool that supports the implementation of SPEM. The EPF's target audience consists of process engineers who are responsible for defining and managing processes using reusable methods, and for configuring and executing them in the context of a project. The EPF is closely related to SPEM.

The second was the Architecture Ecosystem SIG (AESIG), a special interest group that supports the creation, analysis, integration and exchange of information between modeling languages across different domains, viewpoints and from differing authorities (OMG, 2011b).

The third was another RFP, the Case Management Process Modeling (CMPM) (OMG, 2009) that requests proposals that extend the Business Process Modeling Notation (BPMN) (OMG, 2011a) to support the modeling of case management processes.

The last was a specification: the Structured Metrics Metamodel (SMM) (OMG, 2012). The SMM specification defines a metamodel to measure information related to software, its operation and its design (OMG, 2011b).

A second section of RFP additionally requested to consider non-OMG related activities, documents and standards. This section particularly served as the actual starting point for the state-of-the-art review.

The RFP mentioned three ISO/IEC standards to take into account:

- ISO/IEC 15288 Systems and software engineering – System life cycle processes, which establishes a common framework to describe the life cycle of systems (ISO/IEC, 2008b).
- ISO/IEC 24744 Software Engineering – Metamodel for Development Methodologies,whose aim is to establish a formal framework for the definition and extension of development methodologies for information-based domains (ISO/IEC, 2007b).
- ISO/IEC 12207 Systems and software engineering – Software life cycle processes, which establishes a common framework for software life cycle processes (ISO/IEC, 2008a).

Note that these standards use very similar words to describe their scopes: common, framework, describe/define and process. After reviewing them, the search was extended to another set of standards:

- ISO/IEC 15504 Information technology – Process assessment, which defines related concepts and a generic framework for software process assessment (ISO/IEC, 2004).
- ISO/IEC 29110-5-1-2 Software engineering – Lifecycle profiles for Very Small Entities (VSEs) – Management and Engineering Guide: Generic profile group: Basic Profile, in-

tended to be used by very small organizations to establish processes with which to implement any development approach or methodology (ISO/IEC, 2011).

- ISO 9000:2005 Quality management systems – Fundamentals and vocabulary, that covers basic concepts and language of the ISO 9000 family (ISO, 2005).

- ISO/IEC TR 24774:2010 Systems and software engineering – Life cycle management – Guidelines for process description, whose purpose is to provide guidelines to describe processes by identifying descriptive elements and rules (ISO/IEC, 2010).

- IEEE 1074-2006 IEEE Standard for Developing a Software Project Life Cycle Process, which provides a process to create a software project life cycle process (IEEE, 2006).

Sources of knowledge such as Process Reference Models and bodies of knowledge like CMMI (SEI, 2010), PMBOK (PMI, 2004) and SWEBOK (ISO/IEC, 2005) were also analyzed. The analysis continued with the study of Situational Method Engineering, which involves the harmonization and standardization of methods (Harmsen and Saeki, 1996; Brinkkemper, 1996; Harmsen et al., 1994), and of a more recent work by Henderson-Sellers (Henderson-Sellers and Ralyté, 2010; Henderson-Sellers, 2011; Sousa et al., 2012).

Additionally, the agile approach was reviewed, where the SCRUM Guide (Schwaber and Sutherland, 2011) was the principal object of study, and the KANBAN (Shingo, 1989) and LEAN (Poppendieck and Poppendieck, 2003) terminologies were also considered. Lastly, the work done by SEMAT was an important input for the creation of KUALI-BEH.

4.1.2 Identification, definition and modeling

The results of the first stage were used as a basis on which to carry out a more in-depth bibliographical analysis of the theoretical aspects of software development projects. A refined search of common concept candidates to be included in the proposal also took place.

With regard to the aforementioned standards, metamodels, bodies of knowledge, process reference models and each source of knowledge explored, the following approach was adopted to generate KUALI-BEH:

1. First, the concepts used by Software Engineering project practitioners and those found in literature were collected and listed.

2. Once the concepts had been found, their similarities and differences were analyzed in order to delineate whether or not they were equivalent, and an arrangement of groups of similar terms therefore emerged.

3. A representative term was later selected from each group of concepts and its reconciled definition was created.

4. Finally the focus was to associate the selected elements with other relevant elements in order to apply them in a software endeavor context.

After defining acceptance/rejection criteria for the candidates to join the software project common concepts, the items collected were analyzed and selected. The main acceptance/rejection criteria were:

- The concept's explicit requirement by the RFP.
- The commonness of the concept within analyzed sources.
- The concept's usage in industrial and academic contexts.
- The concept's acceptance by practitioners.

4.1.3 Verification

The approach to carry out the verification process was entirely theoretical. The verification was divided in two main tasks: the first one was to submit the proposal to expert reviews (Morales-Trujillo and Oktaba, 2012b), and the second was to apply the framework within the research group in order to create practical examples of its intended usage.

In order to obtain feedback and evaluate the capability of representation, adaptation, comparison and sharing of knowledge using the software project common concepts, the proposal was subject to discussion in the SEMAT community and experts in the field; revisions were also carried out by other research groups.

A template-based representation and the software project common concepts were used to build examples of practices and methods in order to explore their expressiveness and sufficiency. The created examples represented predefined practices from agile and traditional approaches, such as SCRUM and ISO/IEC 29110.

Once the verification process had been completed, the common concepts and their relationships were finally defined (Morales-Trujillo and Oktaba, 2013), see Figure 4-2, thus finishing the engineering cycle and releasing KUALI-BEH 1.0.

4.2 Conclusions

This chapter describes the Identification phase, during which, based on the findings of the state-of-the-art review, it was possible to identify the concepts used in the context of a software project. An in-depth analysis of the wide range of terms and definitions provided in the literature was carried out.

Consequently, the common concepts that comprise the core of KUALI-BEH were identified, defined and modeled, which allowed us to verify their applicability and pertinence. The obtained results were promising, proving that KUALI-BEH was a valuable alternative that would be capable of narrowing the gap between Software Engineering theory and practice.

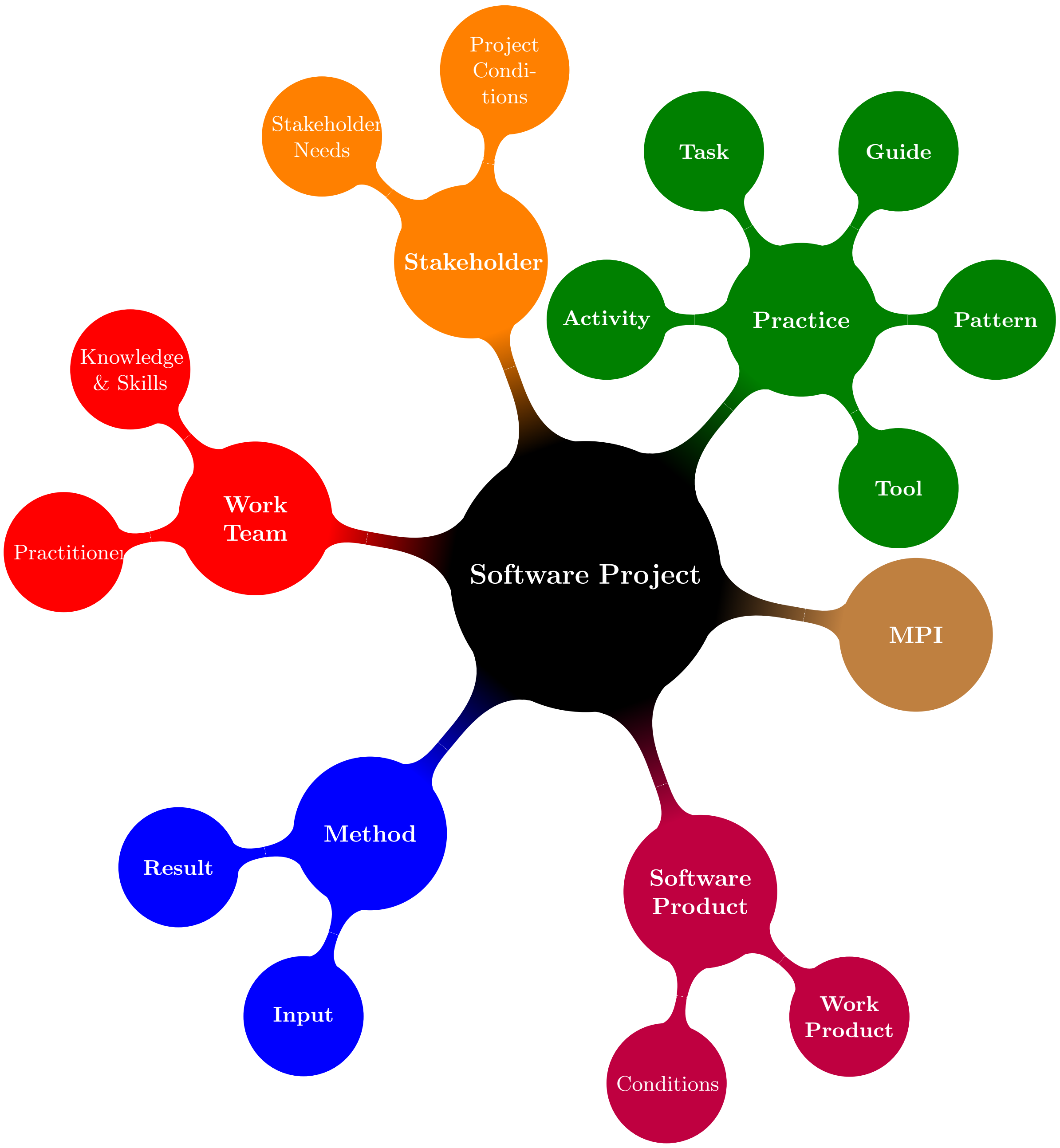


Figure 4-2: Mind map of the identified software project common concepts.

Chapter 5

Software Project Common Concepts

“La decisión correcta es la que tú tomas.”
“The right choice is the one that you make.”

Miguel Romo. Painter, photographer, sculptor, architect and writer (1960 –).

This chapter contains a detailed explanation of software project common concepts that conform the KUALI-BEH proposal. The common concepts are organized in two frameworks: conceptual and methodological. An introduction to the Authoring Extension is also presented and, at the end of the chapter a technological environment developed in order to support KUALI-BEH is shown.

5.1 Overview of the framework

KUALI-BEH has been developed as a proposal in response to the FACESEM RFP. The objective of this RFP was to obtain a foundation for the agile creation and enactment of software engineering methods by development practitioners themselves (OMG, 2011b).

KUALI-BEH consists of two views: static and operational (see Figure 5-1). The static view defines the common concepts needed for the definition of the practitioners' diverse ways of working. These ways of working are arranged into methods composed of practices. This knowledge makes up an infrastructure of methods and practices that can be applied by practitioners.

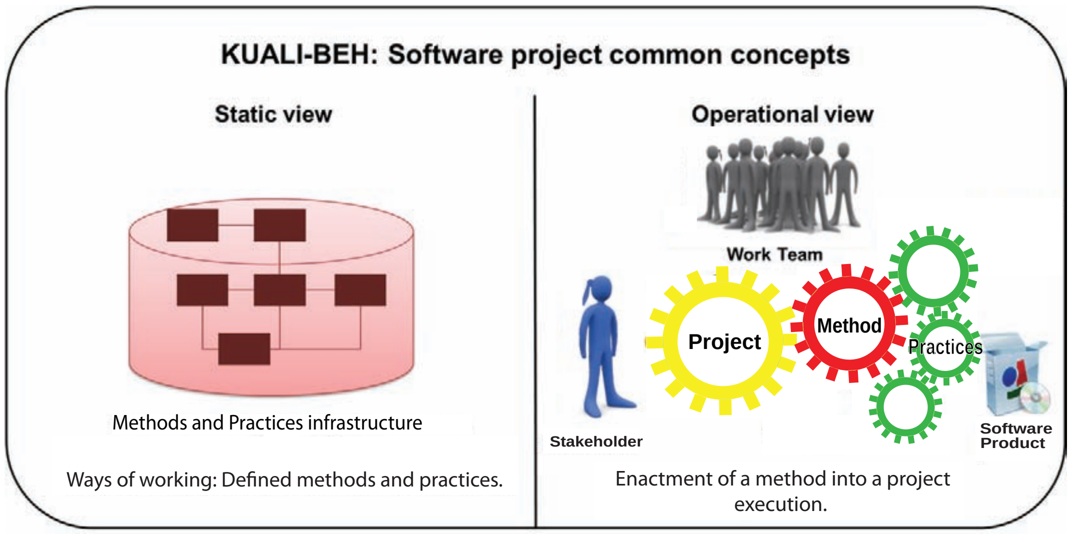

KUALI-BEH: Software project common concepts
Static view
Methods and Practices infrastructure
Ways of working: Defined methods and practices.
Operational view
Work Team
Stakeholder
Project
Method
Practices
Software Product
Enactment of a method into a project execution.

Figure 5-1: KUALI-BEH overview.

The operational view is related to the software project execution. This view provides work teams with mechanisms with which to enact a method and adapt its practices to the specific context and stakeholder needs.

The KUALI-BEH static view's main target audience is Software Engineering practitioners. Its bottom-up approach relies on the fact that practitioners have principally been developing their work without process reference models guides, standards or specifications.

Due to that fact, the model that will guide the organization's development method will be defined by practitioners, by their actual way of working to be more exact, which will be expressed using KUALI-BEH. After that, the knowledge produced by practitioners should be validated and approved before being accumulated and shared, both inside and outside the organization.

KUALI-BEH's static and operational views are described in the following subsections.

5.2 Conceptual framework (Static view)

According to KUALI-BEH, the common concepts (written in italics) involved in software projects and their relationships can be outlined as follows, see Figure 5-2.

A *software project* is the effort of a group of Software Engineering *practitioners* whose objective is to develop, maintain or integrate *software products*. A software project typically originates

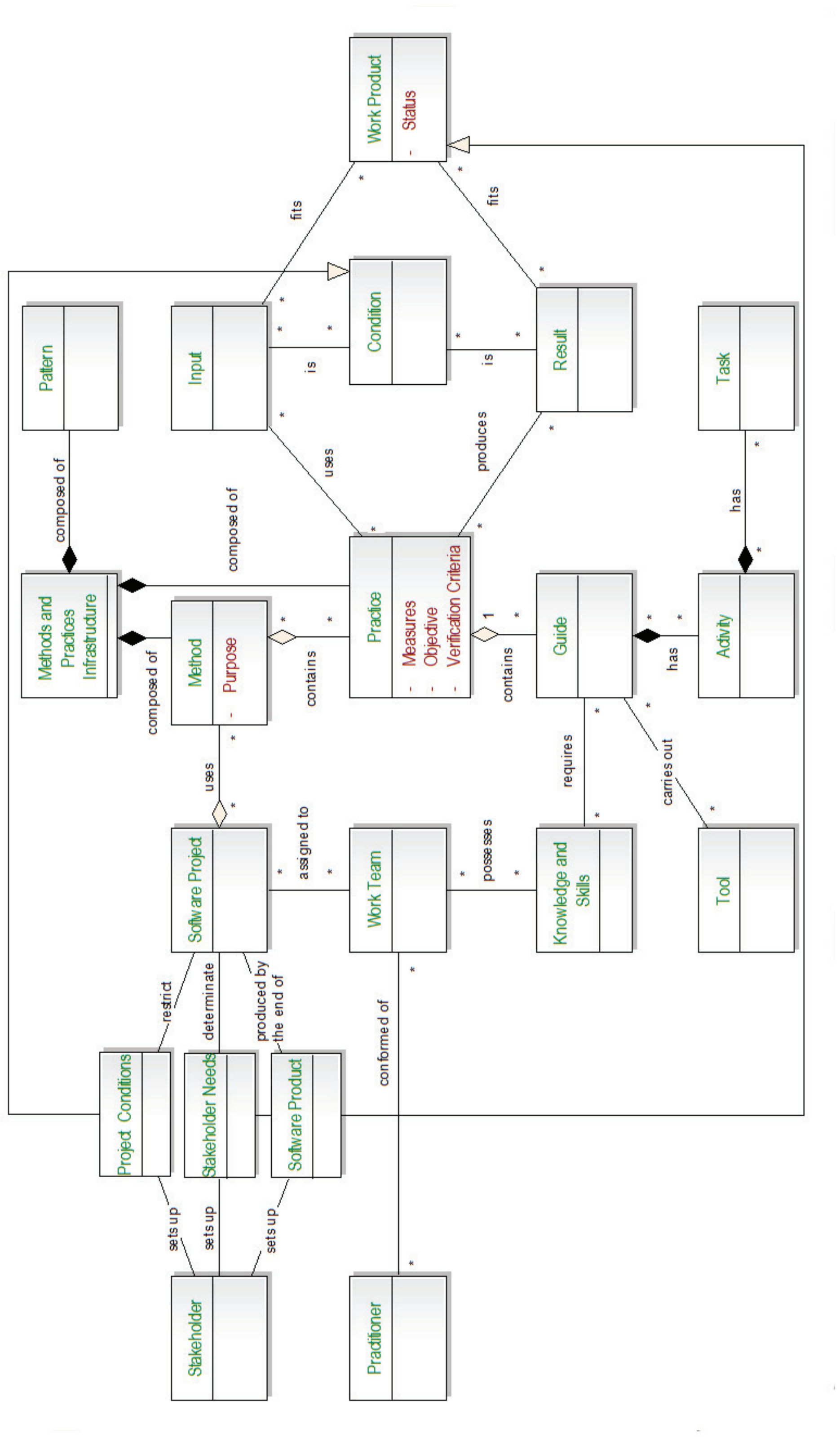


Figure 5-2: Software project common concepts and their relationships and attributes.

from the needs of an individual or an organization: a *stakeholder*. The *stakeholder needs* are expressed to a *work team*, composed of practitioners, under certain restrictions called *project conditions*. While work teams are developing software projects, they are creating their own ways of working according to their own *knowledge and skills*. These ways of working comprise methods and are composed of different *practices*. A practice is, therefore, a set of *activities* and *tasks* which has been repeatedly used in software projects and has proved to be useful.

A collection of practices can be structured by creating a *methods and practices infrastructure*. The aim of this infrastructure is to collect and concentrate the existing ways of working as units, which can be consulted and analyzed by the work team in order to select the appropriate method responding to the particular context of a software project.

Another goal of the method and practices infrastructure is to foster the addition and modification of practices and methods in a controlled manner.

The essential elements required to express the method and practice concepts are reviewed as follows.

In order to define a method, it is necessary to define its purpose, considering the characteristics of the stakeholder needs and the software product desired. In this context, a method pursues a purpose related to developing, maintaining or integrating a software product. The set of practices of which a method is composed should contribute to the achievement of this purpose.

Each practice has the objective of producing a *result* that originates from an *input*. The result should accomplish laid down *verification criteria* that are evaluated by the practitioner's judgment. If the aim is to evaluate the performance of a practice, then it is advisable to define measures that can be collected during the execution of the practice.

The inputs and results can be represented as *work products*, such as documents, diagrams or code, or as *conditions*, such as particular situations, as might be the stakeholder's availability to be interviewed.

Each practice contains a work *guide*, which is a set of activities that transforms inputs into results. Activities are additionally broken down into particular tasks. The guide can be carried out using particular *tools*. Applying the guide correctly requires the specific knowledge and skills of the practitioners involved in the work team.

The KUALI-BEH metamodel is formed of these 20 common concepts in order to share a common representation of knowledge as a set of concepts, attributes and relationships of a domain.

KUALI-BEH common concepts are intended to be used by practitioners and method engineers as the means of description, analysis and reasoning about software projects and the information related to them.

5.2.1 Common concepts definitions

This section presents definitions of KUALI-BEH common concepts and other related terms.

5.2.1.1 Software Project definition

A *software project* is a temporary effort undertaken by a work team using a method in order to develop, maintain or integrate a software product, responding to specific stakeholder needs and under particular conditions.

The stakeholder needs, project conditions and, if applies, already existing software products are considered as the input of a software project. The result is a new, modified or integrated expected software product.

The following definitions are those related to the software project concept.

Stakeholder. A *stakeholder* is an individual or organization having a right, share, claim or interest in a software product or in its possession of characteristics that meet their needs and expectations.

Software Product. A *software product* is the result of a method execution. It may contain a set of computer programs, procedures, and possibly associated documentation and data. It is a specialization of a work product.

Stakeholder Needs. The *stakeholder needs* are the representation of requirements, demands or exigencies expressed by the stakeholders to the work team.

Project Conditions. The *project conditions* are the factors related to the project that could affect its realization. Complexity, size, time and financial restrictions, effort, cost and other factors of the project environment are considered. It is a specialization of a condition.

Work Team. A *work team* is a group of practitioners that work together in a collaborative manner to obtain a specific goal. Business experts and other representatives on behalf of a stakeholder can be included in the work team.

Practitioner. A *practitioner* is a professional in Software Engineering that is actively engaged in the discipline. The practitioner should have the ability to make a judgment based on his or her experience and knowledge.

5.2.1.2 Method definition

A *method* is an articulation of a coherent, consistent and sufficient set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.

5.2.1.3 Practice definition

A *practice* is work guidance, with a specific objective, that advises how to produce a result originated from an input. The guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result. The verification criteria associated to the result are used to determine if the objective is achieved. Particular knowledge and skills are required to perform the practice guide, which can be carried out optionally using tools. To evaluate the practice performance and the objectives' achievement, selected measures can be associated to it. Measures are estimated and collected during the practice execution.

The following definitions are those related to the practice concept.

Input. An *input* is defined as expected characteristics of a work product and/or conditions needed to start the execution of a practice.

Result. A *result* is defined as expected characteristics of a work product and/or conditions required as outputs after the execution of a practice.

Guide. A *guide* is a set of recommended activities aimed to resolve a specific objective transforming an input into a result. Particular knowledge and skills are needed to perform the advised activities.

The same practice may be carried out following different guides, but they should accomplish the practice objective and preserve their input and result characteristics. The tools to support the guide carrying out could be described optionally.

Activity. An *activity* is a set of tasks that contributes to the achievement of a practice objective.

Task. A *task* is a requirement, recommendation or permissible action.

Knowledge and Skills. The *knowledge and skills* are a set of abilities, competences and attainments, acquired by the practitioner and needed to perform a practice.

Work Product. A *work product* is an artifact utilized or generated by a practice. It could have a status associated.

Condition. A *condition* is a specific situation, circumstance or state of something or someone with regard to appearance, fitness or working order that have a bearing on the software project.

Tool. A *tool* is a device used to carry out a particular function.

5.2.2 Method properties

During the authoring of methods and practices, KUALI-BEH advises practitioners with regard to certain attributes. The set of practices that comprise a method should preserve the following properties:

5.2.2.1 Coherency

A set of method practices is *coherent* if each practice objective contributes towards achieving the method's purpose.

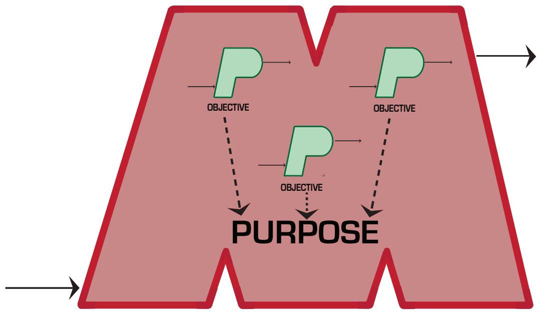


Figure 5-3: Coherent set of practices.

Figure 5-3 illustrates a coherent set of practices. Graphical symbol M represents a method and P a practice (see section 5.2.6)

5.2.2.2 Consistency

A set of method practices is *consistent* if:

- there exists at least one practice whose input is similar to the method's input and at least one practice whose result is similar to the method's result AND

For each practice of the set:

- its result is similar to the input of another practice AND
- its input is similar to the result of another practice.

Note that two or more elements are similar if, according to the practitioner's judgment, their characteristics are analogous. Figure 5-4 illustrates a consistent set of practices.

5.2.2.3 Sufficiency

A set of method practices is *sufficient* if the achievement of all practice objectives entirely fulfills the method purpose and each of the practice results are used as an input of another practice or are a result of the method.

Figure 5-5 illustrates a sufficient set of practices.

The coherency property will be determined by the practitioners' judgment, while the consistency could be evaluated automatically by a tool, since the criteria needed to determinate its

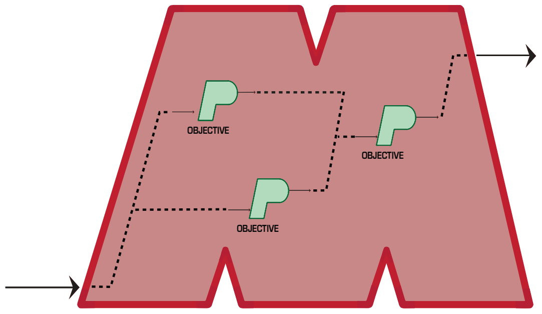


Figure 5-4: Consistent set of practices.

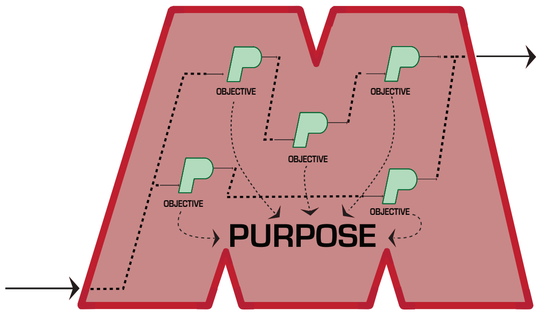


Figure 5-5: Sufficient set of practices.

achievement are the inputs and results of practices, which must be stored in a machine consumable format. Once both properties have been evaluated, the sufficiency could also be verified automatically.

5.2.3 Methods and Practices Infrastructure

Methods and practices infrastructure (MPI) is a set of methods and practices generated by the organization members as the result of experience, abstraction or apprehension. This knowledge base can be continuously expanded and modified by the practitioners.

The MPI is intended to be used by practitioners as a source of proven organizational knowledge to define a software project's way of working. It can also be useful in training new practitioners in the organization.

The knowledge contained in the MPI is manipulated through the definition of certain operations:

5.2.3.1 Family of Practices

A *family of practices* is a group of practices that shares an objective. Each of the practices belonging to the family of practices achieves the same objective. Also, the practices can be grouped by inputs or results.

5.2.3.2 Practice Pattern

A *pattern* is a set of practices that can be applied as a general reusable solution to a commonly occurring problem within a given context.

5.2.4 Methods and Practices Infrastructure operations

This subsection presents operations that are required to compose a method and modify any element of the MPI.

5.2.4.1 Composition

The *composition of practices* consists in putting together practices in order to create a method with a specific purpose, to form a family with a particular objective, or to create a pattern

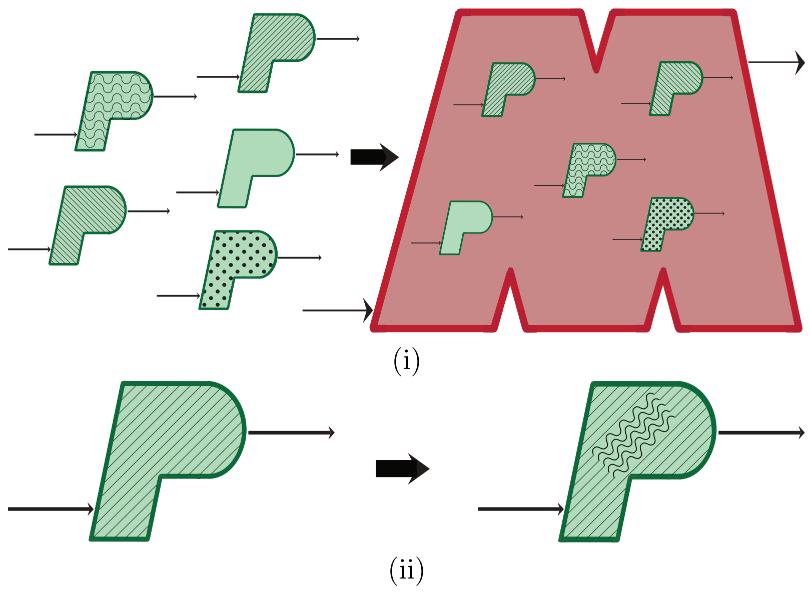


Figure 5-6: Composition and modification of practices.

as a reusable solution. The practices are taken from MPI and organized according to the practitioner's judgment. Figure 5-6 illustrates the composition of a set of practices into a method (i) and the modification of a practice (ii). The graphical symbol M represents a method, while P represents a practice. This graphical representation is for didactic use.

5.2.4.2 Modification

A *practice modification* consists of the adjustment or change made by a practitioner to a component of a practice. The modification could be applied to an input, result, objective, guide or any other element that is part of a practice.

The modification operation can also be applied to methods, practices organized as families, individual practices and practice patterns.

5.2.5 Common concepts templates

The MPI and its content are extensible and adaptable in order to support the needs of a wide variety of methods and practices. This achieves flexibility in the definition and application of these methods by practitioners in a work team.

Practitioners can use a set of templates to extend the MPI and to register the basic information of software projects. The templates are expected to be filled in when the practices and methods are authored. The templates used to capture particular software project information and definitions of method and practices are shown below.

A method or a practice can be authored by practitioners using the templates shown in Tables 5-1 and 5-2, respectively. The templates request the information and data required by the method and practice common concepts. These data have to be collected by the practitioners according to their experience and knowledge. The filled in template will be stored in the organizational methods and practices infrastructure.

Two approaches are considered in order to facilitate their usage: on the one hand, templates are a sequential format that reflects the structure of a method or a practice; on the other hand, a printable version of templates is managed as a double-sided card that pretends to make easier its usage between practitioners.

Table 5-1: Method template structure.

Front side of the card

[identifier]	**Method**	
[name]		
Purpose		
[purpose]		
Input		**Result**
[stakeholders needs, project conditions, ...]		*[software product, ...]*

Back side of the card

List of Practices
[practiceRequirements, ..., practiceDelivery]
Method Diagram
[diagram of the set of practices interrelated]

As a whole, KUALI-BEH provides practitioners with everything needed to express their diverse ways of working. It begins with templates as a first approach, and operations and

Table 5-2: Practice template structure.

Front side of the card

[identifier]	**Practice**
[name]	
Objective	
[objective]	
Input	**Result**
[list of expected characteristics]	*[list of expected characteristics]*
Verification Criteria	
[criterionA, criterionB, ...]	

Back side of the card

Guide			
Activity	*[activity]*		
Input		**Output**	
[...]		*[...]*	
Tasks*	**Tools***	**K&S**	**Measures***
[toDoThis, ..., toDoThat]	*[list of proposed tools]*	*[abilities, competences, attainments, ...]*	*[measureA, measureB, ...]*

properties are later defined through authoring, while the knowledge is collected and stored in a generated repository for everybody to consult, adapt and use.

The use of the KUALI-BEH common concepts is illustrated by using the Daily Scrum event. The content of the practice is based on (Schwaber and Sutherland, 2011). The practice name is *Daily Scrum meeting*, its objective is *to synchronize activities and create a plan for the next 24 hours*,one of its measures is *meeting duration* and the verification criteria are that *the Development Team (DT) assessed the progress toward the Sprint Goal and how progress is trending toward completing the work in the Sprint Backlog.*

The practice inputs are: a work product: the *Sprint Backlog*; and a condition to be achieved: *each DT member can explain: (i) what has been accomplished since the last meeting?, (ii) what will be done before the next meeting?, and (iii) what obstacles are in the way?*

The practice results are: a work product: the *updated Sprint Backlog*; and three conditions: *The DT's level of project knowledge was improved, the communication is improved and the impediments to development are identified and/or removed.*

Its only activity is *to inspect the work since the last Daily Scrum and forecast the work that could be done before the next one*, which can be decomposed in four tasks: *(i) each DT member is asked the three questions; (ii) each DT member explains them; (iii) updates to the Sprint Backlog; and (iv) identification and/or removing impediments.* No special tool is required.

Finally, the required knowledge and skills for the DT members are: *to be self-organized, to be cross-functional and to have specialized skills and areas of focus.*

The Daily SCRUM agile practice was expressed using the practice template and the common concepts presented in this thesis. Tables 5-3 and 5-4 show the Daily SCRUM practice as an example of how common software project concepts and a practice template can be used.

5.2.6 Graphical representation

A graphical representation of the software project common concepts is proposed in this section. This representation is meant to be used specifically by practitioners. It will be used by a work team mainly to manipulate defined methods and practices, not to define them. Figure 5-7 shows the software project representation as letter J.

The letter M is used to represent graphically a method, together with its input and result.

Table 5-3: Example of Daily SCRUM practice (front side of the card).

DailySCRUM	**Practice**
Daily Scrum meeting	
Objective	
To synchronize activities and create a plan for the next 24 hours.	
Input	**Result**
Work products: • *Sprint Backlog* *Conditions:* • *Each Development Team (DT) member can explain:* – *What has been accomplished since the last meeting?* – *What will be done before the next meeting?* – *What obstacles are in the way?*	*Work products:* • *Sprint Backlog [updated]* *Conditions:* • *The DT's level of project knowledge was improved.* • *The communication is improved.* • *Impediments to development are identified and/or removed.*
Verification Criteria	
DT assessed the progress toward the Sprint Goal and how progress is trending toward completing the work in the Sprint Backlog.	

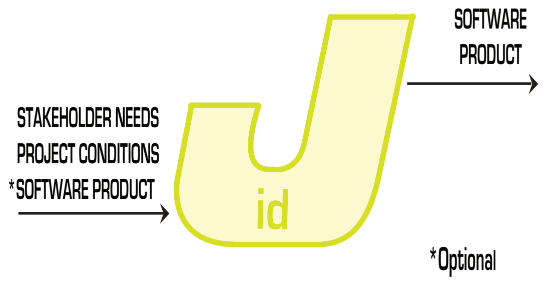


Figure 5-7: Software Project symbol.

Table 5-4: Example of Daily SCRUM practice (back side of the card).

Guide			
Activity	*To inspect the work since the last Daily Scrum and forecast the work that could be done before the next one.*		
Input		**Output**	
Work products: • *Sprint Backlog* *Conditions:* • *Each DT member can explain:* – *What has been accomplished since the last meeting?* – *What will be done before the next meeting?* – *What obstacles are in the way?*		*Work products:* • *Sprint Backlog [updated]* *Conditions:* • *The DT's level of project knowledge was improved.* • *The communication is improved.* • *Impediments to development are identified and/or removed.*	
Tasks*	**Tools***	**K&S**	**Measures***
1. Each DT member is asked the three questions. *2. Each DT member explains them.* *3. To update the Sprint Backlog.* *4. To identify and/or remove impediments.*	*None*	*DT members are self-organizing. DT members are cross-functional. DT members may have specialized skills and areas of focus.*	*Meeting duration (15-minute time-boxed)*

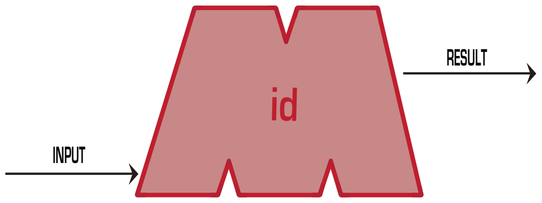


Figure 5-8: Method symbol.

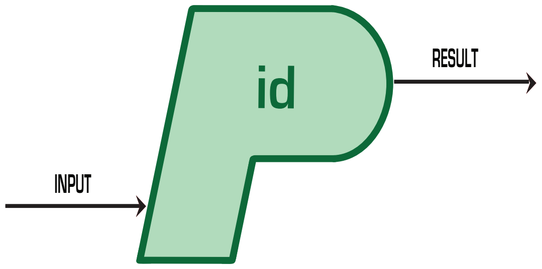


Figure 5-9: Practice symbol.

See Figure 5-8.

The letter P is used to represent a practice, its input and result. See Figure 5-9.

The aim of these graphical symbols is to facilitate the representation and manipulation of the previously defined and instantiated common concepts. These symbols are proposed to be used during work team discussions and facilitate their comprehension. These graphical symbols can be adjusted and improved by the practitioners.

5.3 Methodological framework (Operational view)

The KUALI-BEH operational view describes the software project execution. It expresses the enactment of a method by a work team during a software project execution. The method enactment implies the changes made to the method and its practice instance states. New terms related to the states of method and practice instances are written in bold type.

A new software project starts when the work team gets to know the stakeholder needs and is informed about the project conditions. In the case of a maintenance or software integration project, the already existing software product(s) should also be available.

At the beginning of the project, the work team selects a method from the organizational practice and method infrastructure according to the general characteristics of the project. In order to perform the selected method successfully, the work team has to fulfill the knowledge and skills requirements specified in the practices guide. If this is not the case, appropriate training is recommended.

The **selected** method usually has to be **adapted** in accordance with stakeholder needs and project conditions.

The purpose of adapting a method is to identify work units to be completed during the software project execution. The work team attains this goal by analyzing the practices of the selected method and, if necessary, applying the practice substitution, concatenation, splitting or combination.

The consistency, coherency and sufficiency properties of the original set of practices must be preserved. The resulting set of practices is instantiated as work units planned to be executed during the project. Each practice instance work unit requires following the practice guide, and the method consequently changes to the adapted state.

When a required input is available, the work team assigns it to the appropriate practice instance. The practice instance, with an assigned input, changes to a **can-start** state. When at least one practice is in a can-start state, the method reaches a **ready-to-begin** state.

The work team starts the practice instance execution by estimating the measures associated with the practice, agreeing on the work distribution, on who is responsible for it and beginning to work. This means that the practice instance changes to an **in-execution** state and the method enactment changes to an **in-progress** state.

The work team may decide to interrupt the practice instance execution, and the practice instance therefore changes to a **stand-by** state. At some point, the work team may decide to restart and the practice instance again changes to an in-execution state.

The practice instance execution produces a result, which should be verified by the work team with the use of result verification criteria. At this moment the practice instance changes to an **in-verification** state.

If the work team verifies that the result is correct, the practice instance is **finished**. If this is not the case, then the work team should correct the result and the practice instance again goes back to the in-execution state. In some cases, the work team may decide to cancel the practice instance. If the practice has finished or is **cancelled**, the measures associated with the practice should be collected.

The method enactment can change to a **progress-snapshot** state whenever the work team produces a verified result, cancels a practice instance, or changes to the stakeholder needs or the project conditions occur. In this state, the work team has to analyze the situation and decide to take one of the following actions:

- Assign available input to the existing practice instance and continue the enactment of the method.
- Apply an adaptation of method practices, taking into account the practice instance cancellation, the stakeholder needs change requests, the changes to the project conditions, or anything else that may affect the project.

Lastly, the method enactment can be cancelled if the work team decides to do so, or finished if the expected software product is produced and all the practice instances are finished or have been cancelled.

5.3.1 Practice Instance Lifecycle

During the enactment of a method by a work team (WT), each practice is initially instantiated and it state is then constantly changing until it has finished or has been cancelled. The valid practice instance states during their lifecycle are shown in Table 5-5

The transitions between practice instance states are described in Table 5-6 .

Figure 5-10 shows the state diagram that represents the practice instance lifecycle.

5.3.2 Method Enactment

A method enactment occurs in the context of a software project execution. Before starting the method enactment, those assigned to the software project work team get to know the stakeholder needs and are informed of the software project conditions. In the case of a maintenance or software integration project, the already existing software product(s) should also be available.

The valid states of a method enactment, carried out by a work team during the project execution, are shown in Table 5-7

Table 5-5: Practice instance lifecycle states

State	Definition
Instantiated	The practice instance is created as a result of the method adaptation. Optionally, measures can be estimated.
Can-Start	The required input has been assigned to the practice instance and it can start at any time.
In-Execution	The practice instance has been chosen, its measures have been estimated and WT has agreed who is responsible for it. The guide associated with the practice instance is being carried out.
Stand-By	The practice instance execution has been interrupted, its associated items remain paused.
In-Verification	The practice instance result is being verified against the verification criteria.
Cancelled	The practice instance is over, WT has quit its associated items.
Finished	The practice instance is over and its result has been produced correctly.

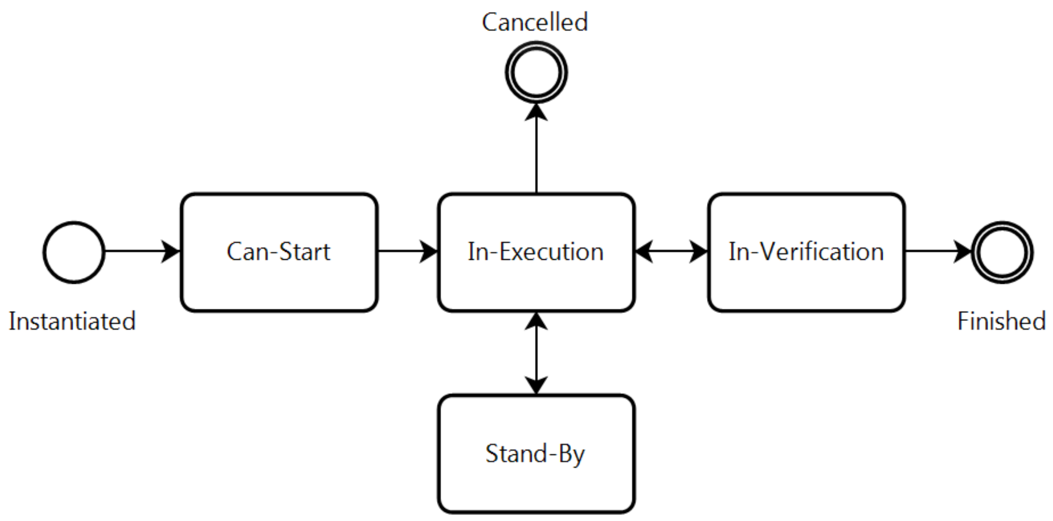


Figure 5-10: Practice Instance state machine.

Table 5-6: Practice instance lifecycle transitions

From state	Event that causes the transition	To state
Instantiated	WT assigns work products and/or conditions, which meet the required practice input characteristics. Optionally WT can estimate the practice measures.	**Can-Start**
Can-Start	WT chooses a practice instance, estimates the practice measures, agrees who is responsible for it and starts its execution.	**In-Execution**
In-Execution	WT decides to interrupt the practice instance execution.	**Stand-By**
In-Execution	WT decides to verify the result produced by the practice instance execution.	**In-Verification**
In-Execution	WT decides to cancel the practice instance execution.	**Cancelled**
Stand-By	WT decides to restart the practice instance execution.	**In-Execution**
In-Verification	WT realizes that the work products or conditions do not meet the result verification criteria and corrections to them are required. WT verifies them as incorrect.	**In-Execution**
In-Verification	WT confirms that the generated work products and/or reached conditions meet the result verification criteria. WT verifies them as correct.	**Finished**

Table 5-7: Method enactment states.

State	Definition
Selected	The method has been selected from the organizational methods and practices infrastructure according to general characteristics of a project (new development, maintenance or integration). The WT members have to fulfill the required knowledge and skills specified in the method practices guides. If it is not the case, appropriate training is needed.
Adapted	The method has been adapted and the resulting set of practices is instantiated as work units planned to be executed during the project.
Ready-to-Begin	The method has at least one practice instance in Can-Start state. The method is ready to begin at any time.
In-Progress	The method has at least one practice In-Execution, Stand-By or In-Verification states. The method remains in this state while it is being applied.
Progress-Snapshot	The method context is being analyzed and under discussion in order to take actions.
Cancelled	The method is over and its result has not been produced.
Finished	The method is over and its result can be delivered.

The transitions between method enactment states are described in Table 5-8

Figure 5-11 shows the diagram of possible states of the method enactment.

The method enactment can reach more than one state at the same time, owing to the behavior of the practice instances lifecycle. For example, at one particular moment, one group of practice instances may be in an execution state, other practices may be in a can-start state and others may have finished, signifying that the method enactment is at different states at the same time. The method enactment behavior can therefore be represented as a variation of a non-deterministic finite-state machine.

5.3.3 Operational boards

During the execution of a project, the work team needs to visualize the project's on-going performance. The method enactment and the practice instance boards are used to display information that is relevant to the project.

The KANBAN approach was taken into account in order to design the operational boards. The column-based board is preserved, however its complexity increases since more information

Table 5-8: Method enactment transitions.

From state	**Event that causes the transition**	**To State**
Selected	WT adapts the selected method, taking into account stakeholder needs and project conditions. WT analyzes the selected method practices and, if necessary, applies the practice substitution, concatenation, splitting or combination. For each practice of the adapted method the practice instances are created and, optionally, the practices measures estimated.	**Adapted**
Adapted	WT assigns an input to at least one practice instance.	**Ready-to-Begin**
Ready-to-Begin	WT chooses a practice instance in Can-Start state, estimates the measures associated to it, agrees on work distribution, on who is responsible for it and begins its execution.	**In-Progress**
In-Progress	WT verifies a result or decides to pause the execution of a practice instance.	**In-Progress**
In-Progress	WT produces a verified result and collects measures; or WT cancels a practice instance and collects measures; or changes occur in stakeholder needs or project conditions.	**Progress-Snapshot**
Progress-Snapshot	WT assigns available inputs to the existing practice instances, that changes their states to the Can-Start state.	**Ready-to-Begin**
Progress-Snapshot	WT applies method practices adaptation, taking into account the practice instance cancelation, the changes in stakeholder needs and/or project conditions, or anything else that can affect the project. As a result, new practices are Instantiated.	**Adapted**
Progress-Snapshot	WT decides to stop the method permanently.	**Cancelled**
Progress-Snapshot	WT produces the expected method result and all of the practice instances are in the Finished or Cancelled states.	**Finished**

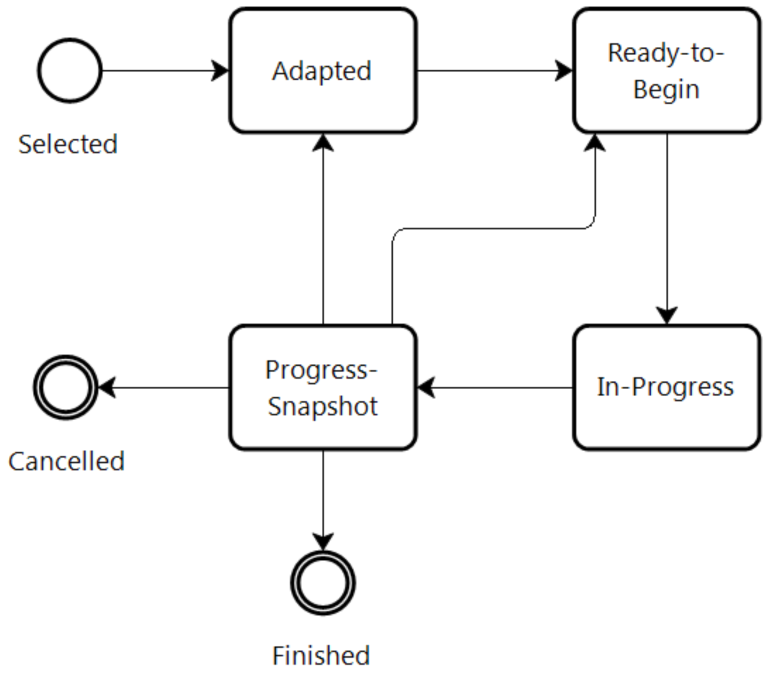


Figure 5-11: Method Enactment state machine.

is presented. It is important to say that the boards are designed in order to be managed through a software tool, if it is not the case, a simpler version can be used.

In the next subsections each board is presented in detail.

5.3.3.1 Method Enactment board

The method enactment board communicates method states mainly. The practice instances, organized by state, are associated to method enactments states. Optionally, responsible and reporting date can be added in each practice instance row. A numerical value can be assigned to each practice instance state in order to calculate the global progress of the method enactment.

A section for work products and/or conditions used by the practice instances paired with their respective status is also optional. Table 5-9 shows a proposed board for the method enactment.

Table 5-9: Method Enactment board.

project id		**Method Enactment Board**					*[today's date]*	*[end's date]*
Input				**Result**				
[list of inputs]				*[list of results]*			**Days left**	
Enactment States								
ID	**Adapted**	**Ready-to-Begin**	**In-Progress**			**Progress-Snapshot**		**Global**
	Instantiated	**Can-Start**	**In-Execution**	**In-Verification**	**Stand-By**	**Cancelled**	**Finished**	**Progress**
1			*[practice name, responsible and reporting date]*					*60*
2	*[practice name, responsible and reporting date]*							*20*
3							*[practice name, responsible and reporting date]*	*100*
							Total	180/300
	Work Product/Conditions							
	[list of work products and/or conditions paired with their respective status]							

5.3.3.2 Practice Instance board

The practice instance board reflects the practice state at one particular moment. Each practice instance board also represents the responsible for it work team and associated to it measures. A numerical value together with the estimated and actual start and end dates can be associated to each practice instance state in order to calculate its progress. Table 5-10 shows a proposed board for practice instances.

5.3.4 Methods and practices adaptation

Method adaptation is the action carried out by the work team, taking into account the stakeholder needs and their changes, the project conditions and other factors that may affect a software project.

The purpose of adapting a method is to identify and/or modify the work units that need to be completed during the software project execution. This goal is attained with the following actions:

- The practitioners must analyze the practices from the method selected or the remaining practice instances and, if necessary, apply the practice substitution, concatenation, splitting or combination.
- The resulting set of practices is instantiated as work units planned to be executed during the software project. Each of the practice instances involves following the practice guide.

KUALI-BEH defines its adaptation operations as follows:

- **Substitution:** this consists of replacing a practice from a method with another equivalent practice.
- **Split:** this consists of the partition of the original practice into two different practices, thus preserving the accomplishment of the original objective and similar inputs and results.
- **Combination:** this consists of uniting two different practices into one. The resulting practice preserves the accomplishment of the original objectives.

Table 5-10: Practice Instance board.

[project id-method id-practice id]			**Practice Instance Board**			
Input			**Result**			
[list of inputs]			*[list of results]*			
Work Team Practitioners			**Measures**			
			Estimated		**Actual**	
[list of responsible practitioners]			*[list of measures estimations]*		*[list of actual measures]*	
Activity Progress						
Activities	**Progress**		**Responsible**	**Comments**		
[activity ID]	*[numerical value]*		*[practitioner]*	*[comments and important notes]*		
Practice Instance States						
Instantiated	**Can-Start**	**In-Execution**	**In-Verification**	**Stand-By**	**Cancelled**	**Finished**

- **Concatenation:** if one practice has a result that is similar to the input of another practice, both can be integrated into one practice by applying the concatenation operation. The resulting objective will be the union of both original objectives.

Figure 5-12 illustrates the operations of substitution (i), splitting (ii), combination (iii) and concatenation (iv). The graphical symbol M represents a method, while P represents a practice. This graphical representation is for didactic use.

5.4 KUALI-BEH Authoring and Enactment Extension

KUALI-BEH proposal was used as a basis to generate an Extension (OMG, 2013) for the ESSENCE proposal. Four sub-Alphas were proposed and anchored to the Endeavor area of concern.

On the one hand the Authoring Extension, anchored to Way-of-Working Alpha, is formed from sub-Alphas:

- Practice Authoring
- Method Authoring

On the other hand, the Enactment extension, anchored to Work Alpha, is made up from sub-Alphas:

- Practice Instance
- Method Enactment

The four sub-Alphas are presented briefly in the following subsections, while its full description is shown in Appendix A.

5.4.1 Practice Authoring

Practice Authoring provides a framework for the definition of the practitioners' different ways of working. This knowledge forms an infrastructure of methods and practices that is defined and applied by the practitioners in the organization.

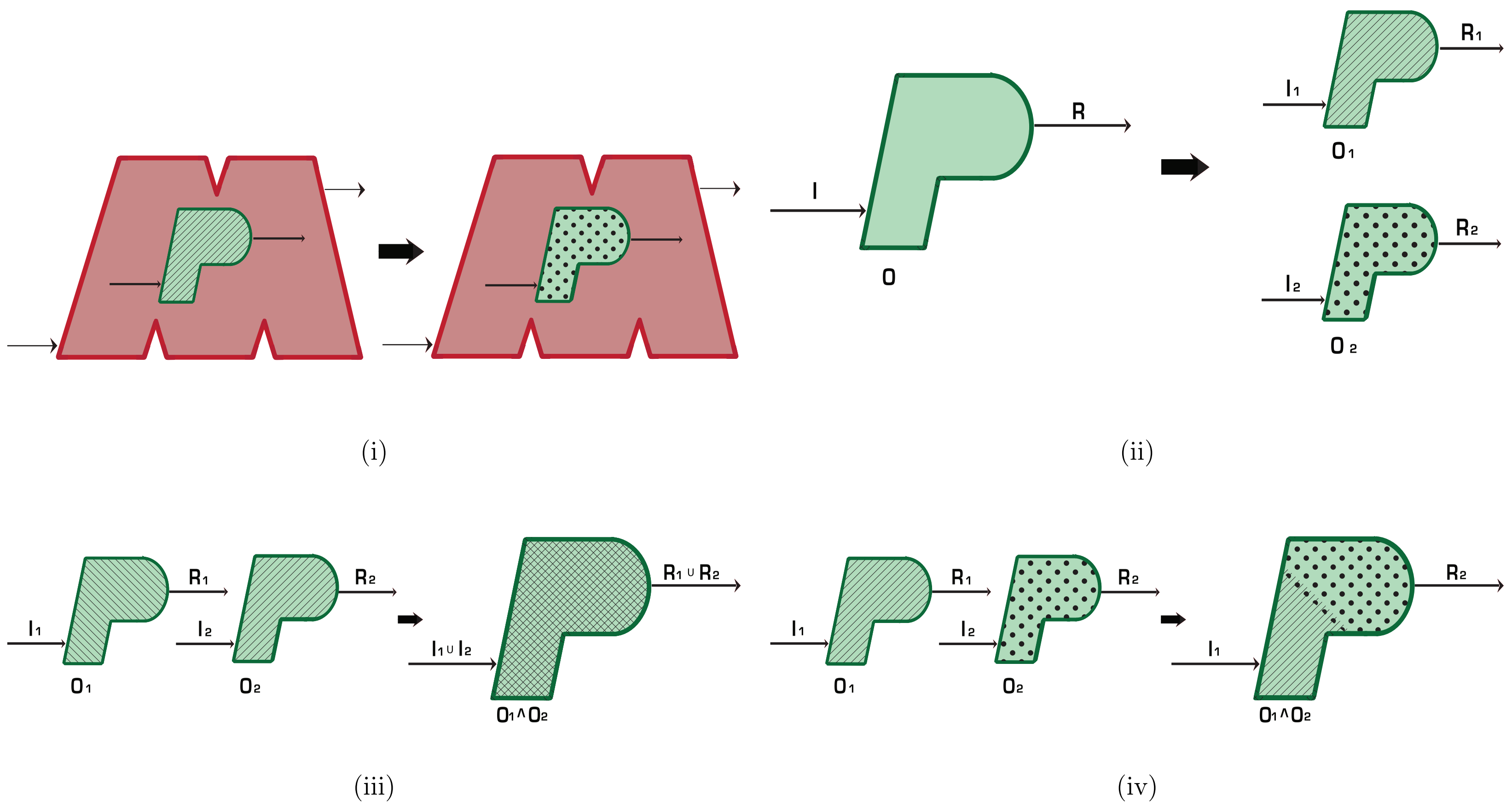


Figure 5-12: Operations of adaptation.

The super-ordinate Alpha of the Endeavor area of concern is Way-of-Working, and the associations that attach it to Practice Authoring are the following:

- **Defines — Way-of-Working:** the progress of the Method Authoring defines the maturity of the practitioners' way of working.
- **Composes — Practice Authoring:** a method is composed of the authored practices.

The Practice Authoring states are defined as follows:

- **Identified:** the way of working to be authored as a practice is identified by the practitioners.
- **Expressed:** the way of working is expressed as a practice using the practice template.
- **Agreed:** the practice is agreed on by the practitioners.
- **In Use:** the practice is used in software projects by the practitioners as their way of working.
- **In Optimization:** the practice is adapted and/or improved by the practitioners based on their experience, knowledge and external influence.
- **Consolidated:** the practice matures and is adopted by the practitioners as a routine way of working.

The Way-of-Working Alpha is expressed by the Practice Authoring sub-Alpha as shown in Figure 5-13.

5.4.2 Method Authoring

Method Authoring permits the definition of the practitioners' different ways of working and is composed of the authored practices. The Method Authoring is analogous to the Practice Authoring with the exception of two new states, which are defined as follows, see FIgure A-2:

- **Integrated:** the method is integrated as a composition of agreed practices.

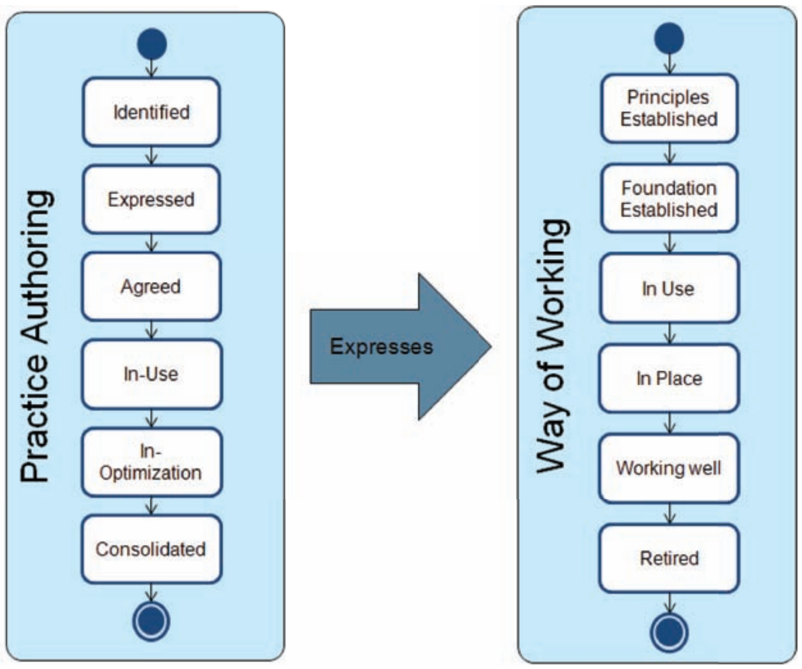


Figure 5-13: Practice Authoring defines the Way-of-Working (OMG, 2013).

- **Well Formed:** the method is agreed on by the practitioners and accomplishes the properties of coherency, consistency and sufficiency.

These new states of Method Authoring substitute the Expressed and Agreed states of Practice Authoring, respectively.

5.4.3 Practice Instance

Practice Instance occurs during the enactment of a method by practitioners, each practice is initially instantiated as work to be done. Later it changes its state to can start, in execution, stand by or in verification until it is finished or cancelled. Its super-ordinate Alpha of the Endeavor area of concern is Work.

To assess the state and progress of Practice Instance a checklist is provided in Table 5-11.

5.4.4 Method Enactment

Method Enactment occurs in the context of a software project execution. Before starting the method enactment, the practitioners assigned to the software project get to know the stakeholder

Table 5-11: Checklist for Practice Instance Alpha.

State	Checklist
Instantiated	• The practitioners have identified the work to be done. • The needed work unit has been created as the practice instance. • The practice instance measures have been optionally estimated by practitioners.
Can-Start	• The required practice instance input has been created and assigned. • The practice instance measures have been estimated.
In-Execution	• The practitioners have chosen a practice instance that can start. • The practitioners responsible for the practice instance have been agreed upon • The practitioners are working on the practice instance following the guide.
Stand-By	• The execution of the practice instance has been interrupted. • The practitioners have paused any work related to the practice instance.
In-Verification	• The practitioners have produced a result after executing the practice instance. • The practitioners are verifying the result using the related completion criteria.
Cancelled	• The practitioners have stopped permanently the practice instance work. • The associated items of the practice instances have been quit.
Finished	• The practitioners have finalized the practice instance work. • The practitioners have produced a result, which was verified as correct.

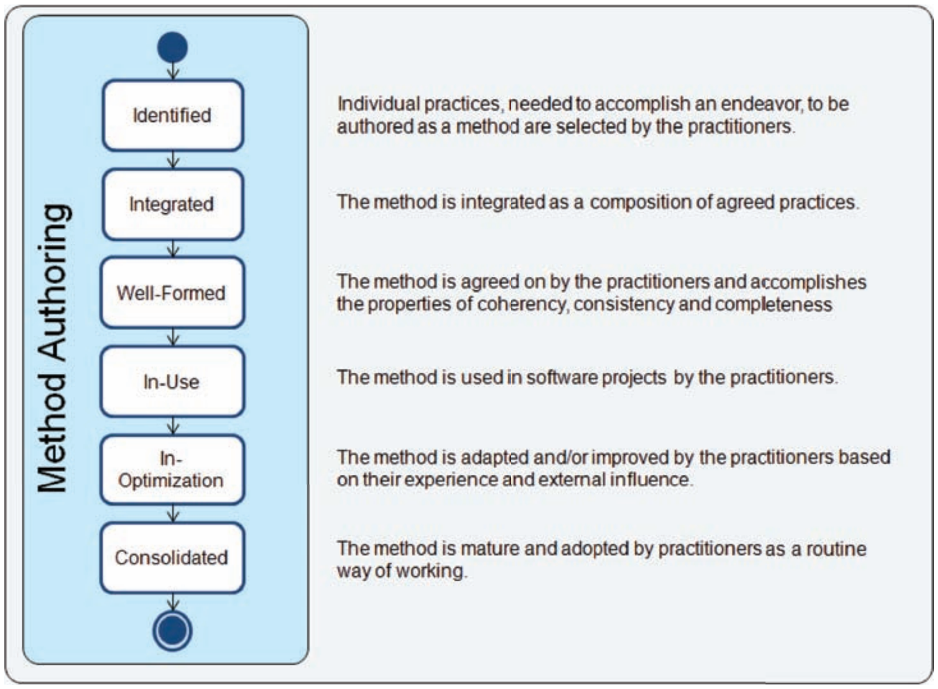

Method Authoring
Identified
Integrated
Well-Formed
In-Use
In-Optimization
Consolidated
Individual practices, needed to accomplish an endeavor, to be authored as a method are selected by the practitioners.
The method is integrated as a composition of agreed practices.
The method is agreed on by the practitioners and accomplishes the properties of coherency, consistency and completeness
The method is used in software projects by the practitioners.
The method is adapted and/or improved by the practitioners based on their experience and external influence.
The method is mature and adopted by practitioners as a routine way of working.

Figure 5-14: Method Authoring defines the Way-of-Working (OMG, 2013).

needs and are informed about the software project conditions.

In case of a maintenance or software integration project, the already existent software product(s) should also be available. Its super-ordinate Alpha of the Endeavor area of concern is Work.

To assess the state and progress of Method Enactment a checklist is provided in Table 5-12.

5.5 Technological environment: KBTool

According to (Arni-Bloch, 2005), the different kinds of technological environments needed to support the engineering of methods can be classified in three tools:

- The first tool is a methods repository.
- The second tool is based on the method meta-model and it is responsible for the product and process definition.
- The third tool is the method instantiation tool. This tool or this family of tools have to

Table 5-12: Checklist for Method Enactment Alpha.

State	Checklist
Selected	• The practitioners have selected a well-formed method from the methods and practices infrastructure. • The practitioners have fulfilled the required competencies specified in the method practices guides.
Adapted	• The practitioners have analyzed the stakeholder needs and conditions of the software project. • The practitioners have adapted the selected method. • Each of the practices of the method has been instantiated as work units planned to be executed during the software project.
Ready-to-Begin	• The method has at least one practice instance in Can Start state. • The method and the practitioners are ready to begin the work.
In-Progress	• The practitioners are applying the method. • The practitioners have paused any work related to the practice instance.
Progress-Snapshot	• The practitioners are analyzing the method execution context. • The practitioners are discussing and taking decisions about the work continuation as it was planned or if the method requires an adaptation.
Cancelled	• The practitioners have stopped permanently the method execution. • The associated items of the method have been quit. • The result has not been produced.
Finished	• The practitioners have finalized their work. • The practitioners have produced a result that can be delivered.

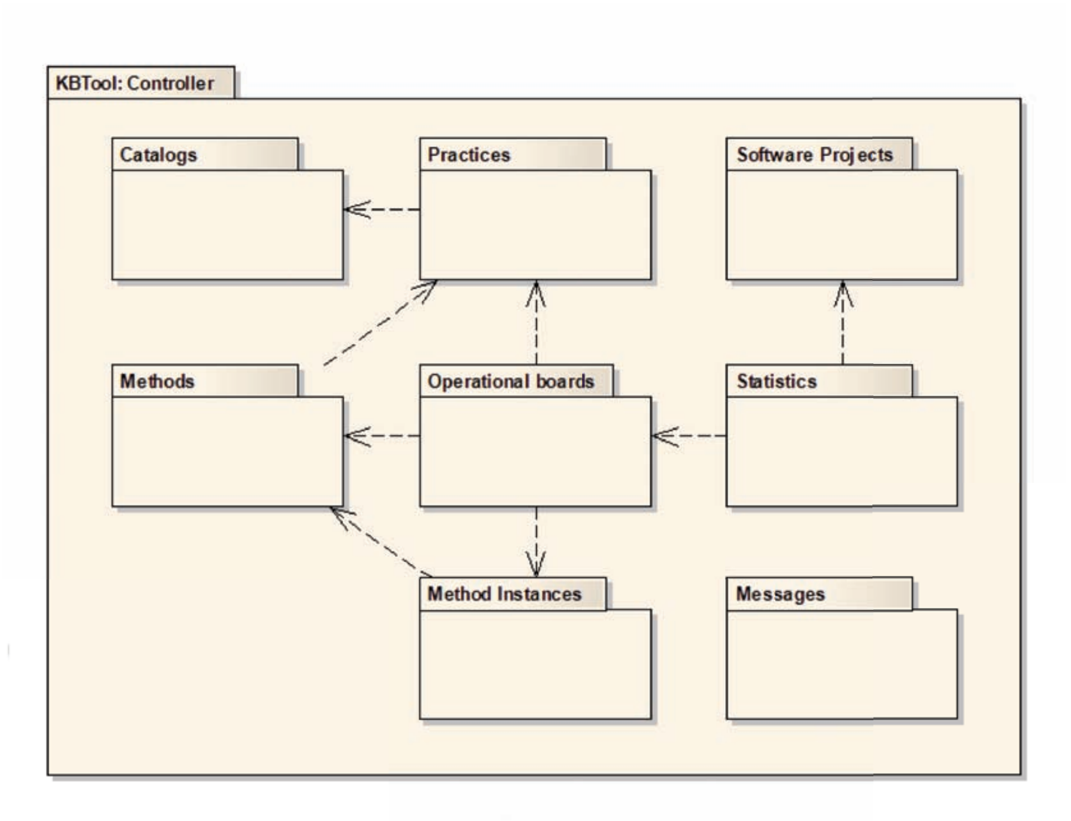


Figure 5-15: KBTool software architecture.

be able to perform the enactment of the constructed method.

Based on these requirements, the design and construction of a set of software tools with which to support KUALI-BEH, KBTool (Urrutia et al., 2013), has also begun. The architecture, technological considerations and the modules of KBTool are presented in the following subsections.

5.5.1 Architecture and technological considerations

KBTool is a modular tool which applies the Model-View-Controller pattern to its architecture, see Figure 5-15

The technological considerations followed during the development of the tool were:

- **To the Model:** Hibernate framework and the use of DAO pattern on a PostgreSQL database was the combination responsible for the date persistence of the tool.

- **To the View:** Java Server Faces (JSF) and PrimeFaces as an extension were used to create presentation elements of the application.
- **To the Controller:** Java as programming language and Session Beans as framework were the choices to implement business logic and behavior of the system. Also eXtensible Markup Language (XML) was used to provide an alternative format of data in order to foster the validation of the method properties.

The full description of KBTool architecture is presented in (Tapia and Ibargüengoitia, 2014).

5.5.2 Modules of KBTool

KBTool consists of four modules: KBSview, which implements the KUALI-BEH's Static view; KBOview, in charge of the Operational view; KBMetMan, which deals with the validation process of method properties; and KBProjMan, which manages the information and statistics of software projects. Each of these modules are detailed as follows.

5.5.2.1 KBSview

This module allows practitioners to express their ways of working through KUALI-BEH common concepts. It has a catalog-centered approach to manage the instances of common concepts individually. To achieve this, KBSview implements the following catalogs:

- Knowledge and Skills
- Tools
- Tasks
- Verification Criteria
- Measures
- Work Products/Conditions
- Characteristics
- Types

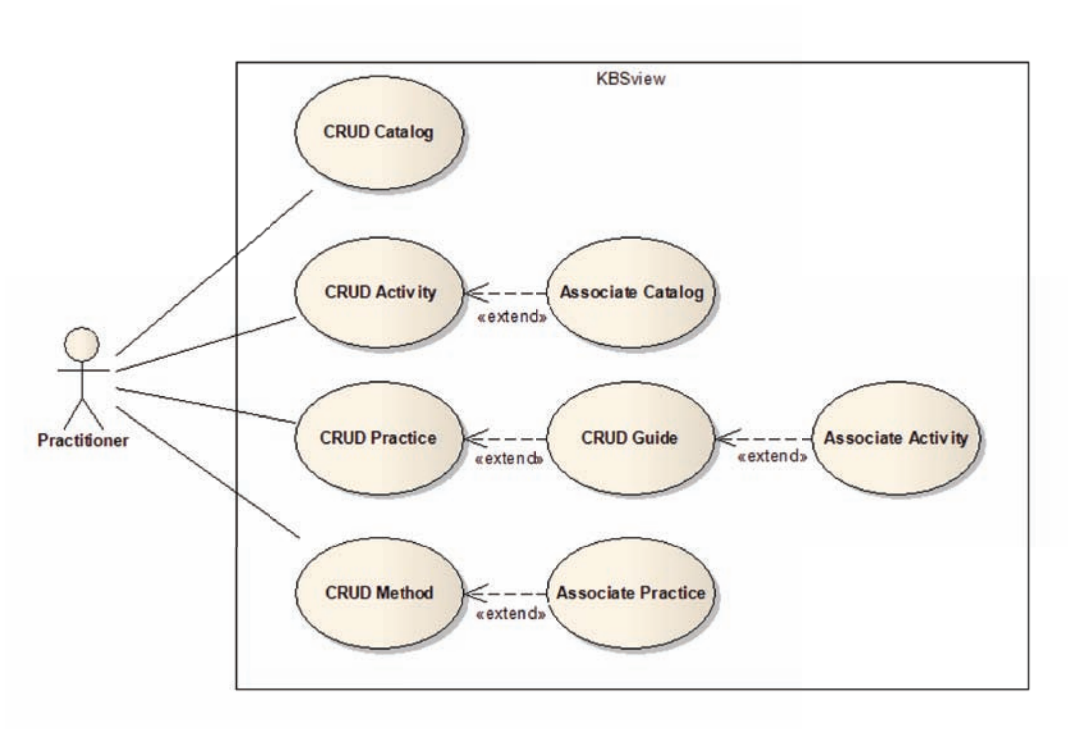


Figure 5-16: KBSview functionalities.

Using these catalogs a practitioner can define a practice by adding activities and associating them with the desired elements, and finally to make a method using previously created practices. Figure 5-16 shows the use cases implemented by KBSview module.

Figure 5-17 shows a screenshot of the tool when a practitioner is creating a practice.

Figure 5-18 shows a screenshot of the tool when a practitioner is adding a new work product or condition.

The full description of KBSview is presented in (Barrera and Oktaba, 2014).

5.5.2.2 KBMetMan

This module provides a mechanism to validate method properties defined in KUALI-BEH. KBMetMan main functionality is the semi-automatic validation of properties that allows practitioners to evaluate the Coherence of a method by checking each of the practices' objectives and comparing them against the method purpose. Due to the fact that this process involves interpretation of a natural language, the decision if the property is achieved or not relies completely

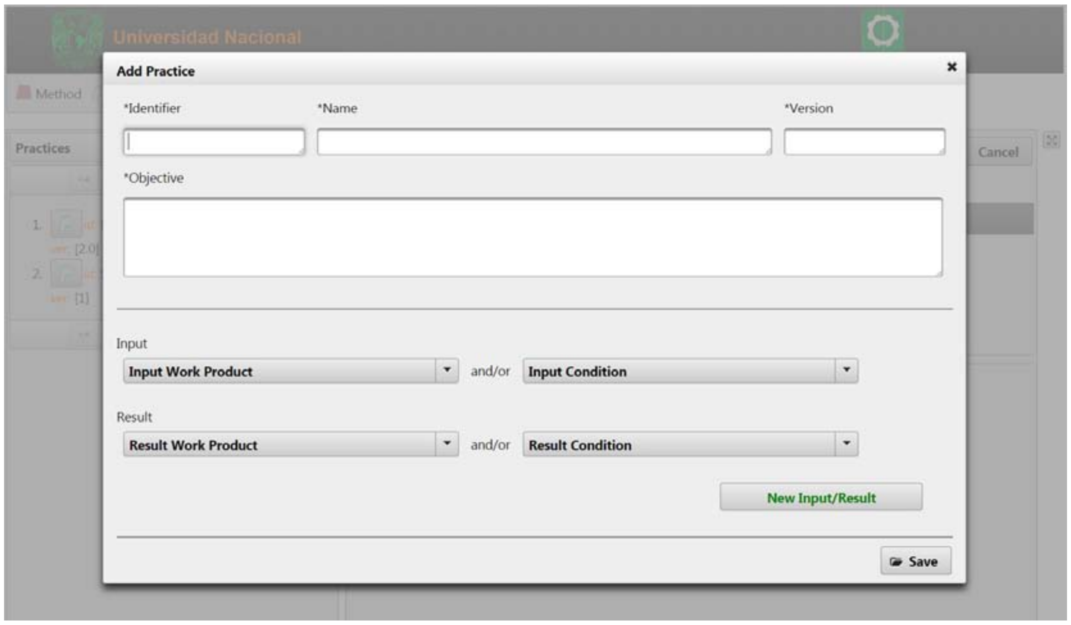


Figure 5-17: KBSview allows the creation of practices.

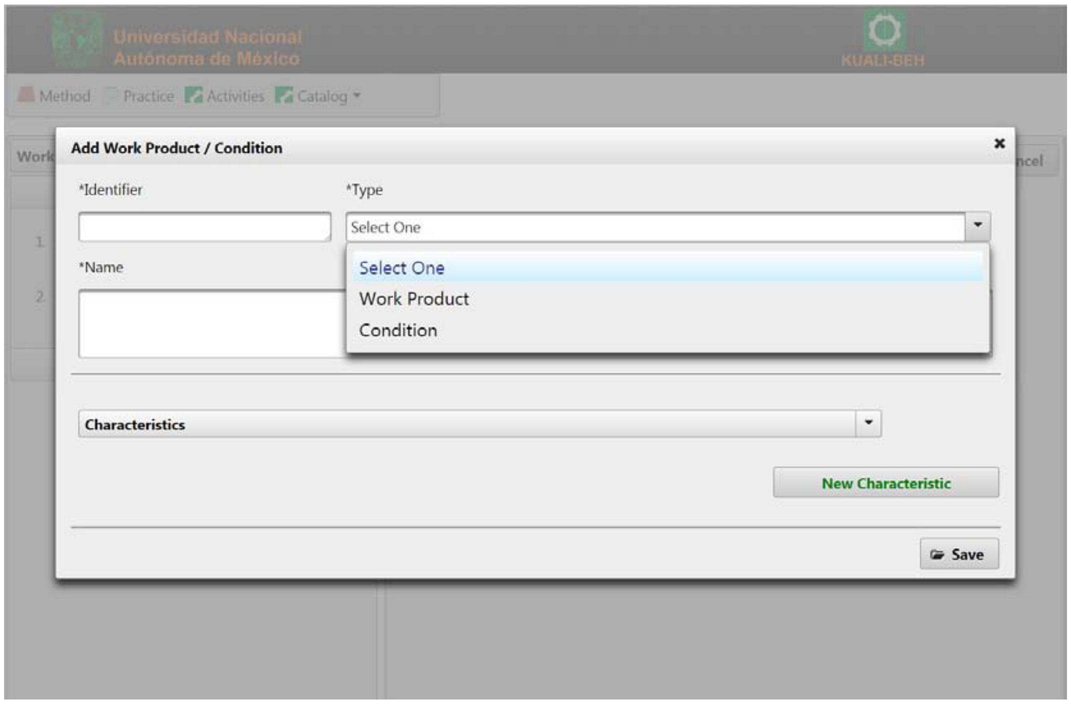


Figure 5-18: KBSview permits to add a new work product or condition.

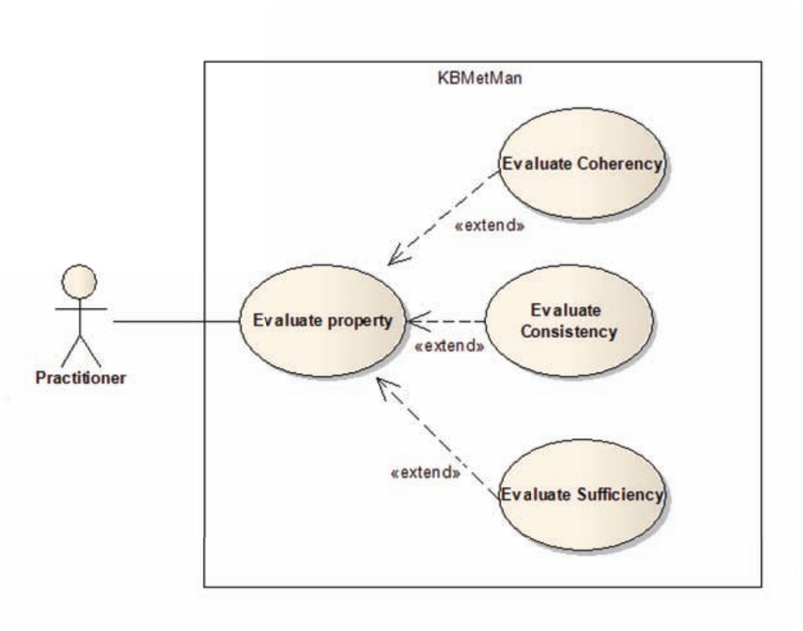


Figure 5-19: KBMetMan functionalities.

on practitioner.

The Consistency property validation is done by the tool, making a comparison between inputs and results of each practice that comprise a particular method.

Lastly, the Sufficiency property is evaluated in a semi-automatic way: first the tool verifies if the Coherency and Consistency properties are accomplished, and if it is not the case, the Sufficiency property will not be achieved. But, if the first two properties are achieved, the system allows practitioner to decide if the third property is satisfied.

Figure 5-19 shows the use cases of KBMetMan module.

The properties evaluation process is based on logical rules executed using an XML schema.

The full description of KBMetMan is presented in (Medina and Oktaba, 2014).

5.5.2.3 KBOview

The goals of this module are:

- To provide practitioners with a visual alternative to compose and adapt methods and practices.

- To allow practitioners to manage and control their projects based on the use of operational boards.

5.5.2.4 KBProjMan

Finally, the module of KBProjMan will manage the data collected from each software project developed in the organization. The purpose here is to give practitioners an interface that supports the decisions taking process based on statistics obtained from the collected data.

It is important to mention that the first versions of KBSview and KBMetMan have been released in 2014. However the KBOview and KBProjMan are future projects.

5.6 Conclusions

KUALI-BEH defines a kernel of common concepts that facilitates transformation of practitioners' tacit knowledge into explicit knowledge. The conceptual framework provides understanding of concepts and terms related to software projects, and offers practitioners the means to describe methods as a composition of practices using the language of common concepts.

On the other side, the methodological framework related to software project execution allows practitioners to visualize the project's on-going performance and gives them elements to adjust and adapt their ways of working according to the situation.

Finally, the development of the technological environment that supports KUALI-BEH was presented. Those tools were created with the aim to endorse the usage and spread the application of KUALI-BEH among practitioners.

Chapter 6

Formalization

"O caos é uma ordem por decifrar."
"Chaos is merely order waiting to be deciphered."

José de Sousa Saramago. Writer (1922 – 2010).

This section presents the formalization process of KUALI-BEH, specifically of its common concepts, its properties and its adaptation operations. KUALI-BEH formalization consists of three parts: a Language, an Algebra and a Knowledge representation.

6.1 Background

Precisely specifying the process by which a Software Engineering activity takes place is a challenging task (Wang et al., 2006). Every aspect of software development, particularly in large systems, demands a great deal of knowledge and understanding of the software practitioner (Devanbu and Jones, 1994). Moreover, according to (Edwards, 2003) and (Selfridge et al., 1992) creating software is one of the most knowledge-intensive professions. Therefore, Software Engineering community has been motivated to turn to practitioners in order to collect all the knowledge they have, making it a relevant research line for the discipline.

Since knowledge creation is an uninterrupted process and so as its collection, which leads to the need for a process to manage it. (Schneider, 2009) establishes that knowledge management process should address all of the following tasks:

- To acquire new knowledge
- To transform it from tacit or implicit into explicit knowledge and back again
- To systematically store, disseminate, and evaluate it
- To apply knowledge in new situations

Providing Software Engineering with a knowledge-based approach allows us to create abstract models and reason about them. At this point the wide scope of the discipline becomes an obstacle. Presentation and integration of the knowledge-based approach into the everyday working world of software engineers is a critical challenge for the Knowledge-Based Software Engineering (KBSE) community (Selfridge et al., 1992).

In 1992 (Selfridge et al., 1992) identified three crucial questions to give Software Engineering a knowledge-based focus:

- What part of the software process is targeted?
- What knowledge is applicable and how can it be represented, acquired, and maintained?
- How can we present the knowledge to developers, teams, and managers to improve the quality, cost, and timeliness of software development?

Having answered these questions, researchers can create a model to represent the knowledge of a targeted software process. Once the model of a process is precisely defined in a formal manner, process analysis techniques can be applied to such a model to identify problematic and erroneous steps, or to leverage efficiency improvements (Wang et al., 2006).

For the purposes of this thesis, the approaches used to formalize software project common concepts were:

- Ontologies for common concepts definitions.
- Situational Method Engineering and Sets Theory to state method properties and adaptation operations.

- Description Logics for knowledge representation.

These approaches are described in detail in the next subsections.

6.1.1 Ontologies

According to (Calero et al., 2006b) an important challenge faced by current communities of researchers and practitioners in the field of Software Engineering and technology is the lack of explicit knowledge shared among members of a group/project, with other groups and with other stakeholders.

The ambiguity of the natural language implies mistakes and nonproductive efforts. Ontologies can mitigate these problems and, farther, some authors have intended to use ontologies as a back-bone of software tools and environments (Calero et al., 2006b).

An ontology, defined by (Schneider, 2009), is a data model that represents a set of concepts within a domain and relationships between those concepts. It is used to reason about objects within that domain.

(Ruiz and Hilera, 2006) identified the main usages of ontologies in Software Engineering, and these are:

- To clarify the knowledge structure.
- To reduce conceptual and terminological ambiguity.
- To allow the sharing of knowledge.

Ontologies are a means to conceptualize; in the words of (Ruiz and Hilera, 2006) conceptualization is understood to be *"an abstract and simplified version of the world to be represented: a representation of knowledge based on objects, concepts and entities existing within the studied area, as well as the relationships existing among them"*.

In (Gómez Pérez et al., 2004) the authors have also considered important to add to this definition two new requirements, which resulted in the following specification:

1. Formalized, it is meant that a machine can process it.
2. Shared, it is understood that the knowledge acquired is the consensus of a community of experts.

The benefits of using ontologies identified by (Schneider, 2009) are the following:

- It can be checked for inconsistencies.
- Reasoning can help to detect derived relationships or implicit class memberships.
- Errors can be removed, thus improving the quality of a knowledge base.
- It can be imported and shared.

In Software Engineering there are many examples of application of ontologies in order to understand a specific field of knowledge. Some examples are:

- Engineering of the ontology for the Software Engineering Body of Knowledge (Abran et al., 2006).
- Software development methodologies and endeavours (González-Pérez and Henderson-Sellers, 2006).
- Software maintenance ontology (Dias et al., 2003)
- Software measurement (García et al., 2006)
- An ontological approach to the SQL:2003 (Calero et al., 2006a)

As we mentioned before, ontologies represent knowledge items in the form of concepts, relationships and attributes, which must be expressed as statements. There are different formats and languages to represent statements, for example we can use Resource Description Framework (RDF) (W3C, 2014) or Web Ontology Language (OWL) (W3C, 2012). RDF is a general-purpose language for representing and referencing information on the Web. It is intended for situations in which this information needs to be processed by applications rather than be presented to people directly (Schneider, 2009).

RDF represents simple statements as a graph of resources, their properties and values; based on (Schneider, 2009) an RDF statement consists of three elements:

- **Subject:** Someone or something considered as a resource; it may be any person or item represented by a Uniform Resource Identifier (URI).

- **Predicate:** Indicates the subject's relation to another concept or the subject's activity.
- **Object:** Defines what the subject is related to or what the subject is doing.

As for OWL, it is a markup language for publishing and sharing ontologies on the web built on RDF (Schneider, 2009).

The usage of standardized languages like RDF and OWL helps to define and share ontologies. However an important part of ontologies is the generation of new knowledge from them, which is called reasoning, and a tool that supports applying logic, querying and reasoning with ontologies is called a reasoner. Some examples of reasoners are:

- RACER[1]
- HermiT[2]
- FaCT++[3]

It is important to mention that most of the tools that support management of ontologies comprise more than one module, one of which is a reasoner. For the purposes of this thesis HermiT was chosen as a reasoner.

Section 6.2 presents a KUALI-BEH ontology.

6.1.2 Situational Method Engineering

Situational Method Engineering focuses on configuration of system development methods tuned to the situation of a project at hand (Harmsen and Brinkkemper, 1995). The building blocks of a situational method are called method fragments. In principle, any coherent product, activity, or tool being part of an existing generic or situational method is a method fragment (Harmsen and Brinkkemper, 1995).

According to (Harmsen et al., 1994), Situational Method Engineering contributes to the fulfillment of the following requirements:

[1] `http://www.sts.tuharburg.de/~r.f.moeller/racer/`
[2] `http://hermit-reasoner.com/`
[3] `http://owl.man.ac.uk/factplusplus/`

- **Flexibility:** It assumes that the method to be used in a certain development project is situational, that is, completely tuned to the project situation at hand and results in an approach that is as flexible as possible.

- **Experience accumulation:** The controlled adaptability of the method allows for the addition of project experience. This can be achieved by adapting method building blocks, which are stored in a central data base. This experience can then be used again in new projects.

- **Integration and communication:** All methods and building blocks are stored in one common repository.

- **Quality:** The fact that flexibility should be controlled guarantees that the constructed situational method meets the same quality requirements as standard methods do.

In order to satisfy those requirements a software tool is needed. (Harmsen and Brinkkemper, 1995) identify the functionalities of a tool that will support Situational Method Engineering as:

- **Representation:** allows for the description of method fragments. Method fragments can be described in any kind of language that is able to represent products or processes.

- **Administration:** provides facilities to insert and modify method fragments in the Method Base.

- **Selection:** provides functions to retrieve method fragments from the Method Base. The method engineer should be able to execute queries on the Method Base.

- **Assembly:** allows for the assembly of selected method fragments. Process fragments and product fragments can be combined to form larger components, eventually leading to a situational method.

An initial formalization of method fragments is presented by (Harmsen et al., 1994). This formalism can be organized in four groups: Sets, Predicates, Functions and Rules.

The defined sets are:

$$\mathcal{M} : \text{ the set of method fragments.}$$

$$\mathcal{P} : \text{ the set of process fragments.}$$

$$\mathcal{R} : \text{ the set of product fragments.}$$

And the restrictions that apply to these sets are: $M = P \cup R$ and $P \cap R = \emptyset$

$$M_c : \text{ the set of conceptual method fragments.}$$

$$M_t : \text{ the set of technical method fragments.}$$

P_c, P_t, R_c, and R_t are defined similarly.
Also the following sets are defined as:

$$\mathcal{V} : \text{ the set of textual property values.}$$

$$\mathbb{N} : \text{ the set of natural numbers.}$$

Using the elements of these sets, (Harmsen et al., 1994) defines predicates applied over them:

- **consists_of** over $M \times M$
- **precedes** over $P \times P$
- **requires** over $P \times R$
- **produces** over $P \times R$
- **supports** over $M_t \times M_c$

For example, if we say that x consists of y, then we have that:

$$\textbf{consists\_of}(\boldsymbol{x}, \textbf{y}) \Rightarrow (x \in P \wedge y \in P) \vee (x \in R \wedge y \in R)$$

Two functions are also defined as:

- *goal* over M to V
- *level* over P to $\mathbb{N}$

An example of the function *level* could be:

$$level(\text{“Database Design”}) = 3$$

It means that the granularity level of the fragment “Database Design” is 3. The granularity level is used to search over the *granularity tree*, which is built on the **consists_of** predicate. With this tree, the user knows which fragments are part of a fragment.

Last but not least, the rules are defined as follows:

Rule 1: The level number of a fragment is one higher than the level number of its constituent parts:

$$\forall m_1, m_2 \in M[consists\_of(m_1, m_2) \rightarrow level(m_1) = level(m_2) + 1]$$

Rule 2: The levels of consecutive fragments in a method should not differ more than 2:

$$\forall m_1, m_2 \in M[precedes(m_1, m_2) \rightarrow |level(m_1) - level(m_2)| < 3]$$

Rule 3: If a process fragment requires a product fragment, this should be produced by a preceding process fragment:

$$\forall p_2 \in P\ \forall r \in R\ \exists p_1 \in P[requires(p_2, r) \rightarrow produces(p_1, r) \cap precedes(p_1, p_2)]$$

Rule 4: A conceptual method fragment should always be supported by its corresponding technical method fragment. This requirement will be met if the goals of the fragments are the same:

$$\forall t \in M_t\ \forall c \in M_c[(supports(t, c) \rightarrow goal(t) = goal(c)]$$

Rule 5: Each product fragment should be produced by a process fragment:

$$\forall r \in R\ \exists p \in P[produces(p, r)]$$

This initial formalization was taken as a base for KUALI-BEH operations and properties representation, which is shown in section 6.3.

6.1.3 Description Logics

Description Logics (DL) (Baader et al., 2003) is a family of knowledge representation (KR) formalisms that represents the knowledge of an application domain (Baader and Nutt, 2003).

The basic DL family is the $\mathcal{AL}$-languages (Schmidt-Schaubß and Smolka, 1991), and it is formed by the following syntax rule:

$$C, D \rightarrow A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top$$

It is possible to create statements and build a knowledge base using this language. A knowledge base in DL comprises two components:

- The Terminological Knowledge (TBox)
- The Assertional Knowledge (ABox)

The Terminological Knowledge (TBox) is a collection of concepts and roles. Concepts represent the entities of a *"universe"*, while Roles denote the relations (properties or associations) between these concepts. In other words, the TBox introduces vocabulary of an specific domain.

A basic statement in a TBox is a concept definition, that is, the definition of a new concept in terms of other previously defined concepts. For example, in the context of KUALI-BEH we can define a Practitioner as a Person who is also a Software Engineer, so in DL this concept is represented as follows:

$$\text{Practitioner} \equiv \text{Person} \sqcap \text{SoftwareEngineer}$$

On the other hand, the ABox contains assertions about named individuals in terms of this vocabulary, that is, it contains extensional knowledge about the domain of interest. For example, we can have:

$$\text{Person}(\text{“Miguel”}) \sqcap \text{SoftwareEngineer}$$

It states that the individual "Miguel" is a Person and also a Software Engineer. Given this example for a Tbox, we can derive from it an assertion that *Miguel is a Practitioner.* This assertion now belongs to the ABox.

There are some restrictions in working with DL and according to (Baader and Nutt, 2003) the most important are:

- Only one definition for a concept name is allowed.
- Definitions are acyclic in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

At this point we can observe that Person and Software Engineer are atomic concepts, but DL offers the possibility to build complex descriptions inductively using concept constructors (Baader and Nutt, 2003). By adding more constructors to $\mathcal{AL}$ we can obtain more expressive languages. Table 6-1 shows the spectrum of $\mathcal{AL}$ families.

Table 6-1: DL family of languages, adapted from (Kaplanski, 2008).

Family	Added constructor	Expression
$\mathcal{U}$	Union	C $\sqcup$ D
$\mathcal{E}$	Full existential quantification	$\exists$ R.C
$\mathcal{N}$	At most and at least restrictions	$\leq n$ R, $\geq n$ R
$\mathcal{C}$	Negation	$\neg$ C
$\mathcal{O}$	One of	$a_1, \cdots, a_n$
$\mathcal{I}$	Inverse relation	P,Q,R $\rightarrow$ R$^-$
$\mathcal{Q}$	Qualified number restriction	$\leq n$ R.C, $\geq n$ R.C
$\mathcal{R}$	Complex role inclusion	P $\circ$ Q $\subseteq$ R

Due to the fact that DL is a KR formalism, it is assumed that a KR system should always answer the queries of a user in reasonable time, then the reasoning and decision procedures of DL are their strength (Baader and Nutt, 2003). Therefore a knowledge representation system based on DL consists of four main elements: The TBox and the ABox, plus a reasoner and a user interface, see figure 6-1.

For the purposes of this thesis, the DL-family used is $\mathcal{ALUENC}$, the reasoner is HermiT and the user interface is Protégé[4]. The DL representation of KUALI-BEH is shown in section 6.4.

[4]`http://protege.stanford.edu/`

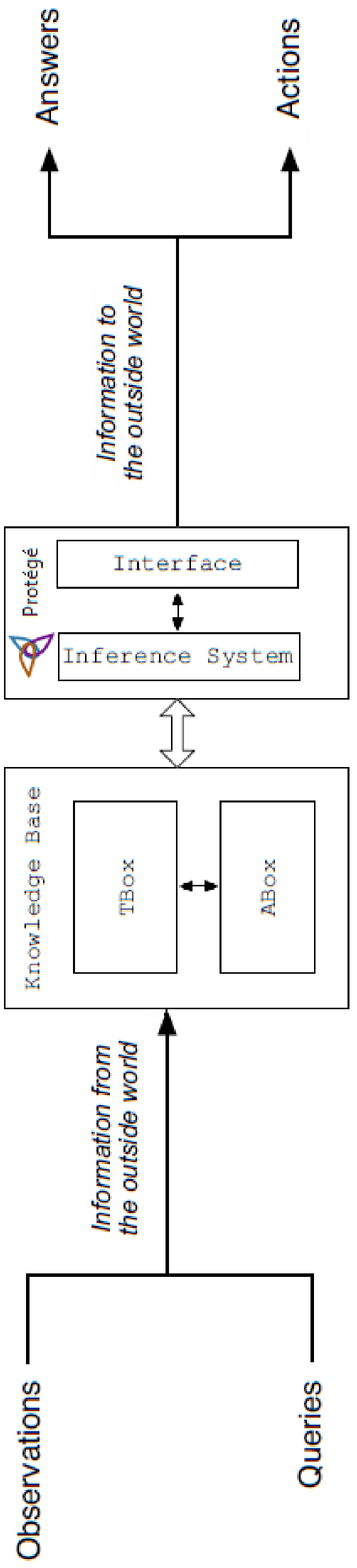


Figure 6-1: Architecture of a system based on DL, adapted from (Baader and Nutt, 2003) and (Wang et al., 2006)

6.2 KUALI-BEH Language

KUALI-BEH language is an initial approach to share a common representation of knowledge as a set of concepts, attributes and relationships of a domain in the form of ontology, abbreviated as KB-O. This ontology is supposed to be used by method engineers as a means to describe, analyze and reason about software projects and related information.

It is important to realize that the ontologies defined using REFSENO serve the purpose of software knowledge management and not as the basis for the implementation of intelligent assistants (Tautz and von Wangenheim, 1998).

REFSENO provides constructs to define concepts with their attributes and relationships between them. The construction of REFSENO ontologies is based on three tables that use text and, optionally, diagrams. The tables contain a glossary of concepts, attributes and relationships respectively. REFSENO allows definition of cardinalities for the relationships and value ranges for the attributes.

In this work we used Representation Formalism for Software Engineering Ontologies (REFSENO) (Tautz and von Wangenheim, 1998) to create KUALI-BEH common concepts ontology, due to the fact that it was created specifically for Software Engineering and because of its simplicity.

The next sections present a general background and the KUALI-BEH ontology requirements specification followed by its definition.

6.2.1 Background of KB-O

The specification of an ontology should contain the domain modeled, the purpose of the ontology, the scope, and administrative information like the authors and knowledge sources (Tautz and von Wangenheim, 1998). Table 6-2 defines KB-O requirements specification.

Table 6-2: KB-O requirements specification.

Domain	Software Projects
Date of creation	June 23, 2012 (Updated November 01, 2014)

KB-O Requirements Specification

Conceptualized by	Miguel Morales Trujillo and Hanna Oktaba.
Purpose	To describe the common concepts involved in software projects and their relationships.
Level of Formality	Semi-formal (UML Diagrams, text and REFSENO tables).

KB-O Requirements Specification

Scope	**List of concepts:** • Activity • Condition • Guide • Input • Knowledge and Skills • Measure • Method • Methods and Practices Infrastructure • Pattern • Practice • Practitioner • Result • Software Project • Stakeholder • Task • Tool • Verification Criteria • Work Product • Work Team

KB-O Requirements Specification

	Instances: • Project Conditions • Software Product • Stakeholder Needs **Attributes:** • Objective (Practice) • Purpose (Method) • Status (Work Product)
Source of Knowledge	See Appendix B.

6.2.2 Definition of KB-O

After establishing KB-O requirements specification, REFSENO suggests a process model to develop the ontology itself. To develop KB-O we used a suggested process model and a Unified Modeling Language (UML) (OMG, 2011d) Class diagram. Note that for the purpose of this proposal, a reduced version of REFSENO is used in order to maintain it readable and easy to assimilate.

The resulting ontology consists of a graphical representation, a UML class diagram, and a textual semi-formal representation of knowledge using REFSENO. Figure 6-2 shows the UML class diagram used to develop the ontology.

This class diagram preserves all the KUALI-BEH common concepts, however it differs from the class diagram presented earlier in Figure 5-2 in the way their associations are represented. The differences between the two figures are due to, in the first place, naming restrictions defined by ontologies and, secondly, the need to represent associations in a machine-consumable format and maximize the usefulness of the metamodel instead of minimizing its complexity, which was

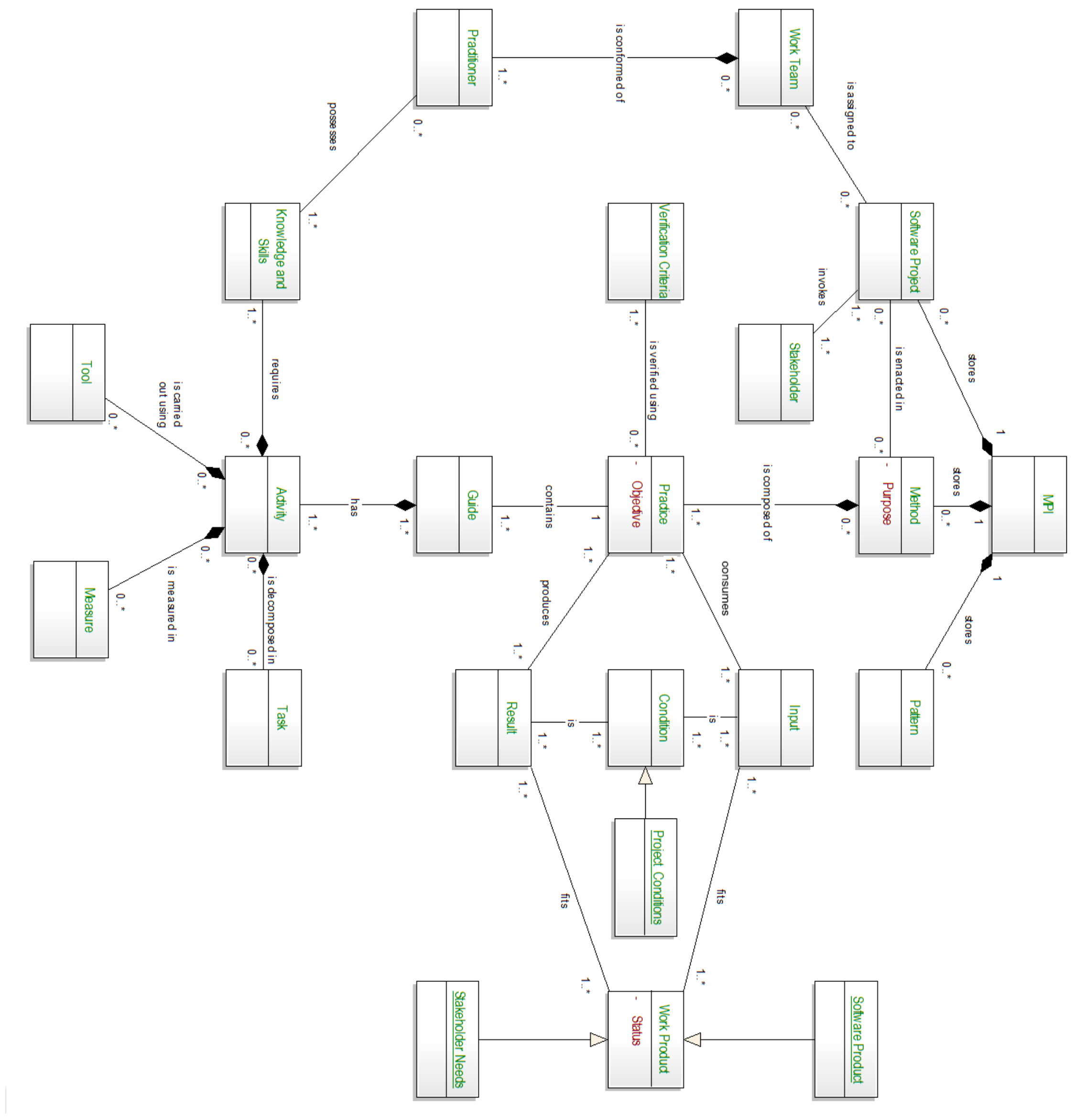


Figure 6-2: KB-O concepts and their relationships and attributes.

the purpose of the first class diagram.

6.2.2.1 Concepts glossary

The concepts glossary lists alphabetically all the concepts of the ontology. One row of the concepts glossary corresponds to one concept. The columns are labeled Name, Definition, Example and References, denoting the respective components of the concept definition. The References column indicates the section of the Appendix B, which contains the sources that were considered and used to create the respective concept.

Table 6-3 displays the glossary of concepts that form the KUALI-BEH ontology.

Table 6-3: KB-O concepts glossary.

Name	**Definition**	**Example**	**References**
Activity	An activity is a set of tasks that contributes to the achievement of a practice objective.	SI.2.2 Document or update the Requirements Specification.	B.1
Condition	A condition is a specific situation, circumstance or state of something or someone with regard to appearance, fitness or working order that have a bearing on the software project.	The team is working together and every member of the team is in context for the coming day's work.	B.3

KB-O concepts glossary

Name	Definition	Example	References
Guide	A guide is a set of recommended activities aimed to resolve a specific objective transforming an input into a result. Particular knowledge and skills are needed to perform the advised activities. The same practice may be carried out following different guides, but they should accomplish the practice objective and preserve their input and result characteristics. The tools to support the guide carrying out could be described optionally.	SI.2.1 Assign Tasks to the Work Team members in accordance with their role, based on the current Project Plan. SI.2.2 Document or update the Requirements Specification. SI.2.3 Verify and obtain approval of the Requirements Specification.	B.5

KB-O concepts glossary

Name	Definition	Example	References
Input	An input is defined as expected characteristics of a work product and/or conditions needed to start the execution of a practice.	Description of work to be done: • Product Description. • General Customer requirements. • Scope description of what is included and what is not. • Deliverables list of products to be delivered to Customer.	B.6

KB-O concepts glossary

Name	Definition	Example	References
Knowledge and Skills	The knowledge and skills are a set of abilities, competences and attainments, acquired by the practitioner and needed to perform a practice.	• Experience eliciting requirements. • Experience in designing user interfaces. • Knowledge of the revision techniques.	B.7
Measure	A measure is a unit or standard of measurement used to assess or to quantitatively estimate the attributes of various aspects.	LOC	B.8
Method	A method is an articulation of a coherent, consistent and sufficient set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.	Software Implementation	B.9

KB-O concepts glossary

Name	Definition	Example	References
Methods and Practices Infrastructure	The methods and practices infrastructure (MPI) is a set of methods and practices learned by the organization members by experience, abstraction or apprehension. This base of knowledge is continuously expanded and modified by the practitioners. It can contain methods, practices organized as families, individual practices or practice patterns. The methods and practices infrastructure is used by the work teams as a source of proven organizational knowledge to define the software projects way of working. It can also be useful in training new practitioners incorporated into the organization.	KB-MPI	B.10
Pattern	A pattern is a set of practices that can be applied as a general reusable solution to a commonly occurring problem within a given context.	Kritarchy Pattern	B.11

KB-O concepts glossary

Name	Definition	Example	References
Practice	A practice is work guidance, with a specific objective, that advises how to produce a result originated from an input. The guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result. The verification criteria associated to the result are used to determine if the objective is achieved. Particular knowledge and skills are required to perform the practice guide, which can be carried out optionally using tools. To evaluate the practice performance and the objectives' achievement, selected measures can be associated to it. Measures are estimated and collected during the practice execution.	Software Requirements Analysis	B.12
Practitioner	A practitioner is a professional in Software Engineering that is actively engaged in the discipline. The practitioner should have the ability to make a judgment based on his or her experience and knowledge.	Miguel	B.13

KB-O concepts glossary

Name	Definition	Example	References
Project Conditions	The project conditions are the factors related to the project that could affect its realization. Complexity, size, time and financial restrictions, effort, cost and other factors of the project environment are considered. It is a specialization of a condition.	KB-Project-Conditions	B.14
Result	A result is defined as expected characteristics of a work product and/or conditions required as outputs after the execution of a practice.	Requirements description: • Functionalities. • User interface. • External interfaces. • Legal and regulative. Each requirement is identified, unique and it is verifiable or can be assessed.	B.15

KB-O concepts glossary

Name	Definition	Example	References
Software Product	A software product is the result of a method execution. It may contain a set of computer programs, procedures, and possibly associated documentation and data. It is a specialization of a work product.	KB-System	B.17
Software Project	A software project is a temporary effort undertaken by a work team using a method in order to develop, maintain or integrate a software product, responding to specific stakeholder needs and under particular conditions. The stakeholder needs, project conditions and, if applies, already existing software products are considered as the input of a software project. The result is a new, modified or integrated expected software product.	KB-Project	B.18
Stakeholder	A stakeholder is an individual or organization having a right, share, claim or interest in a software product or in its possession of characteristics that meet their needs and expectations.	The Client	B.19
Stakeholder Needs	The stakeholder needs are the representation of requirements, demands or exigencies expressed by the stakeholders to the work team.	The Client Needs	B.20

KB-O concepts glossary

Name	Definition	Example	References
Task	A task is a requirement, recommendation or permissible action.	SI.2.2.1 Identify and consult information sources (Customer, users, previous systems, documents, etc.) in order to get new requirements.	B.22
Tool	A tool is a device used to carry out a particular function.	Enterprise Architect	B.23
Verification Criteria	The verification criteria are the parameters used to judge if specified requirements have been fulfilled.	Product Description work product must include: • Purpose • General Customer requirements	B.24
Work Product	A work product is an artifact utilized or generated by a practice. It could have a status associated.	Requirements Specification	B.25

KB-O concepts glossary

Name	Definition	Example	References
Work Team	A work team is a group of practitioners that work together in a collaborative manner to obtain a specific goal. Business experts and other representatives on behalf of a stakeholder can be included in the work team.	KB-WT	B.26

6.2.2.2 Relationships

Relationships model the way in which a particular software engineering entity is related to other software engineering entities and are labeled as follows: Name, Concepts (Cardinality) and Description. The relationships of this ontology are equivalent to the non-terminal concept attributes defined in REFSENO.

Table 6-4 shows the relationships that form KB-O.

Table 6-4: KB-O relationships.

Name	Concepts (Cardinality)	Description
Consumes	Practice (*) — Input (*)	A practice consumes an input.
Contains	Practice (1) — Guide (*)	A practice contains a guide.
Determine	Stakeholder Needs (*) — Software Project (*)	Stakeholder needs determine a software project.
Fits	Work Product (*) — Input (*)	A work product fits an input.
Fits	Work Product (*) — Result (*)	A work product fits a result.

KB-O Relationships

Name	Concepts (Cardinality)	Description
Has	Guide (*) — Activity (*)	A guide has activities.
Is	Condition (*) — Input (*)	A condition is an input.
Is	Condition (*) — Result (*)	A condition is a result.
Is assigned to	Work Team (*) — Software Project (*)	A work team is assigned to a software project.
Is carried out using	Activity (*) — Tool (*)	An activity is carried out using a tool.
Is composed of	Method (*) — Practice (*)	A method is composed of practices.
Is conformed of	Work Team (*) — Practitioner (*)	A work team is conformed of practitioners.
Is decomposed in	Activity (*) — Task (*)	An activity is decomposed in tasks.
Is enacted in	Method (*) — Software Project (*)	A method is enacted in a software project.
Is measured using	Activity (*) — Measures (*)	An activity is measured using measures.
Is produced by the end of	Software Product (*) — Software Project (1)	A software product is produced by the end of a software project.
Is verified using	Practice (*) — Verification Criteria (*)	A practice is verified using verification criteria.
Possesses	Work Team (*) — Knowledge and Skills (*)	A work team possesses knowledge and skills.
Produces	Practice (*) — Result (*)	A practice produces a result.

KB-O Relationships

Name	Concepts (Cardinality)	Description
Requires	Guide (*) — Knowledge and Skills (*)	A guide requires knowledge and skills.
Restrict	Project Conditions (*) — Software Project (*)	Project conditions restrict a software project.
Sets up	Stakeholder (*) — Project Conditions (*)	A stakeholder sets up project conditions.
Sets up	Stakeholder (*) — Stakeholder Needs (*)	A stakeholder sets up stakeholder needs.
Sets up	Stakeholder (*) — Software Product (*)	A stakeholder sets up software product.
Stores	MPI (*) — Pattern (*)	A MPI stores patterns.
Stores	MPI (*) — Method (*)	A MPI stores methods.
Stores	MPI (*) — Practice (*)	A MPI stores practices.
Stores	MPI (*) — Work Product (*)	A MPI stores work products.
Stores	MPI (*) — Conditions (*)	A MPI stores conditions.

6.2.2.3 Attributes

An attribute is represented using the concept attribute table, which is concept-specific and contains one row for every attribute. The columns are labeled as follows: Name, Description, Mandatory, Type and Cardinality. The attributes of this ontology are equivalent to the terminal concept attributes defined in REFSENO.

Table 6-5 presents the attributes that form KB-O.

Table 6-5: KB-O attributes.

Attribute (of Concept)	**Description**	**Mandatory**	**Type**	**Cardinality**
Objective (Practice)	Description of the goal that a practice pursues.	Yes	Text	1
Purpose (Method)	Description of the goal that a method pursues.	Yes	Text	1
Status (Work Product)	Description of the actual state or situation of a work product.	No	Text	1

6.3 KUALI-BEH Algebra

Based on the ideas proposed in (Harmsen et al., 1994), we defined KB-A, which is a set of predicates, functions and operations in order to represent KUALI-BEH common concepts and its method properties and operations.

6.3.1 KB-A axioms and definitions

The KB-A axioms and definitions are expressed as follows:

Axiom 1 *Let $\mathcal{MPI}$ be all the things that can be created using KUALI-BEH.*

$$\mathcal{MPI} = \{\mathcal{M} \oplus \mathcal{P} \oplus \mathcal{J} \oplus \mathcal{W} \oplus \mathcal{C}\}$$

Where:

$$\mathcal{M} = \{m \mid m \text{ is a method}\}$$

$$\mathcal{P} = \{p \mid p \text{ is a practice}\}$$

$$\mathcal{J} = \{j \mid j \text{ is a software project}\}$$

$$\mathcal{W} = \{w \mid w \text{ is a work product}\}$$

$$\mathcal{C} = \{c \mid c \text{ is a condition}\}$$

Axiom 2 *$\mathcal{W}$ and $\mathcal{C}$ are disjoint sets.*

$$\mathcal{W} \cap \mathcal{C} = \emptyset$$

Definition 3 *Let $p_1, \cdots, p_n$ be practices, a method m composed of this set of practices can be expressed as $m = \{p_1, \cdots, p_n\}$*

Definition 4 *The definition of any element of the KUALI-BEH Static view is a conceptual definition and is denoted by the letter "c" as super-index. For example, the conceptual definition of a work product w is denoted as w^c, which is related to its characteristics.*

Definition 5 *The definition of any element of the KUALI-BEH Operational view is a technical definition and is denoted by the letter "t" as super-index. For example, the technical definition of a work product w is denoted as w^t, which is its instance.*

Definition 6 *Practitioners are the only individuals who can determine:*

- *If similarity between inputs and results is held.*
- *If the method purpose is fully achieved.*
- *If the objective of a practice supports a method purpose.*

This ability corresponds to the practitioner's judgment.

6.3.2 KB-A functions and predicates

In this section we explain the functions defined in the KUALI-BEH Algebra.

The objective of a practice p is obtained through the function *objective* applied to the conceptual definition of the practice.

$$\text{objective} : \mathcal{P} \longrightarrow Text$$

In a similar way, the purpose of a method m is obtained through the function *purpose* applied to the conceptual definition of the method.

$$\text{purpose} : \mathcal{M} \longrightarrow Text$$

In the same way, the status of a work product w is obtained through the function *status* applied to the technical definition of the work product.

$$\text{status} : \mathcal{W} \longrightarrow Text$$

In order to decide if the purpose of a method is fully achieved, the function *fully_achieved* is defined.

$$\text{fully\_achieved} : \mathcal{M} \longrightarrow \mathbb{B}$$

The functions *input* and *result* are defined in the same way for methods and for practices. These functions receive a practice or a method and return a set of work products and/or conditions.

$$\text{input} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C}$$

$$\text{result} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C}$$

The KUALI-BEH predicates are defined in the following way:

produces(p_i, r) denotes that a practice p_i produces a result r.

$$\text{produces} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C})$$

consumes(p_j, i) denotes that a practice p_j consumes an input i.

$$\text{consumes} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C})$$

precedes(p_i, p_j) denotes that a practice p_i precedes a practice p_j.

$$\text{precedes} : \mathcal{P} \times \mathcal{P}$$

follows(p_j, p_k) denotes that a practice p_j follows a practice p_k.

$$\text{follows} : \mathcal{P} \times \mathcal{P}$$

supports(p, m) denotes that the objective of a practice p supports the purpose of a method m.

$$\text{supports} : \mathcal{P}^c \times \mathcal{M}^c$$

6.3.3 KB-A method properties

In order to represent method properties, as stated in Definition 3, let us define a method $m \in \mathcal{M}$ as a set:

$$m = \{p \mid p \ \in \mathcal{P}\}$$

Coherency, consistency and sufficiency properties of a method are defined in the next subsections.

6.3.3.1 Coherency

Let us define a function named *coherency*, which receives a method and returns *true* if it is coherent or *false* if it is not.

$$\text{coherency} : \mathcal{M} \longrightarrow \mathbb{B}$$

This function **coherency(m)** is evaluated as follows:

$$\mathit{coherency}(m) = \begin{cases} \mathit{true} & \text{If for all practices } p_i \text{ of a method } m\text{, the objective of } p_i \\ & \text{ supports the purpose of } m. \\ \mathit{false} & \text{Otherwise} \end{cases} \tag{6-1}$$

In other words we have:

$$\text{If } \forall p \in m,\ \text{supports(objective}(p^c)\text{, purpose}(m^c)) = \mathit{true}$$

6.3.3.2 Consistency

Let us define the function *consistency*, which receives a method and returns *true* if it is consistent or *false* if it is not.

$$\text{consistency} : \mathcal{M} \longrightarrow \mathbb{B}$$

This function **consistency(m)** is evaluated as follows:

$$consistency(m) = \begin{cases} true & \text{If all the practice inputs are produced and all the practice} \\ & \text{ results are consumed. Except for the method input and result.} \\ false & \text{Otherwise} \end{cases} \tag{6-2}$$

In other words we have:

$$\text{If } \forall p_1 \in m\ \exists p_2, p_3 \in m\ (\text{produces}(p_1,\ r) \rightarrow \text{consumes}(p_2,\ r) \vee \text{result}(m^c) = r)$$
$$\wedge\ (\text{consumes}(p_1,\ i) \rightarrow \text{produces}(p_3,\ i) \vee \text{input}(m^c) = i)$$

6.3.3.3 Sufficiency

Let us define the function *sufficiency*, which receives a method and returns *true* if it is sufficient or *false* if it is not.

$$\text{sufficiency} : \mathcal{M} \longrightarrow \mathbb{B}$$

This function **sufficiency(m)** is evaluated as follows:

$$sufficiency(m) = \begin{cases} true & \text{If the method } m \text{ is coherent, consistent and its purpose is} \\ & \text{ fully achieved.} \\ false & \text{Otherwise} \end{cases} \tag{6-3}$$

In other words we have:

$$\text{If coherency}(m) \wedge \text{consistency}(m) \wedge \text{fully\_achieved}(m) = true$$

6.3.4 KB-A adaptation operations

In order to characterize the operations of adaptation, let us define a practice P as a triple formed by an Input (I), an Objective (O) and a Result (R)

$$P = (I, O, R)$$

The operations of substitution, concatenation, combination and splitting are defined in the next subsections.

6.3.4.1 Substitution

The *substitution* of practices consists in replacing a practice by another equivalent practice.

Let $P_1 = (I_1, O_1, R_1)$ and $P_2 = (I_2, O_2, R_2)$ be practices
P_1 can be *substituted* by P_2 if and only if
$$P_1 \equiv P_2$$

The equivalence between practices holds when similar results are reached starting from similar inputs and similar objectives are fulfilled. Notice that similarity is recognized and dictated by the practitioner's judgment.

After applying the adaptation operation the original properties of a method are preserved, because of the fact that a new practice holds an objective, input and result similar to the substituted practice.

6.3.4.2 Concatenation

If one practice has a result similar to the input of another practice, both can be integrated into one practice, applying the *concatenation* operation. The original objectives of the two practices are joined into one final objective.

Formally, the concatenation operation is defined as follows:

Let $P_1 = (I_1, O_1, R_1)$ and $P_2 = (I_2, O_2, R_2)$ be practices and R_1 is similar to I_2
A practice P_3 is a correct *concatenation* of the practices P_1 and P_2 if:
$$P_3 = (I_1, O_1 \wedge O_2, R_2)$$

The concatenation operation can be applied as many times as required.

6.3.4.3 Combination

A *combination* of practices consists in bringing two different practices into one. The resulting practice preserves the original objectives and an integrated guide, which is formed by the activities taken from both original practices.

Formally, the combination operation is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices}\\ P = (I, O, R) \text{ is a correct } \textit{combination} \text{ of } P_1 \text{ and } P_2 \text{ if:}\\ I \text{ is similar to } I_1 \cup I_2 \text{ and}\\ R \text{ is similar to } R_1 \cup R_2 \text{ and}\\ O \equiv O_1 \wedge O_2\end{gathered}$$

6.3.4.4 Splitting

A *splitting* of practices consists in the partition of the original practice into two different practices preserving the original objective and similar inputs and results.

Formally, the splitting operation is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2)\text{be practices}\\ P_1 \text{ and } P_2 \text{ are a correct } \textit{split} \text{ of } P = (I, O, R) \text{ if:}\\ I_1 \cup I_2 \text{ is similar to } I \text{ and}\\ R_1 \cup R_2 \text{ is similar to } R \text{ and}\\ O_1 \wedge O_2 \equiv O\end{gathered}$$

If operations of substitution, concatenation, splitting and combination are applied to practices strictly following the mentioned rules, the original properties of coherency, consistency and sufficiency of a method are preserved.

6.4 KUALI-BEH Knowledge

DL is a popular formalism for ontologies and it is regarded as the foundation of some ontology languages (Wang et al., 2006), due to the fact that ontologies contain knowledge about a domain in a precise and unambiguous manner (Wang et al., 2006). DL has been shown as common language for ontologies appearing in Software Engineering process that needs support from Knowledge Engineering and the natural successor in terms of evolution of UML (Kaplanski, 2008).

Based on KB-O, we present KB-K, the knowledge representation of KUALI-BEH using DL, in this section.

6.4.1 Understanding KB-K

To understand better and follow the process of creating KB-K, we will use some examples.

Let us consider the concept *practitioner* defined in table 6-3 of KB-O. This concept can be defined using sets as:

$$\{x \mid Practitioner(x)\}$$

To express roles we proceed in a similar way, let us consider the relationship *isConformedOf* presented in table 6-4. This relationship expresses that a work team is conformed of practitioners, and it can be denoted as follows:

$$\{(x, y) \mid isConformedOf(x, y)\}$$

Now let us transform these sets into First Order Logic (FOL) predicates, which is a common transformation process of semantic networks and ontologies. Let us assume that every work team in our organization must be conformed of practitioners. This assertion can be rewritten in FOL as:

$$\forall x.\text{WorkTeam}(x) \rightarrow \exists y.isConformedOf(x, y) \ \wedge \ \text{Practitioner}(y)$$

Then, this predicate can be rewritten again, but now in DL:

$$\text{WorkTeam} \sqsubseteq \forall isConformedOf.\text{Practitioner}$$

In a similar way, the inverse relationship of *isConformedOf* can be represented as the relationship *belongsTo*. This expresses that each practitioner in the organization belongs to a work team, and can be represented as follows:

$$\text{Practitioner} \sqsubseteq \exists belongsTo.\text{WorkTeam}$$

The creation process of KB-K used two DL association types:

- **has-part:** This type is equivalent to the aggregation/composition in UML.
- **is-a:** This type is equivalent to the generalization in Entity-Relationship model or to inheritance in Object Oriented paradigm.

For example, we identified in KUALI-BEH that an Activity is composed by four elements: Knowledge and Skills, Tasks, Tools and Measures. Then, to express in DL that an Activity *has-parts* we have:

$$\begin{gathered}\text{Activity} \sqsubseteq (\exists\ requires.\text{KnowledgeAndSkills}) \wedge \\ ((\exists\ isDecomposedIn.\text{Task}) \vee \\ (\exists\ isCarriedOutUsing.\text{Tool}) \vee \\ (\exists\ isMeasuredIn.\text{Measure})\ \vee\ true)\end{gathered}$$

Notice that only Knowledge and Skills is mandatory, while Task, Tool and Measure are not.

In KUALI-BEH the *Stakeholder Needs* is a specialization of a *Work Product*. Then, we can say that StakeholderNeeds *is-a* WorkProduct, which is written in DL as:

$$\text{StakeholderNeeds} \sqsubseteq\ WorkProduct$$

This association is used for the three instances defined in KB-O: Stakeholder Needs, Project Conditions and Software Product.

To represent an attribute, for example, we know that a WorkProduct has an attribute named *status* and its datatype is String, so we have:

$$\{x \mid WorkProduct(x)\ \wedge\ (\exists s.Status(x, s)\ \wedge\ String(s))\}$$

In DL the datatype equivalent to String is Text, so we can represent a WorkProduct in DL as follows:

$$\text{WorkProduct} \sqsubseteq (= 1\ (status.Text))$$

Let us examine a more general example, we have a work product named *Database Model* and its status is *Draft*, so in DL we have:

$$\text{WorkProduct(“DatabaseModel”)}\ \wedge\ \text{Status(“DatabaseModel”, “Draft”)}\ \wedge\ \text{Text(“Draft”)}$$

6.4.2 Definition of KB-K

After having transformed KB-O into DL expressions, we generated KB-K, which is defined as follows:

Activity

$\text{Activity} \sqsubseteq (\exists\ guide.\text{Guide}) \wedge$
$(\exists\ requires.\text{KnowledgeAndSkills}) \wedge$
$((\exists\ isDecomposedIn.\text{Task}) \vee$
$(\exists\ isCarriedOutUsing.\text{Tool}) \vee$
$(\exists\ isMeasuredIn.\text{Measure})\ \vee\ true)$

Condition

$\text{Condition} \sqsubseteq (= 1\ (mpi.\text{MPI})) \wedge$
$(\exists\ input.\text{Activity}) \vee$
$(\exists\ result.\text{Activity})$

Guide

$\text{Guide} \sqsubseteq (\exists\ practice.\text{Practice}) \wedge$
$(\exists\ has.\text{Activity})$

Input

$\text{Input} \sqsubseteq (\exists\ practice.\text{Practice}) \wedge$

$(\exists\ is.\text{Condition}) \vee$

$(\exists\ fits.\text{WorkProduct})$

Knowledge and Skills

$\text{KnowledgeAndSkills} \sqsubseteq (\exists\ practitioner.\text{Practitioner}) \vee$

$(\exists\ activity.\text{Activity})$

Measure

$\text{Measure} \sqsubseteq (\geq 1\ activity.\text{Activity})$

Method

$\text{Method} \sqsubseteq (= 1\ (mpi.\text{MPI})) \wedge$

$(\exists\ isComposedOf.\text{Practice}) \wedge$

$(\exists\ isEnactedIn.\text{SoftwareProject}) \wedge$

$(= 1\ (purpose.\text{Text}))$

MPI

$\text{MPI} \sqsubseteq (\exists\ stores.\text{SoftwareProject}) \vee$

$(\exists\ stores.\text{Method}) \vee$

$(\exists\ stores.\text{Pattern}) \vee$

$(\exists\ stores.\text{WorkProduct}) \vee$

$(\exists\ stores.\text{Condition})$

Pattern

$\text{Pattern} \sqsubseteq (= 1\ (mpi.\text{MPI}))$

Practice

$\text{Practice} \sqsubseteq (\exists\ method.\text{Method}) \wedge$

$(\exists\ consumes.\text{Input}) \wedge$

$(\exists\ produces.\text{Result}) \wedge$

$(\exists\ isVerifiedUsing.\text{VerificationCriteria}) \wedge$

$(\exists\ contains.\text{Guide}) \wedge$

$(= 1\ (objective.Text))$

Practitioner

$\text{Practitioner} \sqsubseteq (\exists\ workTeam.\text{WorkTeam}) \wedge$

$(\exists\ possesses.\text{KnowledgeAndSkills})$

Project Conditions

$\text{ProjectConditions} \sqsubseteq\ Condition$

Result

$\text{Result} \sqsubseteq (\exists\ practice.\text{Practice}) \wedge$

$((\exists\ is.\text{Condition}) \vee$

$(\exists\ fits.\text{WorkProduct}))$

Software Product

$\text{SoftwareProduct} \sqsubseteq\ WorkProduct$

Software Project

$\text{SoftwareProject} \sqsubseteq (= 1\ (mpi.\text{MPI})) \wedge$

$(\exists\ workteam.\text{WorkTeam}) \wedge$

$(\exists\ stakeholder.\text{Stakeholder}) \wedge$

$(\exists\ method.\text{Method})$

Stakeholder

$\text{Stakeholder} \sqsubseteq (\exists\ invokes.\text{SoftwareProject})$

Stakeholder Needs

$\text{StakeholderNeeds} \sqsubseteq\ WorkProduct$

Task

Task $\sqsubseteq (\geq 1\ activity.\text{Activity})$

Tool

Tool $\sqsubseteq (\geq 1\ activity.\text{Activity})$

Verification Criteria

VerificationCriteria $\sqsubseteq (\exists\ practice.\text{Practice})$

Work Product

WorkProduct $\sqsubseteq (= 1\ (mpi.\text{MPI})) \wedge$
$((\exists\ input.\text{Activity}) \vee$
$(\exists\ result.\text{Activity})) \wedge$
$((= 1\ (status.Text)) \vee true)$

Work Team

WorkTeam $\sqsubseteq (\exists\ isConformedOf.\text{Practitioner}) \wedge$
$(\exists\ isAssignedTo.\text{SoftwareProject})$

Figure 6-3 shows the resulting KB-K class tree generated in Protégé.

6.4.3 Inferring knowledge using KB-K

As it was mentioned in section 6.1.3, DL systems not only stores terminologies and assertions, but also offers services to reason about them. Typical reasoning tasks are to determine whether a description is satisfiable (i.e., non-contradictory) (Baader and Nutt, 2003). In order to show how KB-K can be used, an example is presented in the next subsections.

6.4.3.1 Source of the example

The example was taken from Annex C: Case Studies and Examples of (OMG, 2008b). It represents a fragment of a typical information system delivery process done in Fujitsu DMR

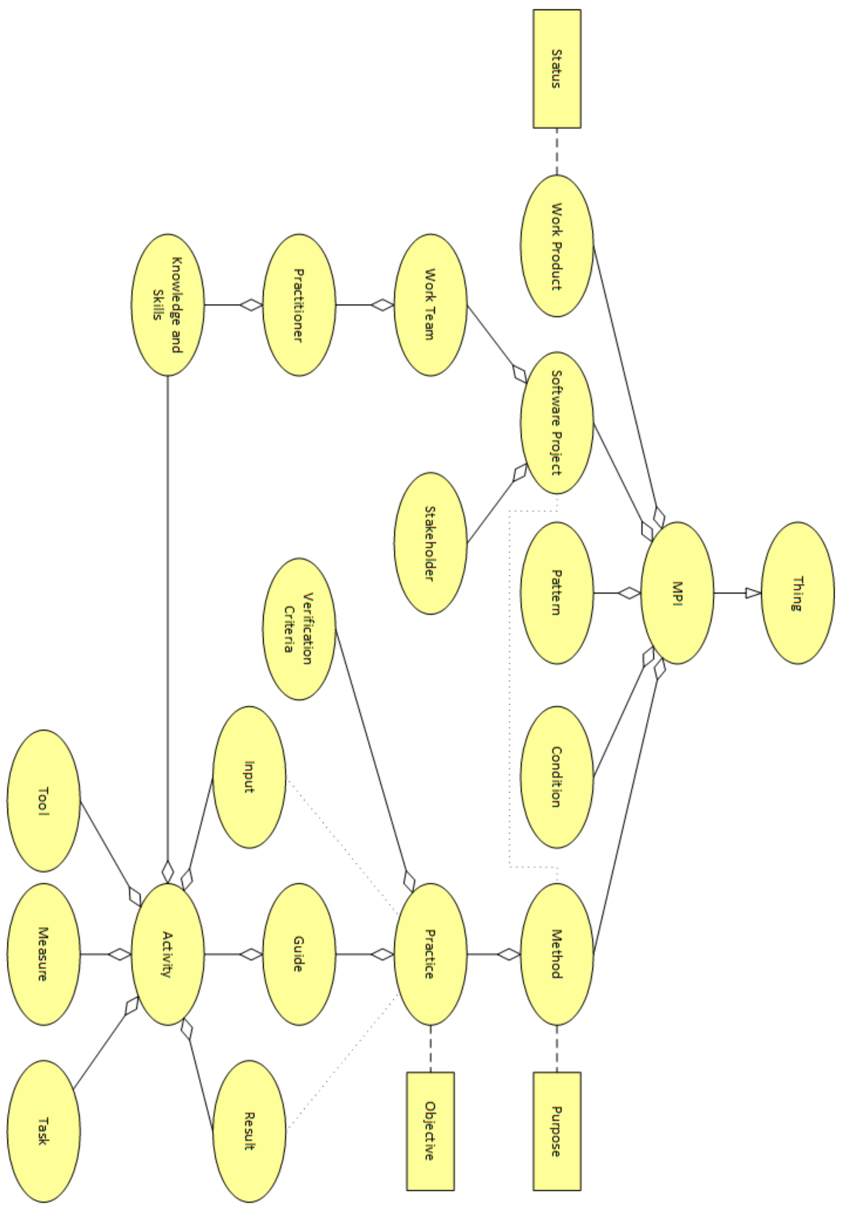


Figure 6-3: KB-K inferred model.

Macroscope modeled in SPEM. The specific SPEM elements appear in bold fonts.

- **Activity {kind: Phase}** : Preliminary Analysis (PA)
  - **Process** : Information System Delivery Process (ISDP)
    - **Activity {kind: Iteration}** : First Joint Requirements Planning Workshop (FJRPW)
      - **TaskUse** : Define Owner Requirements (DOR)
        - **RoleUse** : System Architect (SA)
        - **WorkDefinitionParameter {kind : in}**
          - **WorkProductUse** : Enterprise Architecture (EA)
        - **WorkDefinitionParameter {kind : out}**
          - **WorkProductUse** : Assessment of Current System (AOCS) {state: initial draft}
          - **WorkProductUse** : Owner Requirements (OR) {state: initial draft}
        - **Steps**
          - **Step** : Define objectives based on stated needs (DOBOSN)
          - **Step** : Define the key issues (DTKI)
          - **Step** : Determine the relevant enterprise principles (DTREP)

6.4.3.2 Example of KB-K usage

To provide a comparison between SPEM and KUALI-BEH, two mappings were carried out. On the one hand, a partial mapping between the KUALI-BEH concepts and the SPEM elements was done, see sections C.1 and C.2.

Table 6-6 shows a subset of the mapping, which contains only the required concepts for the purpose of the example.

On the other hand, a mapping between MOF and KUALI-BEH was performed to determine levels of modeling; in our case the example achieves MOF level 0 (Data level), which shows a process that is really enacted on a given project. For the full mapping see section C.3.

Based on the mappings and KB-K, we obtain the following expressions, which are equivalent to the process fragment presented in the previous subsection:

Table 6-6: Mapping between KUALI-BEH and SPEM concepts.

SPEM	*KUALI-BEH*
Phase	Method
Process	Practice
Iteration	Guide
TaskUse	Activity
RoleUse	KnowledgeAndSkills
WorkProductUse	WorkProduct
Step	Task

Method(“PA”) ∧ isComposedOf(“PA”, “ISDP”) ∧

Practice(“ISDP”) ∧ contains(“ISDP”, “FJRPW”) ∧

Guide(“FJRPW”) ∧ has(“FJRPW”, “DOR”) ∧

Activity(“DOR”) ∧ requires(“DOR”, “SA”) ∧

KnowledgeAndSkills(“SA”) ∧ isDecomposedIn(“DOR”, “DOBOSN”) ∧

isDecomposedIn(“DOR”,“DTKI”) ∧ isDecomposedIn(“DOR”, “DTREP”) ∧

Task(“DOBOSN”) ∧ Task(“DTKI”) ∧ Task(“DTREP”) ∧

input(“DOR”, “EA”) ∧ WorkProduct(“EA”) ∧

Status(“EA”, “initial draft”) ∧ Text(“initial draft”) ∧

result(“DOR”, “AOCS”) ∧ WorkProduct(“AOCS”) ∧

Status(“AOCS”, “initial draft”) ∧ Text(“initial draft”) ∧

result(“DOR”, “OR”) ∧ WorkProduct(“OR”)

This example is part of the “proof of concept” of KUALI-BEH formalization, which satisfiability was demonstrated using Protégé and HermiT.

6.4.3.3 Results of the example

In (Wang et al., 2006) it was implied that the process was not complete pointing five inconsistencies related to:

- *Phase* and *Iteration* do not have a performer, since both elements are an specialization of an Activity they must have at least one performer each.
- the *WorkProductUse*'s kinds are not clear, since they are defined through the *WorkDefinitionParameter*.

Our example demonstrates how the SPEM inconsistencies found by (Wang et al., 2006) can be corrected. These inconsistencies are solved by KUALI-BEH because:

1. KUALI-BEH has a well-defined hierarchical approach.
2. KUALI-BEH does not have reflexive associations.

Although direct reasoning about models built on metamodels like SPEM is difficult and it is hard to keep these models consistent (Wang et al., 2006), KUALI-BEH is capable of representing ways of working through a clear model that preserves its structure, consistency and completeness.

6.5 Intended usages

According to (Gruninger and Lee, 2002), the main objectives of formalizations are to improve communication, computational inference and reuse of knowledge. As a whole, formalization of KUALI-BEH (KB-K, KB-O and KB-A) is a starting point to motivate and improve these aspects, giving practitioners and organizations a viable alternative to structure their wide tacit knowledge.

KB-O, the ontology, will permit to unify the vocabulary improving communication among humans, consequently discussions and agreements over an specific domain will be possible.

Applying KB-A, its rules and properties, we can manipulate knowledge in a uniform and standardized manner and achieve communication between humans and computers.

With KB-K that comprises a knowledge base and inference rules, the communication between software tools will foster exchange and analysis of data. Moreover, providing a structure and a management mechanism to the knowledge involved in software projects, we will be able to make computational inferences. Besides, practitioners will have means to evaluate and enrich it.

It is important to keep in mind that the target audience of KUALI-BEH formalization are process engineers; it is supposed that this formalization will be the means of description, analysis and reasoning about software projects.

According to (Zhang and Zhang, 2007), an advantage for knowledge representation is the coupling between theory and practice. In fact, KUALI-BEH was born as a proposal to bridge the gap between theory and practice, providing means to structure and reason about practitioners' knowledge, which makes it possible to enrich the Software Engineering body of knowledge together with process engineers.

Finally, the main intended usage of the formalization is to establish the basis for creating a Computer-Aided Methods Engineering (CAME) tool. There is a necessity of creating a tool that would fulfill the needs defined by (Harmsen et al., 1994) or (Abad et al., 2010), and which were never completely achieved,and in the words of (Arni-Bloch, 2005) *"Unfortunately none of these tools can express the process part of a method and support the enactment of the method process mode"*. We hope that KBTool together with KB-K, A and O will become a tool that really helps practitioners to carry out the processes of method authoring and enactment.

6.6 Conclusions

KUALI-BEH formalization preserves the foundations of Situational Method Engineering and Ontologies. On the one hand it defines the common concepts required to express the practitioners' ways of working enacted during software projects by taking advantage of the characteristics of ontologies.

On the other hand, it permits the adaptation of its elements, practices and methods, in a controlled manner using the KUALI-BEH adaptation operations and properties defined as a set of axioms, definitions, predicates and functions. Following these rules it is possible to customize, modify and assemble complex elements from other elements.

Using the KUALI-BEH adaptation operations it is possible to customize a management process in a controlled manner. We can create, read, update and delete elements, as well as define operations to modify and assemble complex elements from other elements.

Moreover, the hierarchical organization of KUALI-BEH common concepts maintains consistent a granularity level of its elements, no matter if it is analyzed isolated or as a part of a whole.

Finally, through DL, a knowledge representation formalism, which is able to capture virtually almost all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases (Zhang and Zhang, 2007), we can improve communication among humans and machines, allow for computational inference and promote the reuse of knowledge.

We can conclude that with KUALI-BEH formalization it is possible to build a knowledge base with the following characteristics: (i) it has the necessary knowledge (completeness); (ii) the knowledge is faithful to the real world (correctness); (iii) the knowledge is not self-contradictory (consistency); and (iv) the system has efficient algorithms to perform inferences (competence), which, according to (Devanbu and Jones, 1994), are the stringent requirements for a knowledge base.

Part III

KUALI-BEH Validation

Chapter 7

Collaborative Workshop on Software Engineering Methods

"Il tempo dura abbastanza a lungo per chiunque lo userà."
"Time stays long enough for anyone who will use it."

Leonardo da Vinci. Painter, sculptor, architect, mathematician, inventor, anatomist, geologist, cartographer, botanist and writer (1452 – 1519).

KUALI-BEH was validated through a collaborative workshop attended by active practitioners from industry and academy. This chapter presents a detailed description of this workshop.

7.1 Background

Conducting a collaborative workshop on Software Engineering Methods brought about valuable feedback on and improvements to KUALI-BEH. It was attended by 16 participants, practitioners and method engineers from 3 software industry organizations, in addition to 3 Master's students. The workshop methodology included on-site and virtual interactive sessions, together with activities and surveys that the participants had to carry out in order to apply the proposal

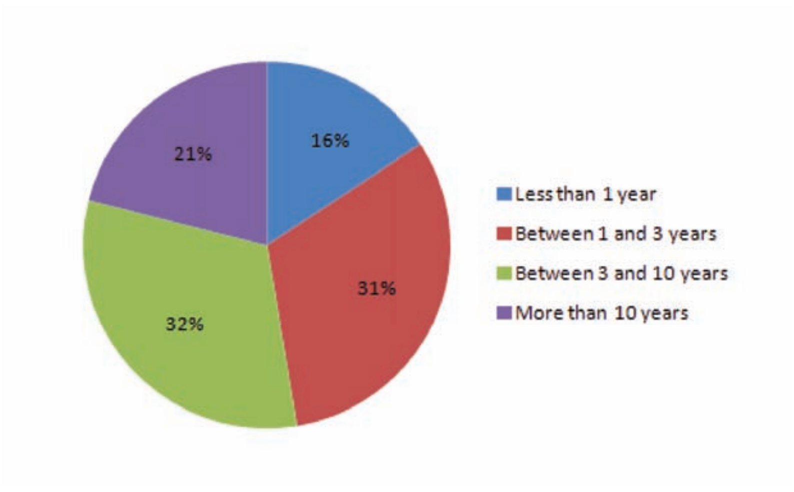


Figure 7-1: Experience in years of participants.

in real life situations and analyze its usefulness, merits and flaws. The workshop took place from March till August 2012.

The participants' experience as active Software Engineering practitioners is shown in Figure 7-1.

The roles developed by practitioners in an organization are shown in Figure 7-2.

7.2 Problem investigation

The objectives of the study were focused on assessing pertinence, appropriateness and proficiency of the KUALI-BEH common concepts. These characteristics were evaluated by asking the following questions:

- Are the common concepts pertinent?
- Are the common concepts definitions appropriate?
- Are the common concepts definitions similar to the ones used in the real context?
- Are the common concepts sufficient?

These questions helped us to make decisions during the acceptance/rejection process, carried out at the Identification phase.

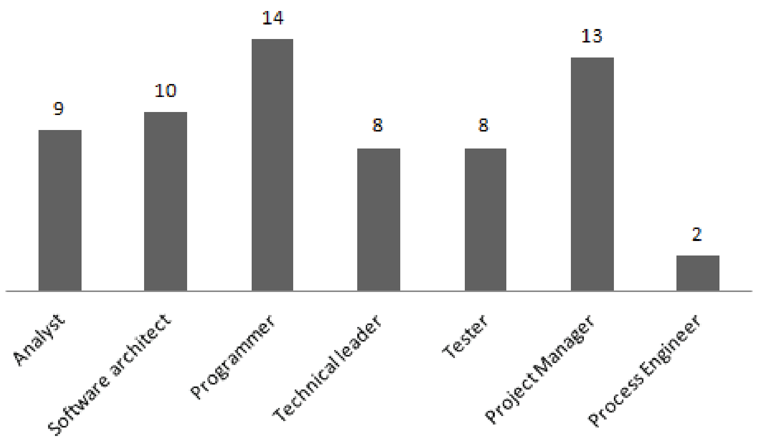


Figure 7-2: Roles developed by the participants in their organizations.

7.3 Artifact design

The artifact to be validated through the workshop was the first version of KUALI-BEH, which was the output of the Identification phase of this research, and corresponded to its first engineering cycle. This version was baselined on February 20th, 2012.

7.4 Design and Implementation

The workshop was divided into eight two-hour sessions that took place every two weeks. The KUALI-BEH theoretical components presented to the participants were:

- **Session 1:** Motivation, background and overview of the framework.
- **Session 2:** The Static view, the definitions of its 20 common concepts and its relationships and the graphical representation.
- **Session 3:** Method properties and the common concepts templates.
- **Session 4:** The Operational view, the Method Enactment and the Practice Instance Lifecycle.
- **Session 5:** The adaptation operations.

- **Session 6:** The Operational boards.
- **Session 7:** Getting it all together.
- **Session 8:** Analysis of results and closure of the workshop.

During the workshop three approaches were used in order to obtain feedback from participants: application of surveys, direct interaction with participants during sessions, which included Q&A runs, and direct observation of activities. Also "homework" activities helped to analyze comprehension and usefulness of the proposal.

During the first part of each session, the instructors presented the theoretical part of KUALI-BEH, and during the second part the participants took part in the following activities:

- **Activity 1:** Document a practice that you execute in your daily work using the practice template.
- **Activity 2:** Document a method and its respective practices that you execute in your daily work using the method template.
- **Activity 3:** Discuss the differences and similarities between real life and the proposed method enactment.
- **Activity 4:** Apply operations in order to adapt the previously documented method.
- **Activity 5:** Adapt the practice instance and method enactment boards to your daily work.
- **Activity 6:** Carry out the method in your organization.

After each session a survey was available online until the next session, during which the results of the most recent survey were analyzed.

- **Survey 1:** Similarity between the proposal and real life.
- **Survey 2:** Pertinence, appropriateness and proficiency of the common concepts.
- **Survey 3:** Pertinence and appropriateness of the method properties.

- **Survey 4:** Pertinence and appropriateness of the practice instance lifecycle and method enactment.
- **Survey 5:** Pertinence and appropriateness of the method adaptation and proficiency of the operations.
- **Survey 6:** Pertinence and appropriateness of the practice instance and method enactment boards.

An example of a survey is presented in Figure 7-3, in this case, the practitioner was inquired about the appropriateness of the common concepts definitions.

7.5 Evaluation

As a result, the proposal was improved by taking into account 93 suggestions from the workshop participants. The suggestions were obtained by surveying the participants or were directly expressed by the participants during the sessions.

The suggestion review process consisted of 4 steps:

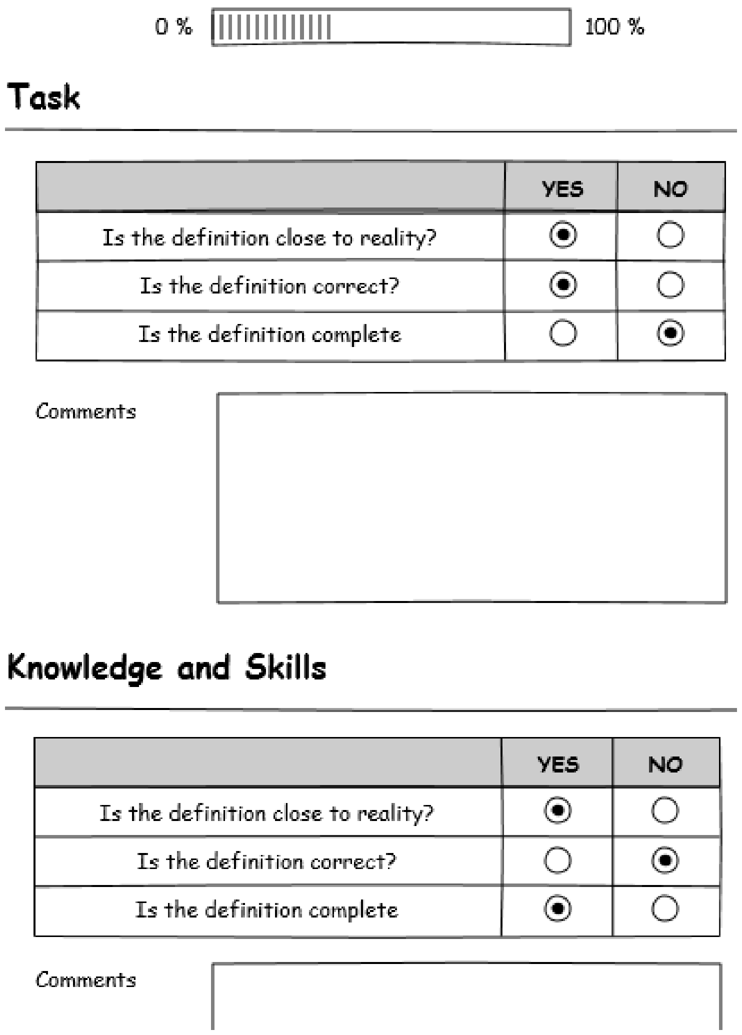
1. First, each of 93 suggestions was associated with its related KUALI-BEH element.
2. During the second step we determined which of them would be rejected, either because they did not apply or were duplicated, thus cutting the previous number to 32 suggestions.
3. The remaining suggestions were later divided into Change/Improvement or Out-of-Scope, thus making the cut 27.
4. Finally, after reviewing and analyzing the 27 suggestions, the KUALI-BEH team fully applied 16, while 11 were included with some modifications.

Figure 7-4 shows the distribution of the obtained suggestions.

A summary of the surveys' results is presented in Tables 7-1 to 7-8.

Tables 7-1 and 7-7 show the mean and average results of the answers that practitioners gave to each statement. The survey used the Likert scale in its traditional five-level format:

1. Strongly disagree



Task

	YES	NO
Is the definition close to reality?	[✓]	
Is the definition correct?	[✓]	
Is the definition complete		[✓]

Comments

Knowledge and Skills

	YES	NO
Is the definition close to reality?	[✓]	
Is the definition correct?		[✓]
Is the definition complete	[✓]	

Comments

Figure 7-3: Fragment of a survey applied to workshop participants.

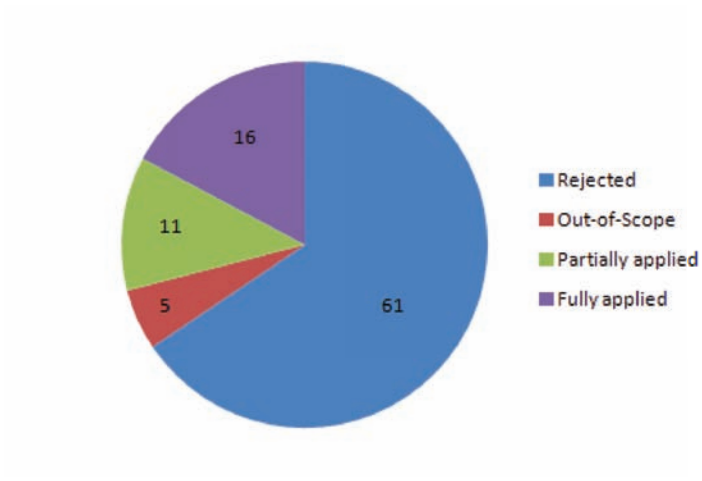


Figure 7-4: Classification of the obtained suggestions.

2. Disagree
3. Neither agree nor disagree
4. Agree
5. Strongly agree

Each value of the first three columns presented in Tables 7-2, 7-3, 7-4, 7-5, 7-6 and 7-8 reflect the number of "Yes" answers given by the participants, while the values of the fourth column represent the number of comments expressed to a respective element of the survey.

After analyzing the suggestions, the main improved areas of KUALI-BEH can be summarized as follows:

- The definitions of *work team*, *guide* and *method* were enhanced.
- The term *tool* was added as a common concept.
- *Concatenation* was added as a new operation of adaptation.
- The *Combination* operation of adaptation changed its name, previously it was named *Merging*.
- Definitions of Method Enactment and Practice Instance Lifecycle were improved.

Table 7-1: Summary of the similarity between the proposal and real life survey (16 answers).

	Median	Average
The description of the Static view represents the way I work or I have worked.	4	4.00
Common concepts are sufficient to define my way of working.	4	4.00
The UML class diagram that shows the relationships between concepts is easy to understand.	5	4.36
My way of working can be described through practices.	4	4.00
The practice pattern contains enough elements to describe practices.	4	3.94
In the organization where I work, the way of working is handled by projects.	5	4.65
In the organization where I work, the way of working can be described through methods.	4	4.47
In the organization where I work, our work can be organized into practices.	5	4.65
The description of the Operational view represents the way I work or I have worked.	4	3.82
When my team is assigned to a project, how we will work is chosen by us.	4	3.36
The way of working of my team changes according to the characteristics of the project.	4	3.91
In the organization where I work, at the beginning of a project, my team is certain about the most of work units to be performed throughout the project.	4	3.45
The diagrams that represent the Practice Lifecycle and the Method Enactment are easy to understand.	4	3.82
I identify the base of knowledge of the organization where I work.	4	4.21
In the organization where I work, I consider feasible to build an Methods and Practices Infrastructure.	5	4.64
In the organization where I work, the Methods and Practices Infrastructure could be used as a tool to train staff and new members.	5	4.64
Sharing my knowledge and making my way of working available to others makes me feel insecure.	2	2.00
Sharing my knowledge and making my way of working available to others will bring us benefits.	5	4.86

Table 7-2: Summary of the pertinence, appropriateness and proficiency of the common concepts survey (19 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Software Project	18	19	19	1
Method	16	18	19	2
Practice	17	19	19	0
Stakeholder	17	19	19	0
Software Product	16	19	18	1
Stakeholder Needs	16	18	19	1
Project Conditions	17	19	18	0
Work Team	16	19	18	1
Practitioner	16	19	18	1
Input	19	19	19	0
Result	19	19	19	0
Guide	17	19	19	0
Activity	17	19	19	0
Task	16	19	18	1
Knowledge & Skills	16	19	18	1
Work Product	17	19	19	0
Condition	16	19	18	1
Pattern	19	19	19	0
MPI	19	19	19	0

Table 7-3: Summary of the pertinence and appropriateness of the method properties survey (16 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Coherency	14	16	16	0
Consistency	13	16	15	1
Sufficiency	12	16	15	3

Table 7-4: Summary of the pertinence and appropriateness of the practice instance lifecycle survey (13 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Instantiated	10	13	12	3
Can-Start	7	12	11	4
In-Execution	7	13	12	1
In-Verification	7	11	12	2
Stand-By	8	13	11	1
Cancelled	8	12	13	2
Finished	8	12	12	2

Table 7-5: Summary of the pertinence and appropriateness of the method enactment survey (13 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Selected	8	12	13	1
Adapted	10	13	13	2
Ready-to-Begin	10	13	13	0
In-Progress	10	13	13	1
Progress-Snapshot	8	13	12	3
Finished	10	13	13	1

Table 7-6: Summary of the pertinence and appropriateness of the method adaptation survey (7 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Substitution	6	7	7	1
Equivalence	6	7	7	0
Combination	6	7	6	1
Splitting	5	7	6	1

Table 7-7: Summary of the proficiency of the operations survey (7 answers).

	Median	Average
The mathematical notation of the modification operations is easy to understand.	5	4.50
It is sufficient to represent practices as a triplet (Input, Objective, Result).	5	4.17
Substitution, Combination and Splitting operations are sufficient in order to adapt practices.	3	3.17
Applying these operations requires considerable effort on my part.	4.5	3.67

Table 7-8: Summary of the pertinence and appropriateness of the practice instance and method enactment boards (8 answers).

Is the definition of...	close to reality?	correct?	complete?	Free comments
Method Enactment Board	6	8	7	1
Practice Instance Board	7	8	8	0

- The *cancelled* state was added to Method Enactment.
- The Method Enactment board was redesigned adding some fields.
- The definition of *Family of practices* term was included.

The main drawbacks mentioned by the practitioners or observed by the KUALI-BEH team were:

- The level of detail of the expressed way of workings varies among organizations, mainly among tasks and activity concepts.
- The definition of verification criteria and measures are mainly considered to be part of the project managers' responsibilities, and giving the control of these variables to practitioners was new and unusual for them.
- The need for a tool or set of tools that implements the templates and the boards during enactment is mandatory.

At the end of the workshop, benefits were identified and obtained by both parties. On the one hand, during interviews the software industry participants identified such benefits as:

- *"We organize our knowledge better though practices and methods".*
- *"Now we can transmit our knowledge to other teammates and apply it in the organization".*
- *"Using this structure, it is easy to effectively train new people in my organization".*
- *"I find this approach with which to document my actual way of working attractive. That is to say, I document what I actually do and not what I am supposed to do".*

7.6 Conclusions

We developed a collaborative workshop to which organizations and active software engineers were invited. During the workshop we validated the first version of KUALI-BEH and we had the chance to try out elements of KUALI-BEH one by one.

Received feedback was a very valuable input in order to improve KUALI-BEH and bridge the gap between theory and practice. Besides we should mention an active contribution and involvement of the participants which helped to establish a confident environment to share opinions and criticize KUALI-BEH.

This collaborative workshop was the closure to the second engineering cycle and the obtained suggestions served as a first step towards developing an improved version of KUALI-BEH.

Chapter 8

Case Studies

"Cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet."
"I have a truly marvelous demonstration of this proposition which this margin is too narrow to contain."

Pierre Fermat. Lawyer and mathematician (1601 – 1665).

In order to prove the usefulness and sufficiency of the framework and its common concepts, KUALI-BEH was validated through three case studies. This chapter describes in detail the case studies and their respective lessons learned.

8.1 General considerations

The research question of the case studies was defined as follows:

Is KUALI-BEH suitable for defining practitioners' ways of working carried out during software projects?

In addition, two more questions resulted relevant for the research:

- Is the effort of applying KUALI-BEH suitable to carry out an authoring project?

- What is the value obtained by the organization after having defined its own practices and method?

Based on these questions, the objectives that are common for the three case studies were defined as follows:

O1. To demonstrate sufficiency of the KUALI-BEH elements to describe practitioners' ways of working.

O2. To measure feasibility of using the concept of Practice to express practitioners' tacit practices.

O3. To identify value obtained by the organization as a consequence of defining its own method.

To decide if the objective was achieved or not the following indicators were collected:

- The indicator associated to O1 was collected through two surveys applied to practitioners.
- The indicator associated to O2 was obtained by measuring the effort required by the practitioners to document their ways of working.
- The indicator associated to O3 was obtained through an interview during the Feedback step, in which practitioners expressed the benefits and drawbacks derived from structuring their ways of working through KUALI-BEH.

In the next subsections each of the case studies is described in more detail.

8.2 Case Study 1: InfoBLOCK

The first case study was developed in InfoBLOCK[1], a Mexican organization with offices in Mexico City and Guadalajara. InfoBLOCK started in 1997 and focuses on design, development and manufacture of equipment and programs for workforce management (attendance and schedules control, task planning, employees' checkpoints management and reports).

In this case study participated two InfoBLOCK's directors and two programmers. The programmers had between 1 and 3 years as active practitioners and reported having developed Analyst, Tester, Project Manager and Programmer roles.

[1] `http://www.infoblock.com.mx/`

8.2.1 Background

At the moment of the beginning of the case study, the organization did not have a defined development method nor did it follow a reference model or standard. In consequence the needs expressed by the organization were:

- To define the actual software development process followed in the organization.
- To be aware of what is being done by the work team at a particular moment during the development process.

The artifact to be validated through this case study was the second version of KUALI-BEH, which was baselined on August 13th, 2012. This version was the output of the second engineering cycle, which involved the Collaborative Workshop presented in chapter 7.

8.2.2 Design and Execution

The case study was designed as a sequence of seven steps, the execution of each of them are described as follows:

1. **Presentation of the case study.**

   The objectives of the case study were presented by the researchers' team to InfoBLOCK's team and KUALI-BEH and its elements were explained. This step took 90 minutes.

2. **Description of practitioners' way of working.**

   This step was carried out in one work session when the researcher guided practitioners in the documentation of their first practice using the Practice template. It also served to gain an insight into the organization and identify individual goals and daily responsibilities of the practitioners involved in the case study.

   After ensuring the understanding of how to describe their ways of working through the concept of practice, the researcher assigned the practitioners activities that they should perform independently during the next step. This step took 120 minutes.

3. **Documentation of practices.**

InfoBLOCK's team began the documentation of practices and the running of an internal project simultaneously. Throughout this phase the communication between the team and the researcher was carried out by videoconferencing and email.

The team completed the following tasks:

(a) The practitioners documented their way to working using the Practice template. This had been done before they would execute the practice in the internal project.

(b) Then the researcher checked aspects of consistency between the KUALI-BEH concepts and what the team interpreted.

(c) Later, the researcher suggested improvements in the use of concepts but not in the way of working itself.

(d) Finally, the practitioners discussed the suggestions and then generated the 1.0 version of the practice.

Figure 8-1 shows a practice defined by InfoBLOCK practitioners.

4. **Implementation of documented practices.**

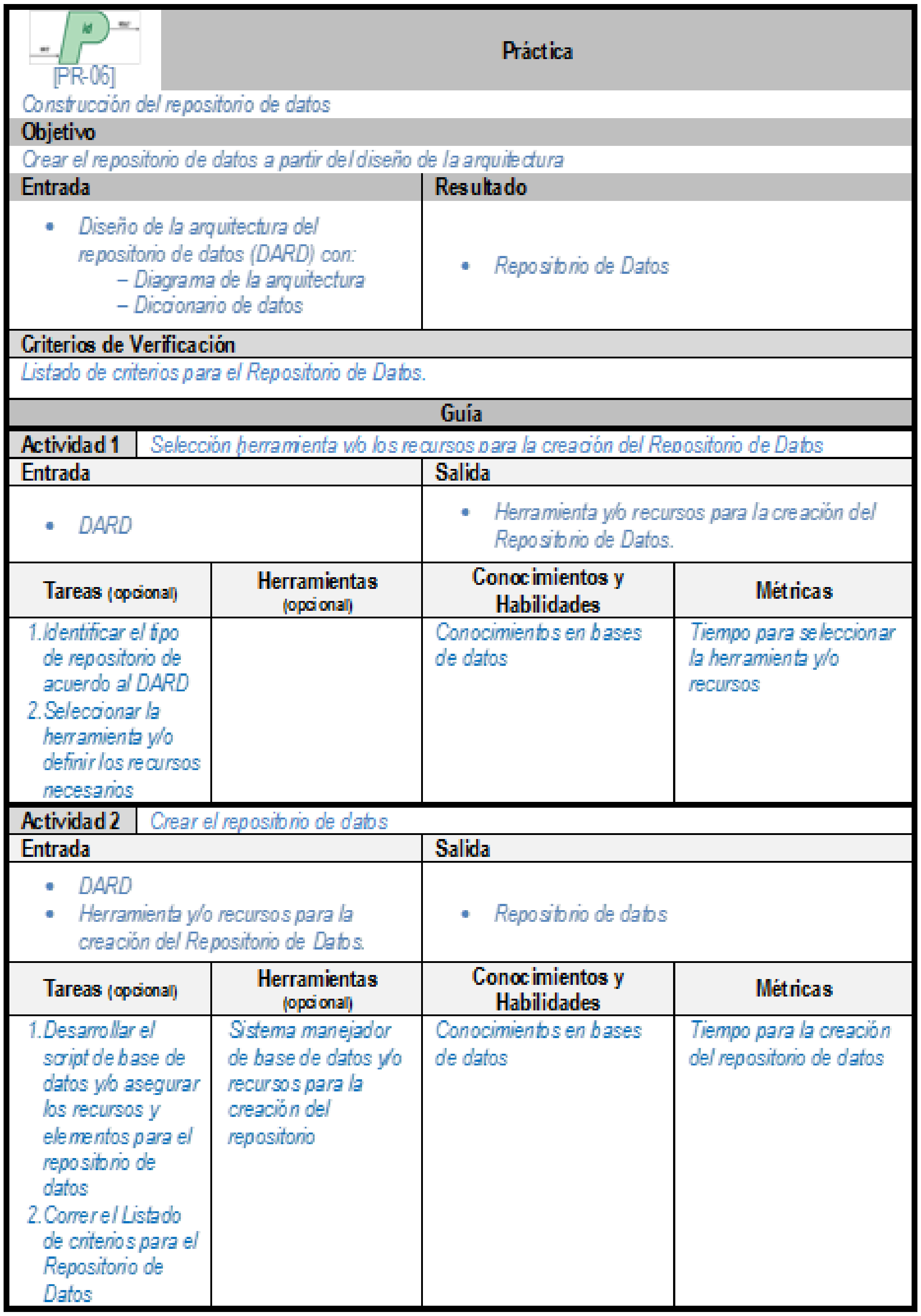
Due to the fact that the case study was developed alongside a real project, the practitioners performed the practices almost immediately after documenting them. This resulted in adjustments to the initial version of the practices, which were carried out following the same tasks developed in the previous step.

During the three weeks of the project, 13 practices were identified, documented, implemented and adjusted by practitioners.

5. **Composition of the method.**

After the practices were documented, executed and adapted, the practitioners proceeded to composing a method, that represents their actual way of working using the 13 created practices.

Then the practitioners identified and integrated their software development method using a composition operation. Later the resulting method was modified to achieve properties



[PR-06]	**Práctica**
Construcción del repositorio de datos	
Objetivo	
Crear el repositorio de datos a partir del diseño de la arquitectura	
Entrada	**Resultado**
• *Diseño de la arquitectura del repositorio de datos (DARD) con:* – *Diagrama de la arquitectura* – *Diccionario de datos*	• *Repositorio de Datos*
Criterios de Verificación	
Listado de criterios para el Repositorio de Datos.	
Guía	

Actividad 1	*Selección herramienta y/o los recursos para la creación del Repositorio de Datos*
Entrada	**Salida**
• *DARD*	• *Herramienta y/o recursos para la creación del Repositorio de Datos.*

Tareas (opcional)	**Herramientas** (opcional)	**Conocimientos y Habilidades**	**Métricas**
1. Identificar el tipo de repositorio de acuerdo al DARD *2. Seleccionar la herramienta y/o definir los recursos necesarios*		*Conocimientos en bases de datos*	*Tiempo para seleccionar la herramienta y/o recursos*

Actividad 2	*Crear el repositorio de datos*
Entrada	**Salida**
• *DARD* • *Herramienta y/o recursos para la creación del Repositorio de Datos.*	• *Repositorio de datos*

Tareas (opcional)	**Herramientas** (opcional)	**Conocimientos y Habilidades**	**Métricas**
1. Desarrollar el script de base de datos y/o asegurar los recursos y elementos para el repositorio de datos *2. Correr el Listado de criterios para el Repositorio de Datos*	*Sistema manejador de base de datos y/o recursos para la creación del repositorio*	*Conocimientos en bases de datos*	*Tiempo para la creación del repositorio de datos*

Figure 8-1: Practice created by practitioners (in Spanish).

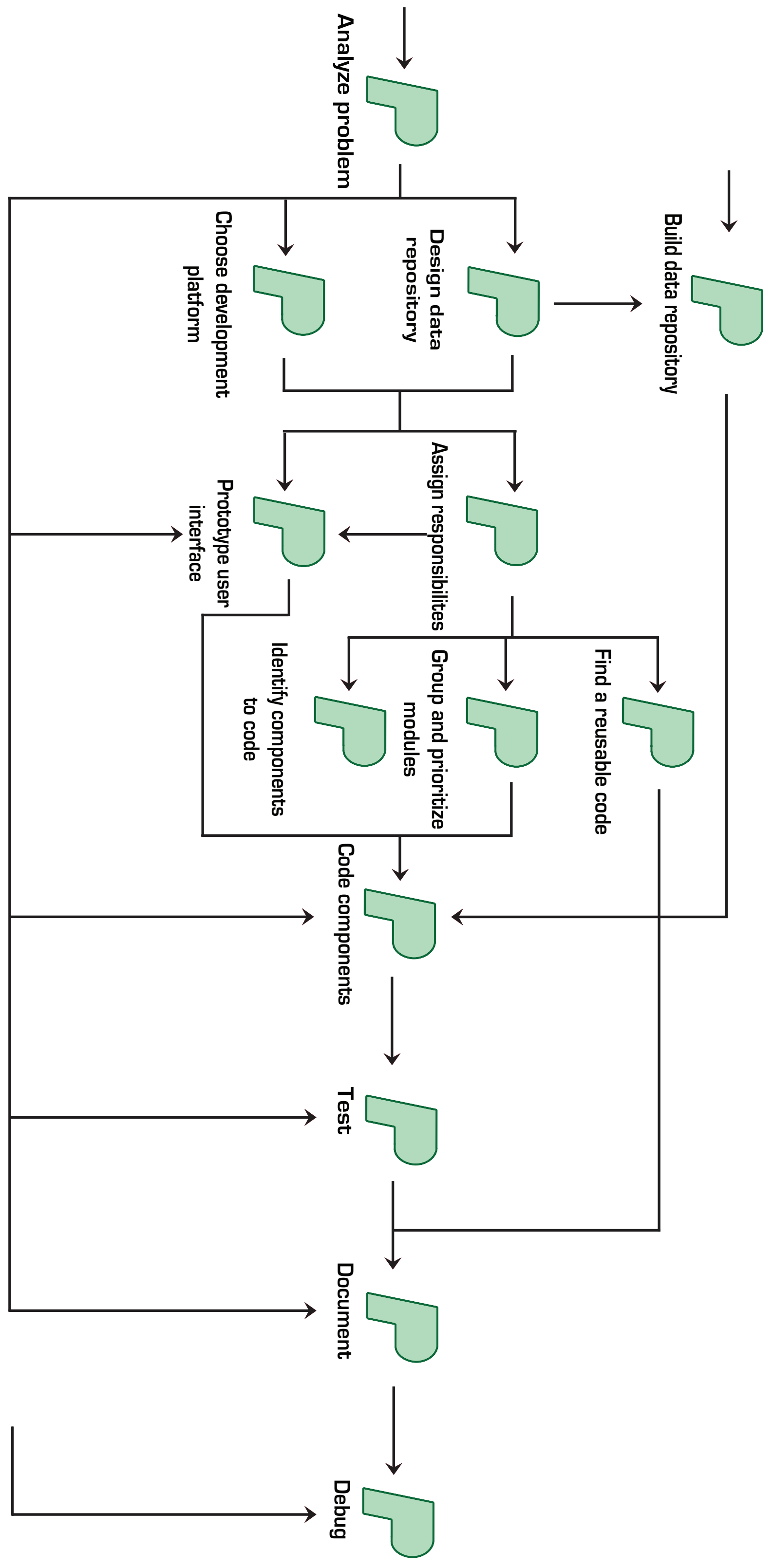


Figure 8-2: Method created by practitioners.

of coherency, consistency and sufficiency, thus attaining a well formed method, shown in Figure 8-2.

The practitioners used the method template and the graphical representation of KUALI-BEH to complete this step.

6. **Presentation of results and Feedback interview.**

   This step consisted in a session where the documented practices and the composed method were presented to the organization's members.

   During this meeting the researchers collected and registered orally-expressed lessons learned of this particular case study, such as suggestions of improvement, benefits and drawbacks of KUALI-BEH.

   Besides, the InfoBLOCK's team had the opportunity to think about their actual way of working and the improvements to be done in future projects.

7. **Closure.**

   At the phase of closure, final generated products and the case study report were delivered to the InfoBLOCK's team.

8.2.3 Analysis

At the end of this case study, a method consisting of 13 practices that were defined with KUALI-BEH was documented. According to the initial objectives of this case study, the following results were obtained:

O1. To demonstrate sufficiency of the KUALI-BEH elements to describe practitioners' ways of working.

Based on the data collected through surveys and practitioners opinions, it was concluded that the elements that conform KUALI-BEH were sufficient to describe the way of working carried out by InfoBLOCK practitioners during the selected real project.

Moreover, we could observe that the common concepts were everyday concepts for practitioners, who used them in a natural and straight-forward manner.

O2. To measure feasibility of using the concept of Practice to express practitioners' tacit practices.

Considering the effort and time required by practitioners to understand KUALI-BEH, we can conclude that making practitioners ways of working explicit in a short period of time was feasible. Except for the first practice, which required the researcher's support in documenting it, 12 more practices were expressed by practitioners themselves without the need to rethink the common concepts.

The total required effort were 10 hours, resulting in an average of less than 50 minutes per practice.

Also it was observed that a practice contained between 3 and 4 activities in average. Finally, documenting their way of working allowed practitioners to identify work products that were generated during the project. A total of 28 work products were identified and grouped into 9 categories.

O3. To identify value obtained by the organization as a consequence of defining its own method.

During the feedback meeting, InfoBLOCK's members expressed several beneficial aspects to the organization, highlighting the fact that they formalized their way of working. The points mentioned by them were:

- *"As a first result now there is clarity on how practitioners work, and they defined it by themselves."*
- *"Having the method defined, now it is possible to distribute it, among other members of the organization, and ask for improvements and unification of the working way".*
- *"With this (the method), that actually belongs to the organization, we will be able to teach and train in-house our new entrants".*
- *"KUALI- BEH brings order to disorder".*
- *"These (the practices) are guidelines that must be implemented in the organization".*
- *"The value of the case study rests with the practitioners themselves".*

- *"This allow us to reassess various aspects and things that are important for practitioners".*
- *"Using three levels, method – practice – activity, it was sufficient to express and control the work done in the project".*

8.2.4 Results

Two practitioners used the bottom-up approach to express their way of working in order to define the organization's software development method. The framework was presented to the practitioners in a two-hour session, at the end of which the first practice was identified and expressed. Followed by 12 more practices and after applying modification operations to some of them, they were agreed as practitioners' actual way of working.

The most important improvement to KUALI-BEH proposal was the adjustment of the practice template in order to make it more legible, the changes were centered on the visual organization of the Guide section.

The results of the case study, expressed by the organization's practitioners and heads during a feedback interview, were:

- *"KUALI-BEH is easy to understand and apply by practitioners"* (four hours training and 6 hours work to define a 13-practice method in a four week project).
- *"Now we have explicit documentation of what we actually do, (which) means value for us (organization managers); these practices can be used to train new employees".*
- *"For us, KUALI-BEH (common) concepts are sufficient to express and model our different ways of working as methods composed by practices".*

Table 8-1 shows the summary of this case study.

8.3 Case Study 2: Entia

A second case study was held at an organization in charge of software projects design called Entia[2]. Entia is a Mexican IT organization established in 2003 as a software developer orga-

[2]`http://www.entia.com.mx/`

Table 8-1: Summary of case study 1.

	InfoBLOCK
Size of project team	2 programmers 2 directors
Previously adopted method or process	No
Project duration	3 weeks
Organization core business process	Software development
Number of expressed practices	13
Total authoring effort (in hours)	10

nization conformed at the beginning by 4 employees, now 20. Entia has executed projects to construct custom software related to finance, retail, manufacturing and service sectors.

In order to increase the projects' success rate and, consequently, the number of projects, Entia developed Innocamp strategy. Innocamp consisted in an intense workshop where all stakeholders collaboratively designed their new custom software development project. This envisioning strategy was created in order to avoid continual coming and going between customers and Entia work team, caused by the lack of definition, clarity and consensus in their needs.

Since 2007 Innocamp has evolved into ActiveAction workshop. The evolution started with the inclusion of games as a way of entertaining the stakeholders during the workshop. Over the time the work team noticed that games helped to decrease resistance and increase concentration level of the stakeholders, causing an important impact on the results of each workshop.

In this case study a director and a coach from Entia and two assistants from the researchers team took part.

8.3.1 Background

The rapid pace of evolution from Innocamp to ActiveAction has brought as a consequence the lack of documentation related to the workshop, while all the knowledge and techniques belonged to the people in charge and not to the organization.

For that reason, the objective defined by the Entia director was:

- To document each step of the workshop activities.

The artifact to be validated through this case study was the third version of KUALI-BEH, which was baselined on November 12th, 2012. This version was the output of the third engineering cycle, which involved case study 1.

8.3.2 Design and Execution

This case study was divided into the following six steps:

1. **On-site observation of the workshop.**

   In order to understand the novel game-based technique, its purpose and how it was carried out, an on-site observation was conducted. The team involved in this step was the director, the coach and one assistant. The assistant was in charge of observing the workshop guided by the coach with participation of real clients, while the director was the game specialist.

   It is important to mention that ActiveAction workshop takes between 10 and 12 hours, for that reason this step took 720 minutes.

2. **Presentation of the case study.**

   The objectives of the case study were presented by the researchers' team to the Entia's team. Subsequently KUALI-BEH and its elements were explained and this step took 60 minutes.

3. **Description of practitioners' way of working.**

   This step was carried out in one work session, during which the researcher guided the director and assistants through the documentation of their first practice using the Practice template.

   After ensuring the understanding of how to describe the workshop through the concept of practice, the researcher assigned to assistants the activities that they should perform during the next step. This step took 60 minutes.

4. **Documentation and adaptation of practices.**

   During the phase of practices documentation the assistants identified a total of 19 practices involved in the workshop. The assistants together with the director expressed, agreed and composed these practices as a method.

The team completed the following tasks:

(a) The assistants documented the workshop activities using the Practice template.

(b) Then the researcher checked aspects of consistency between the KUALI-BEH concepts and what the assistants interpreted.

(c) Later, the assistants presented the practices to the director, who suggested modifications.

(d) Finally, the assistants applied modifications and generated the 1.0 version of the practice.

The challenge of this step was to identify the inputs and results of each practice, since Entia used to manage all of them as one work product: a mind map. During the three months of the project, 19 practices were expressed by the assistants and were agreed upon by the game expert.

5. **Composition of the method.**

   Finally the method was integrated and studied to provide scientific and theoretical backgrounds in order to explain the benefits of adding games to this kind of practices. The 19-practices method is shown in Figure 8-3.

6. **Presentation of results.**

   This step consisted in a session where the documented practices and the composed method were presented to the Entia director. The main result of this case study was the "unusual" method and its practices, which was presented as a research paper (Morales-Trujillo et al., 2014) during the 9th International Conference on Evaluation of Novel Approaches to Software Engineering 2014 (ENASE'14) that took place in Lisbon, Portugal.

8.3.3 Analysis

At the end of this case study a method composed of 19 practices using the template Practice KUALI-BEH was documented. According to the initial objectives the following results were obtained:

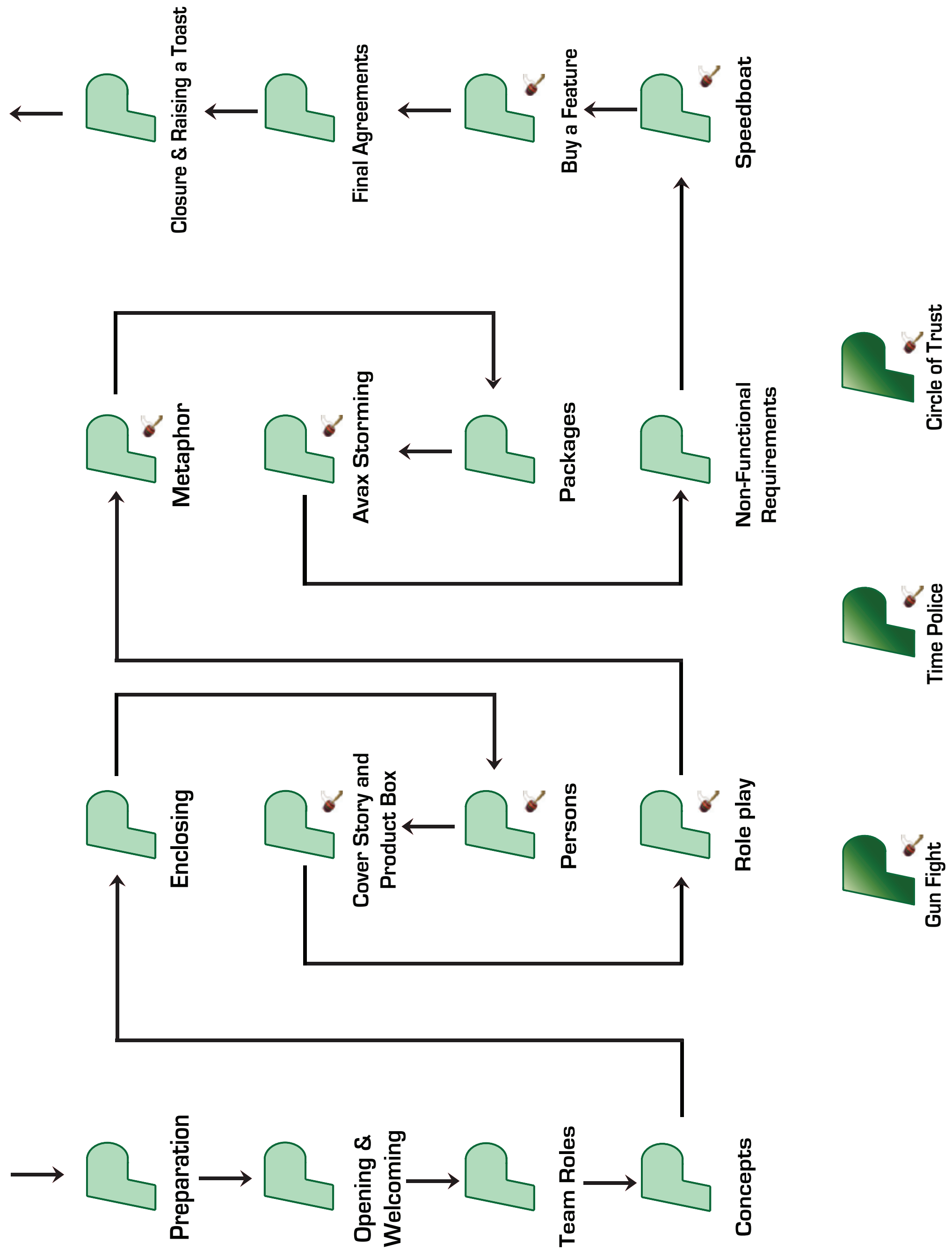


Figure 8-3: ActiveAction method diagram (Morales-Trujillo et al., 2014).

O1. To demonstrate sufficiency of the KUALI-BEH elements to describe practitioners' ways of working.

Based on the data collected through the surveys and the assistants opinions, it was concluded that the elements that conform KUALI-BEH were sufficient to describe the game-based method.

O2. To measure feasibility of using the concept of Practice to express practitioners' tacit practices.

In this case of study the required effort was 26 hours, resulting in an average of 82 minutes per practice. It was concluded that expressing the workshop practices in short period of time was feasible.

O3. To identify value obtained by the organization as a consequence of defining its own method.

The only objective defined by the Entia's director was to document each step of the workshop activities and it was fully achieved. Besides documenting the method reduced the possibility of variation and allowed Entia to identify ways to improve it and, more importantly, it allowed replication of the workshop in affiliates.

Last but not the least, by publishing the paper at ENASE'14 Entia became more visible.

8.3.4 Results

On the one hand, this case study gave Entia many benefits and allowed it to achieve important business goals in order to remain competitive and visible.

On the other hand, KUALI-BEH proposal was improved and its usefulness in a context where software development was not the main purpose was demonstrated.

Table 8-2 shows the summary of this case study.

8.4 Case Study 3: Tic-Tac

The third case study was developed at San Luis Potosi Tech Institute (ITSSLP) in a software development entity called Tic-Tac. This entity was under the Tech business incubator program.

Table 8-2: Summary of case study 2.

	Entia
Size of project team	1 director 1 coach 2 assistants
Previously adopted method or process	No
Project duration	3 months
Organization core business process	Software projects design
Number of expressed practices	19
Total authoring effort (in hours)	26

In this case study participated two professors and two recently graduated system engineers.

8.4.1 Background

The software development entity was carrying out projects without a defined method or process to follow. For that reason the objectives defined by professors in charge of coordinating the entity were:

- To document their software development process.
- To train new work team members using the defined process.

The artifact to be validated through this case study was the KUALI-BEH Authoring Extension, which was baselined at November 12th, 2012.

8.4.2 Design and Execution

In order to satisfy the needs defined by the ITSSLP team, this case study added an objective to the previously three defined in section 8.1. The new objective was established as follows:

O4. To measure the effort required to train a new work team member using the defined method.

The indicator that will demonstrate if the objective was achieved or not was collected in the following manner:

- The indicator associated to O4 was obtained by measuring the effort required by the practitioners to train a new work team member.

1. **Description of practitioners' way of working.**

   At this phase the researcher led a work session, during which a first practice was documented together with the ITSSLP team. It also served to gain insight into the organization and individual goals and responsibilities of those involved in the software development entity.

   After ensuring the understanding of how to describe their ways of working, the team performed next activities by itself.

2. **Documentation of practices.**

   The ITSSLP team started to document its way of working. Throughout this phase the communication between the team and the researcher was by email and videoconferences.

   The team completed the following tasks:

   (a) The practitioners documented their way to working using the Practice template. This had been done before they would execute the practice in the internal project.

   (b) Then the researcher checked aspects of consistency between the KUALI-BEH concepts and what the team interpreted.

   (c) Later, the researcher suggested improvements in the use of concepts but not in the way of working itself.

   (d) Finally, the practitioners discussed the suggestions and then generated the 1.0 version of the practice.

   This step resulted in 23 documented practices.

3. **Composition of the method.**

   After the practices were documented, executed and adapted, a method that represents the actual way of working was composed using the 23 created practices. The resulting method was then modified to achieve properties of coherency, consistency and sufficiency, thus attaining a well formed method, shown in Figure 8-4.

   It can be noted that differently to the other case studies, the ITSSLP proposed a division in phases for their method.

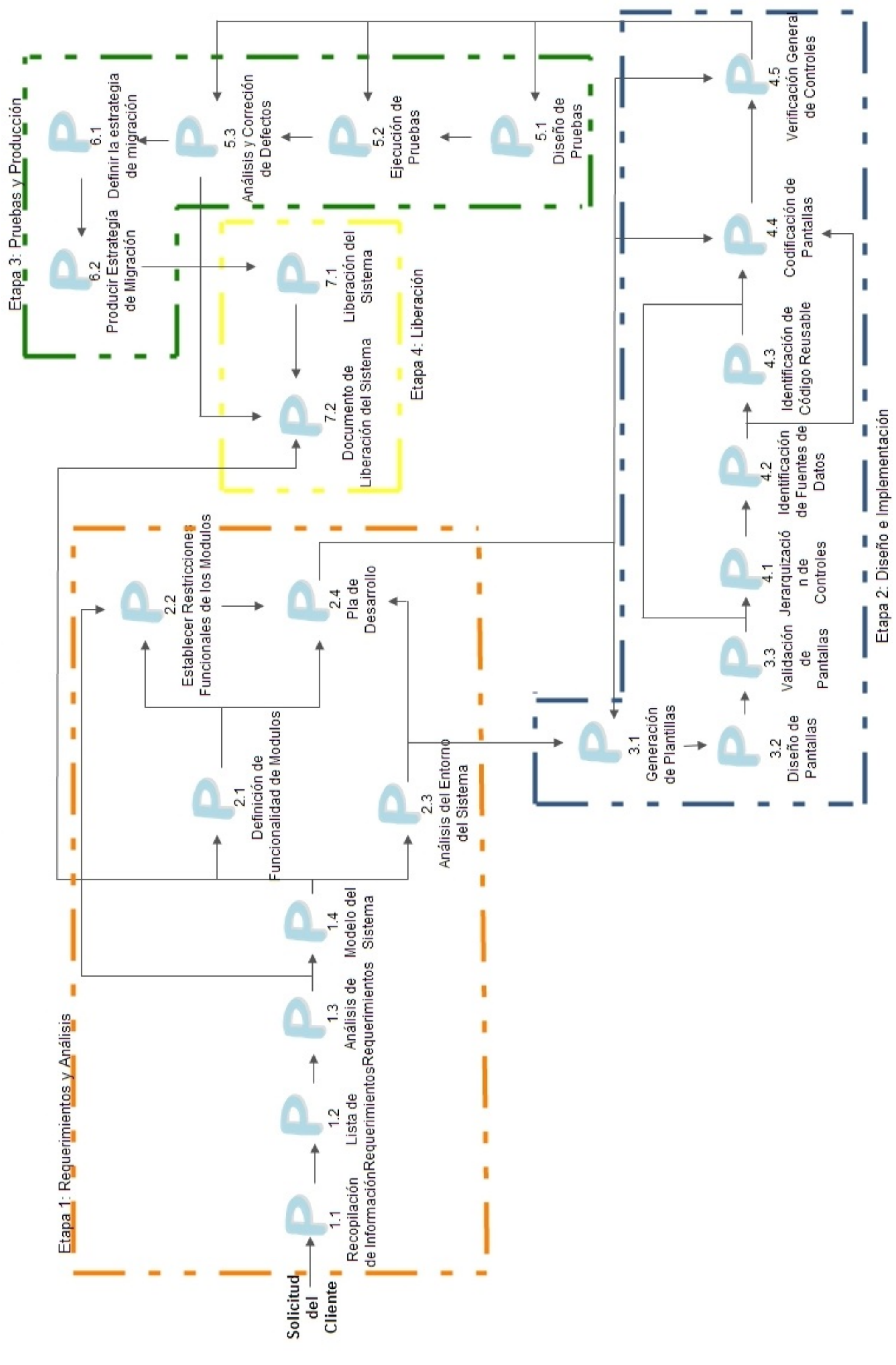

Solicitud del Cliente
Etapa 1: Requerimientos y Análisis
1.1 Recopilación de Información
1.2 Lista de Requerimientos
1.3 Análisis de Requerimientos
1.4 Modelo del Sistema
2.1 Definición de Funcionalidad de Modulos
2.2 Establecer Restricciones Funcionales de los Modulos
2.3 Análisis del Entorno del Sistema
2.4 Pla de Desarrollo
Etapa 2: Diseño e Implementación
3.1 Generación de Plantillas
3.2 Diseño de Pantallas
3.3 Validación de Pantallas
4.1 Jerarquizació n de Controles
4.2 Identificación de Fuentes de Datos
4.3 Identificación de Código Reusable
4.4 Codificación de Pantallas
4.5 Verificación General de Controles
Etapa 3: Pruebas y Producción
5.1 Diseño de Pruebas
5.2 Ejecución de Pruebas
5.3 Análisis y Correción de Defectos
6.1 Definir la estrategia de migración
6.2 Producir Estrategia de Migración
Etapa 4: Liberación
7.1 Liberación del Sistema
7.2 Documento de Liberación del Sistema

Figure 8-4: Method diagram created by practitioners.

4. **Adaptation of the method.**

   In order improve and adjust the method, some adaptation operations were applied. ITSSLP team adjusted its method in the following manner:

   - Two practices were concatenated due to the fact that a work product was contained in the other.
   - Two practices were combined in order to facilitate its execution.
   - The method passed from 4 to 7 phases. This change was done in order to make simpler the understanding of the method during training tasks and also to facilitate the work distribution during projects.

   At this step two of the four adaptation operations were applied with beneficial results.

5. **Presentation of results and Feedback interview.**

   During this step the documented practices and the composed method were presented to the ITSSLP authorities.

   In a videoconference the researchers collected and registered orally-expressed lessons learned of this particular case study, such as suggestions of improvement, benefits and drawbacks of KUALI-BEH.

   Another result of this case study was the lessons learned by Tic-Tac, which were reported as a research paper (Arroyo-López et al., 2015) during the International Conference on Software Engineering Innovation and Research 2015 (CONISOFT'15) that took place in San Luis Potosí, Mexico.

6. **Closure.**

   The generated products and the report of the case study were delivered to the ITSSLP's team.

8.4.3 Analysis

At the end of this case study a method that represented the way of working of the ITSSLP software development entity was composed. The 23 authored practices and a list of required work products was also generated. According to the objectives of this case study the following results were obtained:

O1. To demonstrate sufficiency of the KUALI-BEH elements to describe practitioners' ways of working.

The sufficiency of KUALI-BEH elements in order to describe the way of working of practitioners was confirmed. The results collected by the surveys were:

- The elements of a practice are enough to describe practitioners' way of working (15 of 15 points).
- It was possible and appropriate for practitioners to express their tacit practices using the concept of Practice (13 of 15 points).
- The properties defined for methods are understandable and coincide with the features a practitioner would ask for a well-defined method (15 of 15 points).

O2. To measure feasibility of using the concept of Practice to express practitioners' tacit practices.

In this case of study the required effort was 44 hours, resulting in an average of 115 minutes per practice. The professors in charge of the entity expressed satisfaction about the invested effort.

O3. To identify value obtained by the organization as a consequence of defining its own method.

A software development method was documented and served as basis for the execution of the new software projects carried out by the entity. At the moment, two new projects were developed successfully and two new members were trained and integrated to the team.

O4. To measure the effort required to train a new work team member using the defined method.

The ITSSLP followed a seven-step strategy in order to incorporate two new members and the time investment was 7 hours. According to the professors the process' success was related to the participants' knowledge and skills primarily.

8.4.4 Results

This case study experience allowed us to improve KUALI-BEH and its Authoring Extension, precisely it permitted us to define the states and transitions of the Practice Authoring and Method Authoring Alphas in a better way.

Besides we confirmed that two of the adaptation operations are suitable and has real meaning for practitioners, however, verifying their usefulness in other projects is still necessary.

The results of the case study, expressed by the work team and professors during the feedback interview, were:

- *"We were able to organize our way of working".*
- *"It allows us to complete a project properly, it was an important improvement when the team distributed tasks".*
- *"Also we think, that KUALI-BEH guides the practitioner and causes his/her better participation in a project".*
- *"In addition, it will serve to generate a repository of available methods and practices for different projects".*
- *"Team members can think and rethink how they do things".*
- *"We observed that working with this methodology requires a culture of collaborative work".*

Table 8-3 shows the summary of case study.

Table 8-3: Summary of case study 3.

	Tic-Tac
Size of project team	2 professors 2 developers
Previously adopted method or process	No
Project duration	6 months
Organization core business process	Software development
Number of expressed practices	23
Total authoring effort (in hours)	44

8.5 Threats to validity and limitations of the case studies

In order to avoid threats to validity, various factor were considered in the three case studies.

- **Construct validity:** Multiple sources of data were used in order to provide evidence and respond the research question. Interviews, direct observations, surveys and work artifacts were the sources used, covering the three degrees defined by (Runeson et al., 2012).

  Moreover, the validity of the construct (or artifact) is assured since it was created following the mandatory requirements requested by OMG in the FACESEM RFP. Also it was validated by the OMG evaluation team.

- **Internal validity:** The case studies' results demonstrated that the objective for which KUALI-BEH was created was achieved, allowing us to accomplish our goals and the participants' needs.

  Different causal relations were examined:

  - About the surveys' trustworthiness. The data collected through surveys is closely related to the practitioners' experience. In the case studies the participants' experience covered the "juniors" through "seniors" classification.
  - About the number of participants. Although the number of participants is not large, we can say that the sample is representative of a practitioners' profile working in small software organizations.
  - About the participants' age, education and experience. These factors could affect whether they were in favor or against KUALI-BEH, however the sample of participants had diversity.

  After having analyzed these factors, all of them were disallowed, and as a result we have been able to determine that the implementation of KUALI-BEH in the case studies allowed us to achieve the case studies' objectives.

- **External validity:** In spite of the limited number of participating organizations in the case studies, we can state that each organization can be categorized as a typical software developer entity. The three participating organizations shared the main characteristics of very small entities in charge of a software development endeavors, which is the target audience of KUALI-BEH proposal.

The selection of the participants was not intentional, the organizations that participated in the case studies expressed their interest to participate by themselves.

It is important to highlight that the methods that were expressed using KUALI-BEH were actual way of workings of active practitioners involved in real projects.

In consequence, we believe that the sample and the context of this research were representative. Also, thanks to the OMG and SEMAT scopes, the obtained results were contrasted with researchers and practitioners from other countries who backed it up.

Therefore, we conclude that the obtained results provide a possibility to generalize about the subject under research.

- **Reliability:** The case studies were carried out by two researchers and the results were constantly triangulated to third parties, such as colleagues and members of the research group. In addition, the results and lessons learned were informed and disseminated extensively in each case study report delivered to participants.

  Also there is a detailed plan that served as guidance for the case studies available for any future replications.

In order to improve the validity of the case studies, it was compared against the checklists provided by (Runeson et al., 2012).

In addition, the following approaches were taken into account:

- The *prolonged involvement*, mainly in CS2 and CS3 (3 and 6 months respectively), which allowed us to develop a relationship of trust with participants, making the collection of data an unobstructed process and, in general, to develop the case studies in a pleasant atmosphere.

- During CS2 we had two assistants that participated in the data collection process, allowing us to analyze different data sources, such as interviews, surveys and direct observations. This circumstance made *triangulation* possible.

- The three case studies counted with *peer debriefing*, due the fact that the case studies were carried out by two researchers. In addition, findings and results were discussed periodically with other members of the research group.
- The work products and documents generated during each case study were given to the participants so they could review them. This happened, at least, at the end of each step of the study, but if needed, the member checking action was done at any moment.
- A version control strategy was defined since the very beginning of each case study. The strategy was supported by a software tool and, given the technological background of all the participants, the *audit trail* mechanism was easy to follow and its application was very successful.

Finally, limitations of the case studies can be summarized in two points:

- The samples size is small, therefore it limits the power of generalization. It is necessary to replicate case studies in contexts with bigger populations.
- Bias in the case studies could take place and be related with the participants' feeling of being observed and evaluated. It can derive in an alteration of their actual way of working.

8.6 Conclusions

Conducting case studies brought about many lessons learned, which allowed us to improve KUALI-BEH and its validation process. The main lessons learned from these three case studies concern the following issues:

A tool is required to support the practice authoring process. The practice authoring process of each case study was carried out using a word processor. We can say that at the beginning this resulted handy, mainly during the steps that involved reviewing of authored practices by the researcher. Using the *Track changes* function of the word processor, the come-and-go of authored practices between practitioners and researchers became a valuable learning process.

However, to manage the control version strategy of each practice, to make adaptations and to share practices became a harder process. At this point the need of a tool that automatizes this process became essential.

Support from directors and commitment from practitioners is necessary. The validation process is a process that involves many variables to be controlled. On the one hand, it requires the organization's interest in the proposal and understanding of how it will attend to its particular needs. On the other hand, the people in charge of applying the proposal must see benefits reflected in their daily work in order to avoid bias and reluctance. So a two-level support from directors and from practitioners is mandatory and, if one of these levels is not committed, the case study will not be possible.

Fortunately, in our experience, the three case studies had support from directors and their enthusiasm was transmitted to practitioners who took it seriously and got engaged since the very beginning.

To find suitable projects in order to validate the proposal. It is clear that support and enthusiasm are not the only ingredients needed to carry out a case study. The proposal needs to be validated in real context and in a suitable project. While the researchers' team is almost always available and open to conduct case studies, the organization needs to previously estimate time, effort and risks of being involved, which can caused time matching problems between the researcher and the organization's project.

To identify improvements and adjustments to artifact. The case studies allow us to improve KUALI-BEH in many aspects. We can highlight the following improvements and adjustments:

- The improvement of the Practice template, mainly the Guide section.
- The inclusion of the *Method Diagram* section to Method template.
- The adjustment of two of the adaptation operations.

Finally, we can conclude that the combination of TAR and Case Study research methods was a successful experience, allowing us to validate and improve KUALI-BEH in several ways and making the process a valuable experience (Morales-Trujillo et al., 2015b). Also we can established that the effort invested, see Table 8-4, permitted a high ROI to both parties: practitioners and researchers.

Table 8-4: Effort by step in each case study.

	CS1	CS2	CS3
On-site observation	-	720	-
Presentation of the case study	90	60	60
Description of practitioners way of working	120	60	60
Documentation of practices		600	
Implementation of documented practices	390	-	2520
Composition of the method		120	
Training new work team members	-	-	420
Presentation of Results and Feedback	60	60	60
Closure	30	30	30
Total effort of case study (in hours)	**11.5**	**27.5**	**52.5**
Practices authored	13	19	23
Total effort of authoring (in hours)	**10**	**26**	**44**
Average authoring effort per practice	46.2	82.1	114.8

Chapter 9

OMG Experience

"If I see further it is by standing on the shoulders of giants."

Sir Isaac Newton. Physicist and mathematician (1642 – 1727).

This section presents the roadmap followed throughout the OMG standardization process, from KUALI-BEH's origins to its fusion with the ESSENCE proposal and its eventual appearance as a useful and applicable standard.

9.1 Introduction

The OMG is an IT consortium that was established in 1989. The OMG is formed of organizations such as Microsoft, Boeing, Oracle, Ericsson and NASA, and is in charge of developing IT standards such as UML, XMI, CORBA or BPMN. The OMG members form working groups called Task Forces, which are in charge of transforming initiatives into technology specifications.

In 2009 SEMAT emerged with the purpose of generating a theoretical basis for the discipline. Several influential members of the software engineering community participate in SEMAT, some of whom are involved in organizations that are OMG members. SEMAT determined the need to define fundamental concepts for practices, identify specific theories backed up by examples of their successful application, elucidate a set of universals and a kernel language to describe them, and establish a set of metrics to assess software practices, products and people (Jacobson et al., 2010).

The OMG endorsed SEMAT and made it a standard project: A Foundation for the Agile Creation and Enactment of Software Engineering Methods RFP (FACESEM) (OMG, 2011b), which has provided the initiative with greater stability and visibility.

Owing to the fact that SEMAT was closely followed by our research group, as soon as the RFP was published, KUALI-KAANS took it as a promising line of work and responded to the FACESEM RFP with its own proposal named: KUALI-BEH: Software Project Common Concepts (Morales-Trujillo and Oktaba, 2012a).

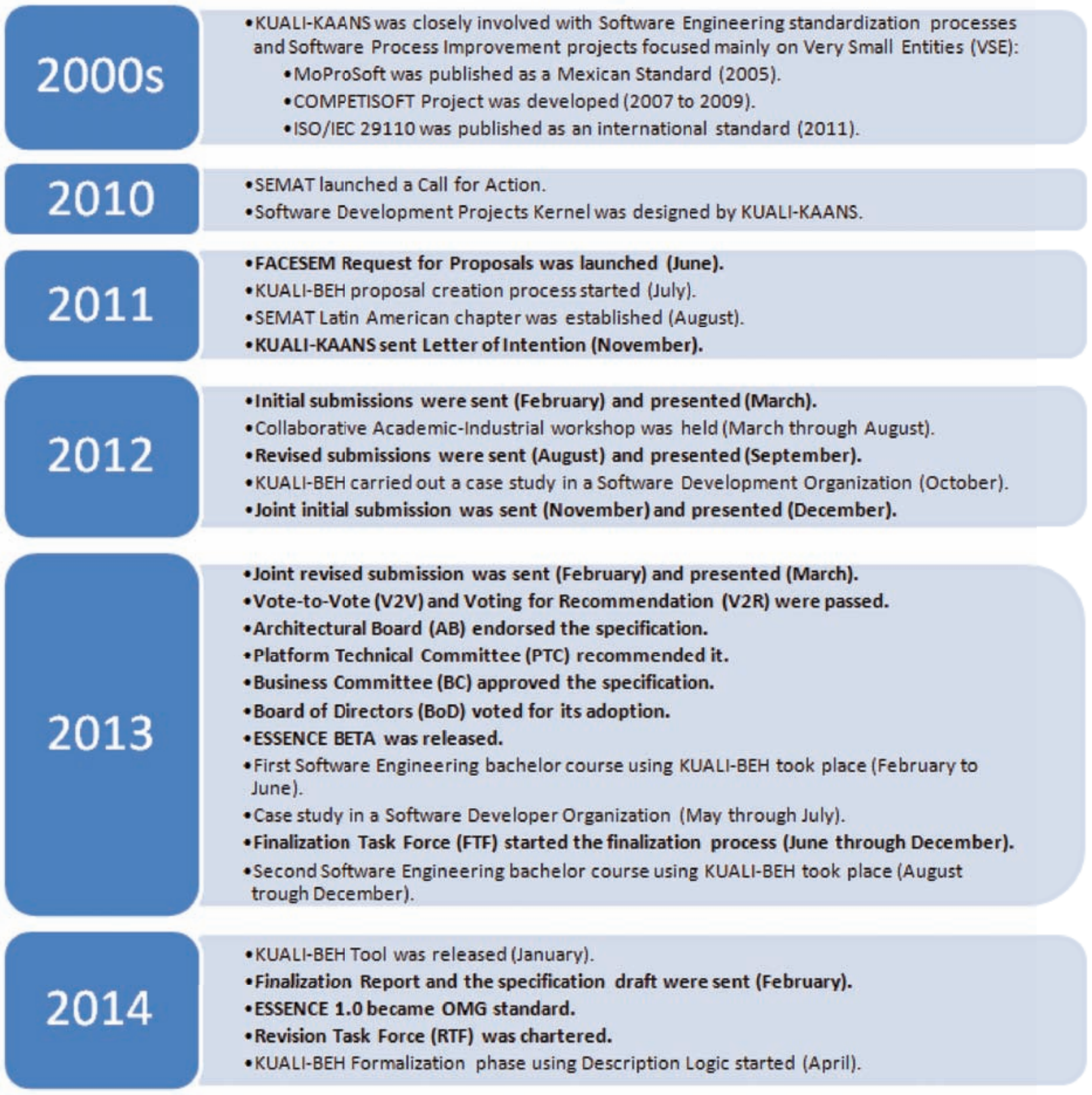
In 2011, the OMG standardization process was started, and two proposals became involved: KUALI-BEH and ESSENCE – Kernel and Language for Software Engineering Methods (OMG, 2013).

The objective of this chapter is to describe the roadmap followed throughout the OMG standardization process, from creating KUALI-BEH, its later fusion with ESSENCE and its eventual (after three years) adoption as a formal OMG standard. KUALI-KAANS background and experience in similar projects, the KUALI-BEH creation process and the specific steps followed during the OMG standardization process (in bold type) are shown in Figure 9-1.

9.2 FACESEM RFP

FACESEM RFP was prepared by the OMG Task Forces between December 2010 and June 2011. On June 24 2011, the OMG Technical Committee voted to issue the RFP and made it publicly available.

The objective of the FACESEM RFP was to obtain a foundation for the agile creation and enactment of software engineering methods by development practitioners themselves (OMG, 2011b). On the one hand, the RFP requested that a kernel of Software Engineering domain concepts and relationships be extensible, flexible and easy to use. On the other hand, it needed a domain-specific modeling language that would allow developers to describe the essentials of their current and future practices and methods (OMG, 2011b). In other words, the RFP requested a framework with which to describe, share and compare current ways of working in Software



2000s

- KUALI-KAANS was closely involved with Software Engineering standardization processes and Software Process Improvement projects focused mainly on Very Small Entities (VSE):
  - MoProSoft was published as a Mexican Standard (2005).
  - COMPETISOFT Project was developed (2007 to 2009).
  - ISO/IEC 29110 was published as an international standard (2011).

2010

- SEMAT launched a Call for Action.
- Software Development Projects Kernel was designed by KUALI-KAANS.

2011

- **FACESEM Request for Proposals was launched (June).**
- KUALI-BEH proposal creation process started (July).
- SEMAT Latin American chapter was established (August).
- **KUALI-KAANS sent Letter of Intention (November).**

2012

- **Initial submissions were sent (February) and presented (March).**
- Collaborative Academic-Industrial workshop was held (March through August).
- **Revised submissions were sent (August) and presented (September).**
- KUALI-BEH carried out a case study in a Software Development Organization (October).
- **Joint initial submission was sent (November) and presented (December).**

2013

- **Joint revised submission was sent (February) and presented (March).**
- **Vote-to-Vote (V2V) and Voting for Recommendation (V2R) were passed.**
- **Architectural Board (AB) endorsed the specification.**
- **Platform Technical Committee (PTC) recommended it.**
- **Business Committee (BC) approved the specification.**
- **Board of Directors (BoD) voted for its adoption.**
- **ESSENCE BETA was released.**
- First Software Engineering bachelor course using KUALI-BEH took place (February to June).
- Case study in a Software Developer Organization (May through July).
- **Finalization Task Force (FTF) started the finalization process (June through December).**
- Second Software Engineering bachelor course using KUALI-BEH took place (August trough December).

2014

- KUALI-BEH Tool was released (January).
- **Finalization Report and the specification draft were sent (February).**
- **ESSENCE 1.0 became OMG standard.**
- **Revision Task Force (RTF) was chartered.**
- KUALI-BEH Formalization phase using Description Logic started (April).

Figure 9-1: KUALI-KAANS, KUALI-BEH and OMG milestones by year.

Engineering.

The resulting framework had to be guided by five premises: (i) the provision of foundations in order to define Methods and Practices, both of which would be the central concepts of the framework; (ii) the inclusion of the elements required to enact a method during an endeavor, focusing on what to produce and how to produce it; (iii) the definition of an operation with which to compose practices in order to build a method; (iv) the creation of an infrastructure in which to store practices and methods, thus allowing practitioners to understand, compose and compare them and (v) the targeting of all of the above to software engineers and practitioners.

Following those premises, the mandatory requirements defined in the RFP were grouped into two sets: the Kernel and the Language. According to (OMG, 2011b), an overview of the Kernel mandatory requirements was:

- A domain model containing the common concepts of Software Engineering and their relationships.
- A set of key conceptual elements with concise definitions, their relationships and the metrics needed to asses them, including the definitions of Method and Practice concepts.
- A set of generic activities undertaken by work teams during projects.
- Sufficient scope to support a broad range of projects and endeavors.
- A mechanism to allow Kernel extensions and tailoring.

According to (OMG, 2011b), an overview of the mandatory Language requirements was:

- A static and operational semantics defined in terms of the abstract syntax.
- A graphical and a textual syntax that formally maps onto the abstract syntax.
- Easiness of use, aimed at practitioners at different competency levels.
- Separation of views for practitioners and method engineers.
- Adaptation operations that would permit methods and practices to be composed, separated, merged and modified.

- Enactment support with which to tailor methods, manage work and monitor progress during projects.
- Working examples of in-use methods and practices to demonstrate the use of the Kernel and Language.

By becoming an OMG member, KUALI-KAANS took the first step as regards responding to the FACESEM RFP. The process of creating a proposal and becoming an official submitter is described in the following sections.

9.3 OMG standardization process steps

After the KUALI-BEH project started, and having clarified that the purpose of the project was to respond actively to the FACESEM RFP, the research group had to become an OMG member and follow the standardization process of the consortium. This section explains the OMG standardization process followed by KUALI-BEH and KUALI-KAANS as part of the submitter team.

9.3.1 Initial submissions

The first step that any of the OMG's member organizations must take if they wish to respond to the RFP with a proposal is to send a Letter of Intent (LOI). At the end of 2011, KUALI-KAANS sent a LOI to confirm its willingness to participate as submitters in FACESEM RFP.

The deadline by which to confirm this willingness to participate was November 22, 2011, by which time five other organizations: Fujitsu, Ivar Jacobson International AB, Model Driven Solutions, PNA Group and Softeam had sent a LOI.

The organizations that send LOIs are automatically registered as voters, but any OMG member can be registered as a voter before the registration closes, and in the case of this FACESEM RFP this specifically occurred on February 22, 2012, and 24 OMG members were listed as voters.

On February 20th, 2012, the initial KUALI-BEH submission was sent to the RFP Submissions desk at OMG Headquarters. 5 of the 6 LOI-organizations submitted three proposals. The proposals were published internally for OMG members. These initial submissions were:

- ESSENCE: Kernel and Language for Software Engineering Methods, whose submitters were: Fujitsu, Ivar Jacobson International AB and Model Driven Solutions, supported by 10 other organizations such as the International Business Machines Corporation and Stiftelsen SINTEF.
- SEMDM ISO/IEC 24744:2007 Software Engineering – Metamodel for Development Methodologies standard submitted by Softeam.
- KUALI-BEH.

The three proposals were placed in the OMG document server and the Four-Week Rule was applied. This rule states that "Any document to be presented to the membership for consideration (a poll) at a Technical Meeting must be available to the entire membership four weeks prior to the beginning of an OMG Technical Meeting" (OMG, 2008a), and its intention is to ensure an adequate period of time in which to review the proposals.

Four weeks after the submission, a Technical Meeting took place in Reston, VA, USA in March, 2012. There, the Analysis and Design Task Force (ADTF) set the official presentations of the proposals on March 21. A meeting organized by the submitters took place on the morning prior to the presentations. The meeting was attended by many members of the ESSENCE submission team and the two members of the KUALI-BEH submission team, while one person from SEMDM attended via virtual means.

During the meeting, each team had a time slot in which to briefly present their proposals, and a discussion later took place which was focused on two points: a first-hand clarification of any doubts and questions related to the other proposals and the indication of any elements that could be taken from each proposal to create a new one as a next step. The meeting took place in a friendly environment and was, from our point of view, an introduction to the OMG world.

In the afternoon, the official presentations to the ADTF started. The objectives of the presentations are, according to (OMG, 2008a):

- To assess how well the submission(s) meet(s) the requirements stated in the RFP;
- To determine whether a vote to issue can occur or whether it is necessary to make a change to the RFP Timetable.

As newcomers, we expected that a specific evaluation panel would be conducting the session, taking notes and asking questions, but this did not occur, or at least not explicitly. The session was directed by a chair in charge of protocol issues who directed the part related to questions and answers. The order of the presentations was SEMDM, KUALI-BEH and ESSENCE, followed by many questions that were principally focused on how each proposal would deal with Risk Management during a project and why none of the proposals re-used SPEM, this being one of the hot-topics at that time.

Moreover, the three graphical representations received a great deal of criticism, and it was for this reason that the ADTF convoked a meeting on the following day, at which an OMG member presented a suggestion on how to manage this issue. The suggestion was the on-going work of another RFP, and was a graphical representation based on a set of simple un-colored geometrical figures that are rapidly gaining popularity with FACESEM submitters.

On the following and final day of the meeting we received an offer from the ESSENCE team to join submissions. If we had been told that this would happen before the whole OMG adventure started, we would not have hesitated to accept, but the offer was declined thanks to the favorable opinions regarding our proposal, and KUALI-BEH continued as a separate effort, at least until the following deadline.

After the preliminary evaluation of the submissions had taken place, the ADTF stated that the revised submission deadline would be August 13, 2012.

9.3.2 Revised submissions: A first approach and fusion

Back in Mexico we continued working on KUALI-BEH, and particularly on the three elements that the OMG evaluation team had identified as being strong:

- The method and practice concepts needed to express and structure ways of working in Software Engineering.
- The template-based approach, which was focused on practitioners.
- The method properties defined to establish whether a method is well-formed.

It was at that moment that the Collaborative Academy-Industry Workshop started, and a proof of concept was completed with promising results. During the workshop KUALI-KAANS obtained invaluable feedback that served as a foundation for the KUALI-BEH revised submission. A clearer definition of terms was constructed, an enactment description closer to reality was built and, more importantly, practitioners who are active in the IT industry found that KUALI-BEH was useful and easy to understand and apply. The workshop also permitted us to distribute and spread the word on KUALI-BEH, thus leading the supporter organizations for the revised submission to increase from 2 to 9.

On August 13, 2012, the new version of KUALI-BEH was sent to the OMG headquarters. On that occasion only two submissions were received, KUALI-BEH and ESSENCE, while Softeam was in negotiations to become part of the ESSENCE submitters' team. At that point the communication between the submitter teams increased considerably in search of a joint submission by the end of the year. Both submissions were sent to each of the SEMAT chapters with a request for opinions on how to manage a hypothetical fusion.

The first step was to create a conceptual mapping between the proposals in order to corroborate the linguistic proximity between terms and definitions. The Latin American chapter used this mapping to generate a report (Zapata-Jaramillo, 2012) that highlighted:

- The importance of keeping the structure proposed in ESSENCE as the basic element of the Kernel.
- The importance of keeping track of the states of both methods and practices, as suggested by the KUALI-BEH proposal. We should also recognize the work involved as regards defining the enactment states and the instance states.
- The importance of concepts such as purpose, objective, measure, and knowledge and skills, considered in KUALI-BEH.
- There was no agreement as to the definition of the Practice concept.

Finally, the report concluded that "More than having similarities, both proposals seem to be complementary to each other" (Zapata-Jaramillo, 2012).

Figure 9-2 shows the mapping carried out by the SEMAT Latin America chapter.

After the report had been sent to other chapters, we received a visit from the SEMAT Latin America chair in Mexico, who facilitated communication between the other chapters, China and

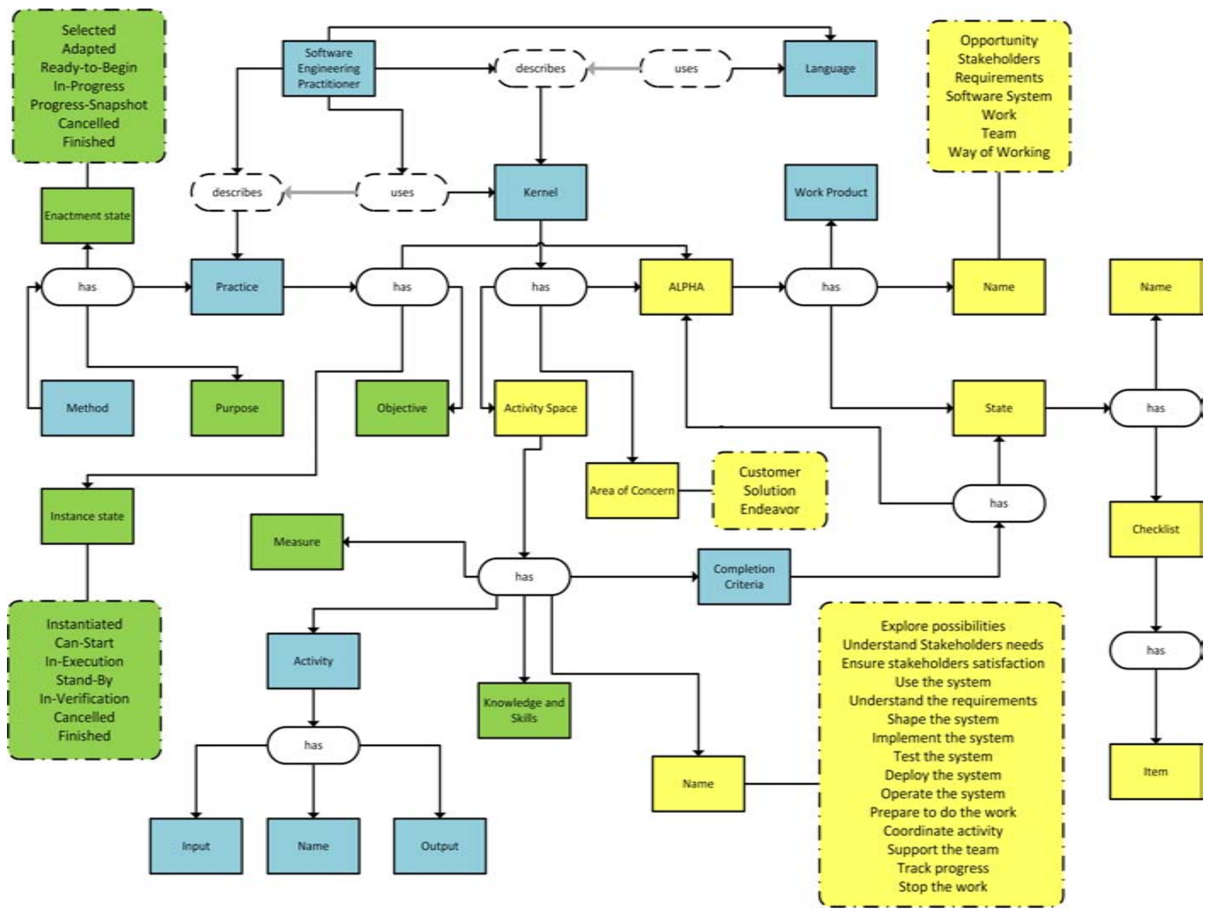


Figure 9-2: Pre-conceptual-schema-based representation of the ESSENCE-BEH proposal, adapted from (Zapata-Jaramillo, 2012).

South Africa, and with the ESSENCE team. The negotiations started with the objective of integrating KUALI-BEH into ESSENCE.

These negotiations led to the definition of a merging proposal, which could be summarized as follows:

- The KUALI-BEH Practice concept must be a language construct with attributes such as objective.
- Alphas must be considered as Practice inputs and outputs.
- The ESSENCE Completion Criteria concepts must include criteria related to work products and/or conditions.
- The KUALI-BEH Measure concept must be included as a language construct.
- The KUALI-BEH Method concept must be a language construct with attributes such as purpose.
- The Method must have a procedure to evaluate its properties of Coherency, Consistency and Sufficiency.
- KUALI-BEH views must constitute an extension of ESSENCE.

The negotiations ended a few hours before the next Technical Meeting started. That Technical Meeting was held in Jacksonville, FL, USA on September, 2012. Both submitter teams had a meeting prior to the presentation session in order to seal the negotiations in person. On that occasion KUALI-KAANS had sent only one person to represent KUALI-BEH, while ESSENCE had full-team in the room, including Ivar Jacobson, who the authors had met personally earlier that year in Reston when the first join attempt did not come into being.

After discussing each of the agreements and putting the final touches to it, KUALI-KAANS, Fujitsu, Ivar Jacobson International AB and Model Driven closed the deal and prepared a joint presentation. Until that moment, each team had its own presentation “just in case”. The official presentation was developed, the announcement of the fusion was effectuated and the ADTF set a new deadline for the joint submission for November 12, 2012.

9.3.3 Revised submission: The joint submission and voting

After the Jacksonville meeting, the major task was to carry out each of the agreements and generate the joint submission. With that in mind, the submitter team was divided into two working groups, one assigned to the Kernel and the other to the Language. Fortunately, the agreements were described in detail and tightly scoped as a result of the previous negotiation, thus making the execution straight forward. For the task to be completed it was necessary for meetings to take place every two weeks in order to assign activities, report status and monitor the progress of the joint submission.

The joint submission was sent before the November's deadline, and on December 07 2012 the Technical Meeting was held in Burlingame, CA, USA. There the presentation describing the submission was focused on how both proposals complemented each other and created a robust submission that still dealt with the mandatory requirements defined by the FACESEM RFP. After the presentation, the submitter team had to respond to the comments and suggestions made by the evaluation team, and the subject of SPEM was again brought up, with IBM being its main supporter. The reasons given for not basing the submission on SPEM were presented, which convinced the evaluation team but not IBM.

The submitter team then called for the Vote-to-Vote (V2V). The spirit of V2V voting is to obtain a consensus as to whether the submission is sufficiently mature, and at least 75% of the members on the voting list wanted to proceed to the recommendation voting.

That day 15 of 24 the OMG members who were registered as Voters were present, and the V2V started. At the end of the voting the 75% of YES votes required had not been achieved, signifying that the ADTF had to request another revision cycle. Until the deadline of February, 2013, the submitter team received and responded to the comments from the evaluation team and prepared the new version of the revised submission.

The next Technical Meeting took place in Reston on March 2013. The revised submission was presented and the submitters responded to each of the evaluation team's issues. A V2V was again called and this time the75% of the YES votes required were easily achieved. This permitted a call for the Voting for Recommendation (V2R), which also was passed with 85% of YES votes.

The ADTF recommended the proposed specification to the Architectural Board (AB), which endorsed the specification a day later. The submission then went to the Platform Technical Committee (PTC), which was at that time formed of 58 OMG Platform members. The PTC voting is carried out on-line in order to give all Platform members the opportunity to vote. 41 members voted with only 3 NOs, and the PTC voting was thus passed.

The submission next went to the Business Committee (BC), which is the part of the Board of Directors (BoD) in charge of determining whether a submission is commercially viable and an implementation of it is imminent, and a favorable opinion was obtained. After the BC evaluation had been passed, the last step was to wait for the BoD voting.

The BoD's announcement was eventually made, and ESSENCE became an OMG BETA specification and its finalization process began.

9.3.4 Finalization

A Finalization Task Force (FTF) is responsible for drafting the changes that convert an adopted submission into a formal specification (OMG, 2008a). FACESEM FTF was chartered after the AB had voted, but work was started on it only after the BoD had voted. The first FTF working meeting was held in Berlin, Germany, on June 18, 2013, where the BETA specification was made publicly available and the strategy needed to receive, analyze and respond to the proposed changes, suggestions and recommendations was defined.

FACESEM FTF was formed of 9 members from 9 different organizations, including KUALI-KAANS, and the majority of the FTF members were non-submitters. The time limit to receive issues was December 09, 2013. In the meanwhile, any person or organization, OMG members or otherwise, could download the BETA version of the specification, review it and submit its issues to FTF by using a form on the OMG web page or sending it by email. When an issue was received at the OMG, it was adapted and redirected to the FTF chair. The FTF chair is in charge of coordinating the FTF tasks, and principally prepares the FTF report and drafts the Alpha specification. Note that issues can be sent by FTF members, OMG members and non-OMG members, and that an important source of issues is also the AB and the evaluation team. After the reception of issues had closed, the FTF members met at the Technical Meeting in Santa Clara, CA, USA, on December 09, 2013. At that meeting, the strategy needed to manage the issues was confirmed, and the software tool that permits us to view, sort, prioritize, give status, define its impact, propose solutions and assign tasks to FTF members was introduced. After using this tool, each of the registered issues was responded to.

The issues were prioritized, and a subset of issues was selected and assigned to particular FTF members. These FTF members were responsible for proposing a solution, but anybody else could also propose solutions. The FTF discussed and defined solutions for the issues, and when a response had been obtained for all the issues in the subset, a ballot was created and the FTF could vote as to whether or not the response to the issues would be applied, or the voter could abstain. In order to quorate a ballot, at least 5 of the 9 FTF members had to vote, and if a member did not vote in two ballots, then he or she had to leave the FTF, although this did not occur at this FTF.

Three days later the ballot was opened and the issues that had been passed were applied. This process took place 4 times until all the issues had been responded to. The FTF report and the formal specification draft were then prepared.

Once the draft of the specification, in conjunction with the OMG Technical Editor, and the FTF report had been completed, both were submitted to the OMG Headquarters on February 26, 2014. The four-week rule was then applied and the FTF presented their results to the AB during the Reston Meeting in March, 2014, which endorsed the adoption of the specification. The PTC later voted its adoption, followed by the BC review and culminating with the BoD verdict. This became ESSENCE 1.0 (OMG, 2014), an OMG formal specification of which KUALI-BEH is part, after three years of hard work.

9.3.5 Post-Adoption

The ESSENCE standardization process will continue and it is currently at the Revision phase, during which a Revision Task Force (RTF) is chartered, and is in charge of producing new minor revisions to existing, formally published specifications (OMG, 2008a). KUALI-KAANS is again part of the RTF.

The RTF operated much as the FTF did as regards collecting issues, resolving them and voting on the resolutions, culminating in the 1.1 version of ESSENCE.

The next step in this process will be to satisfy a Testing Task Force (TTF). The TTF supports the standardization of the test suites that will be used in certification programs (OMG, 2008a).

Finally, the last step is the Specification Retirement; here a Request for Retirement (RFR) is called in order to retire a formal specification from OMG adopted technologies. An action of this type would occur if an adopted specification were to be superseded, was never fully implemented, or had simply reached the end of its useful life (OMG, 2008a).

After being part of the process, KUALI-KAANS has some suggestions that would improve the OMG standardization process:

- The chairs in charge of managing the presentation sessions should not be part of any of the submitter teams involved in the session;
- The Evaluation team should be explicitly named to the submitters and it should identify itself clearly during presentations;
- The Evaluation team should prepare and send an individual evaluation report of each submission to submitters; and
- In order to motivate and facilitate the participation of more universities in the OMG, it would appear to be fair to reduce the membership and registration fees for universities, taking into account that they are non-profit organizations.

9.4 Lessons learned

The Industrial and Academic spheres are complementary *worlds of knowledge* that resist working together, and it would appear that interest is always present, and that the objectives pursued are very close but achieving a real and effective collaboration is difficult. After three years of continuous and productive synergy, we can conclude that participating in this process has been a vast and rewarding experience. The lessons learned during this period of time concern:

The wide variety of IT standards. During the development of the standard, we discovered the existence of a wide variety of IT standards (supported by ISO, IEC, IEEE and also OMG) that lack an aligned definition as regards the use of terms. This is not a recent phenomenon, and dates from more than a decade ago (Rout, 1999). According to (Henderson-Sellers et al., 2014) the terminology definitions vary significantly among standards, even in those backed by the same standardization body. Moreover, it is stated that across pairs of similarly-focused standards semantics of the terms can often be contradictory and misaligned.

Efforts to harmonize proposals and create ontologies have been made in order to align the deviation between definitions. In (Pardo et al., 2012) a framework for supporting the harmonization of multiple models is presented, while in (Henderson-Sellers et al., 2014) the creation of a comprehensive set of definitions that conforms to a standard domain ontology is suggested.

Any standardization process therefore continuously deals with the challenge of unifying criteria, creating homogeneous definitions and harmonizing concepts. In the context of our standardization process these situations were handled in different ways, and in our opinion, the most effective means of management is to ask a team of volunteers to deal with the issue, allow the team to present their solution, then reach a consensus and apply the solution. During the FTF the chair also prepared a ballot to vote for the resolution of issues, which improved the whole process and made it agile.

The OMG standardization process. As for the rules and procedures of the OMG standardization process, we discovered and suggest that:

- It is agile but not easy to understand and follow, and as newcomers we had to study the process in depth and it was frequently necessary to seek clarification;
- Negotiation with other submitters sometimes had to be put off or did not take place immediately owing to their business commitments;
- An *Evaluation Team* should be named and formal *evaluation reports* should be produced for each submission. It would be very useful to have an extra meeting with the submitters during the Technical Meetings in order to discuss these reports.
- The Evaluation Team should *analyze and grade the conformance to the mandatory requirements* one-by-one, making its analysis available to all the Voting members.
- A more "democratic" evaluation might be carried out in order to discover and accept how each mandatory requirement has been responded to, thus allowing the *Voting members to vote for each requirement and decide which of the proposals deals with them best.* This would also permit a discussion on how to manage opposing ideas to be opened.

- The work between meetings should be monitored by an *OMG Control Force* or by a web-based system like that of the FTF; this would allow all the issues contained in the evaluation reports to be correctly managed and tracked.
- An *OMG Intermediary Force* should be available *to guide negotiations between submitters.* It is a noble OMG tradition to motivate the fusion of submissions, but it should be seriously considered that if no agreement is reached between submitters, this will not imply that one party should prevail.
- This Intermediary Force should also develop *conciliation tasks* based on the Evaluation Team's reports *regarding the equality between members.* In our experience, most of the OMG members are influential organizations in the IT sector that have power when a decision has to be made. Bearing in mind that not all members have this kind of weight, their proposals should not be considered less valuable or less significant.
- This Intermediary Force should also indicate how to join submitted proposals and which aspects should be joined, mostly based on the voting.
- *V2V voting could slow down the process*, and in our experience particularly, the V2V stretched the process out over four months to no apparent benefit.
- A mechanism to ensure that supporting organizations are really interested in and support the creation of the proposal, and not only in words.

The appreciation of standard-related work from the scientific world. Standardization processes require the consensus of industry and academics, and standardization signifies collecting accepted and generally consented knowledge that has proven to be effective in practice. In order to carry out this process, members of both parties should attend meetings, defend positions, discuss and make agreements that are always focused on transforming ideas into proposals and finally into standards. As a whole, standardization implies a huge endeavor that can take up years of a researcher's work.

However, standardization efforts are not appreciated in the research curricula, thus making noteworthy projects of this nature very difficult within the academic community. Academy members are far more inclined to attempt to publish papers, one after another, in recognized journals and conferences, induced by the publish or perish threat, thus leaving the important and fundamental task that is standardization to one side.

The validation of proposals. OMG RFPs clearly state the need to demonstrate the usefulness of a proposal by showing a *Proof of Concept*. Its proof must be presented from the initial submissions. But, how would it be possible to manage the dilemma of validating something at a very early stage? How can the skepticism of aspirant "guinea pigs" be dealt with? Moreover, are organizations interested in testing something that is not yet a standard?

In particular, when somebody was interested in testing KUALI-BEH, a collaboration had to be planned, on the one hand because a training course concerning the proposal had to be prepared and on the other because the participating organization had to find a suitable project in which to carry out the case study. Negotiations and agreements could take weeks or months when following these steps.

As an alternative, we developed a collaborative workshop to which organizations and active software engineers were invited. During the workshop we trained them in KUALI-BEH and they also had the chance to try out elements of KUALI-BEH part by part. At the end of the workshop the organizations then asked for a continuation and we had the opportunity to carry out case studies with already trained participants.

The need for tools to facilitate the use of a standard. It should be emphasized that there is an OMG statement to the effect that before an OMG standard is officially adopted, the BoD must ensure that the submitter team will implement or use the specification in a product. This is a distinguishing constraint and an advantage in comparison to other standardization bodies.

KUALI-KAANS developed a tool focused on practitioners in order to create, modify and compose methods and practices. It is worth mentioning that the tool was developed by Master's degree students who were involved in the effort, and that this was a valuable experience for them.

9.5 Conclusions

SEMAT's phrase *"support a process to refound Software Engineering based on a solid theory, proven principles and best practices"* (Jacobson et al., 2009) delimited a common ground for the community involved in the discipline, which includes theorists and pragmatics. The call attracted people from all around the globe who started to contribute on this line. A mechanism with which to manage and control it was therefore needed. Thanks to its endorsement by the OMG it became possible to establish a process to create an IT standard that dealt with the Software Engineering community's initiative.

Moreover, the significance of the SEMAT Call for Action and later FACESEM RFP became an international collaboration. Researchers and practitioners, academics and industry, worked together with the aim of creating a standard that moved them closer to the goal.

In this thesis we have presented how our research group responded and created KUALI-BEH, thus making an active contribution to the SEMAT initiative, and most importantly, to the OMG specification process when the proposal was integrated with ESSENCE and constituted an improved standard (Morales-Trujillo et al., 2015a).

Although the standardization process followed in OMG proved to be strong and effective, it still has some room for improvement. We believe that the evaluation process of submissions could be enhanced by focusing on the individual analysis and assessment of each mandatory requirement defined in the RFPs, rather than evaluating the submission as a whole. It is important for evaluation reports containing the analysis results to be delivered to submitters in order to improve their submissions. An Intermediary Force to manage fusions between submissions may also be of help, and lastly rules such as V2V could be avoided.

After living through this experience we have realized that there is a wide variety of standards that undertake similar aims. Nevertheless, they manage terms and definitions that differ in usage and meaning, thus making it difficult to unify criteria. ESSENCE is not the only work that seeks to homogenize concepts. (ISO/IEC, 2007b) and (Henderson-Sellers et al., 2014) present very promising alternatives and results to consider in order to confront this challenge.

Various lines for future work have been identified; on the one hand, KUALI-KAANS will continue with the Formalization step of the KUALI-BEH project and finalize the construction of two software tools to facilitate the use of the framework by practitioners and organizations. On the other hand, the research group has important background and experience in working with VSEs of which we shall therefore take advantage, in addition to motivating the usage and application of the standard created among VSEs.

Moreover, the post-adoption steps of the OMG standardization process will continue. Since KUALI-KAANS is still part of the RTF, it will continue to develop corresponding revision tasks, will seek feedback from more organizations and will improve the standard.

We trust that this successful experience will serve as another example of how an industrial-academic collaboration can reach a valuable and relevant outcome for both parties. We hope to help to convince universities and organizations to start this kind of collaborations or at least to consider them in a near future.

Part IV

Conclusions

Chapter 10

Conclusions

"Sto ancora imparando."
"I am still learning."

Michelangelo di Lodovico Buonarroti Simoni. Sculptor, painter, architect, poet and engineer (1475 – 1564).

This chapter presents conclusions and final results obtained from this research. We start by analyzing the achievement of the objective and goals of the thesis, then we enlist publications derived from this research and finally discuss future research lines.

10.1 Analysis of research goals

This thesis has presented KUALI-BEH, a bottom-up metamodel that provides software engineering practitioners with an authoring framework with which to express, adapt and share their ways of working as a collection of methods and practices.

In chapter 1 the main objective of our research was defined as:

To define a kernel of common concepts/framework that supports the expression of practitioners' ways of working carried out during software projects.

The achievement of the main objective is based on the attainment of five Goals (Gs). The achievement of these goals and consequently also of the research objective is explained as follows:

G1. To identify the universals of software projects carrying out an in-depth study in Method Engineering, Situational Method Engineering, Process Reference Models, Process Metamodels and related Standards.

Chapter 3: State-of-the-Art presented a systematic review of the literature which deals with proposals that support description and modeling of software processes. Also it classifies and analyses the main findings that influenced this research. *Chapter 4: Identification* summarized the first engineering cycle of this research. During that cycle KUALI-BEH common concepts were identified, defined and modeled.

G2. To define a kernel of common concepts which facilitates the transformation of practitioners tacit knowledge into explicit.

Chapter 5: Software Project Common Concepts presented a full description of the kernel of common concepts that shape KUALI-BEH. Furthermore, it described an introduction to the Authoring Extension designed in order to facilitate the transformation of practitioners' tacit knowledge into explicit knowledge.

G3. To formalize the description of methods as a composition of practices using the defined language of common concepts.

Chapter 6: Formalization presented formalization of KUALI-BEH common concepts, its properties and its adaptation operations. The approaches used to achieve this goal were: Sets Theory, Ontologies and Description Logic.

G4. To establish a conceptual framework that supports understanding between concepts, terms and relations around software projects.

Chapters 5: Software Project Common Concepts and *6: Formalization* established a conceptual framework of KUALI-BEH, so the concepts, terms and relations around software projects are clearly understood and traced.

G5. To validate the framework through case studies.

Chapters 7: Collaborative Workshop on Software Engineering Methods and *8: Case Studies* presented the validation process of KUALI-BEH. First through a collaborative workshop attended by active practitioners from industry and academy, and later through three case studies in real organizations. During this experience the usefulness and sufficiency of the framework and its common concepts was validated.

O. To define a kernel of common concepts/framework that supports the expression of practitioners' ways of working carried out during software projects.

As a whole, based on the achievement and fulfillment of the five goals we can establish that the main objective of this research was achieved. KUALI-BEH is a kernel of software project common concepts that supports practitioners in the definition of their ways of working. KUALI-BEH elements (conceptual, methodological and technological) were applied in real-world situations supporting the purpose for which it was created.

10.2 Lessons learned from applying the research method

After three years of continuous work, we can conclude that TAR resulted in a valuable research method to the purposes of this thesis. Its application, in combination with case studies, allowed us to validate and improve KUALI-BEH after each engineering cycle. The iterative approach of TAR allowed us to obtain continuous feedback and validation of the artifact in a real context, which are two TAR's main advantages in our opinion.

As researchers we had the opportunity to develop the three roles identified by Wieringa: Designer, Helper and Researcher. Starting as designers, we created an artifact aimed at resolving a class of problems present in the industry. During the engineering cycles we inserted the artifact into organizations which were affected with this class of problem. In order to apply the proposed treatment, and so acting as helpers, we used the artifact and assessed its functioning.

Later on, taking advantage of the lessons learned and now as researchers, we analyzed the obtained effects, the achieved value and the resulted trade-offs with the aim of adjusting and improving the artifact.

Validating the artifact in a real context allowed us to gain experience and generate knowledge through the lessons learned, which according to (Endres and Rombach, 2003) are the main means of obtaining knowledge in Software Engineering discipline.

On the other hand, TAR reported benefits for the organizations involved in this research. At the end of the case studies we could establish that:

- Each organization achieved the stated objectives.
- Practitioners were trained into a new technology, in this case KUALI-BEH, with no “extra” cost.
- The results of each case study were used to achieve business goals, it means, the artifact resulted useful and applicable for their particular contexts.
- Collaboration and partnership ties between the university and the organization were promoted.

Finally, we can define TAR as a research method which gives researchers a valuable opportunity to learn and obtain knowledge by applying theories into practice in order *to solve real problems solving real problems.*

10.3 Support for results

The main results, produced through the development of this PhD thesis, have been published in Software Engineering related forums. These publications are organized by forum type in the following subsections.

10.3.1 International standards

1. *ESSENCE – Kernel and Language for Software Engineering Methods.* Object Management Group (OMG), 2014.

10.3.2 Papers in international journals

1. Miguel Morales-Trujillo, Hanna Oktaba and Mario Piattini. *The Making of an OMG Standard.* Computer Standards & Interfaces. ISSN: 0920-5489, 2013 Impact Factor: 1.177, DOI: 10.1016/j.csi.2015.05.001, 2015.

10.3.3 Papers in international conferences

1. Miguel Morales-Trujillo, Hanna Oktaba and Mario Piattini. *Using Technical-Action-Research to Validate a Framework for Authoring Software Engineering Methods.* 17th

International Conference on Enterprise Information Systems (ICEIS'15). April, 2015. Barcelona, Spain. Pages 15–27. ISBN: 978-989-758-097-0. Acceptance ratio: 22%. **Best Student Paper Award**.

2. Miguel Morales-Trujillo and Hanna Oktaba. *Improving Software Projects Inception Phase Using Games: ActiveAction Workshop.* 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'14). April, 2014. Lisbon, Portugal. Pages 180–187. ISBN: 978-989-758-030-7. Acceptance ratio: 33%.

3. Miguel Morales-Trujillo and Hanna Oktaba. *Identificación y Formalización de un Núcleo de Conceptos Comunes para Proyectos de Desarrollo de Software.* XV Iberoamerican Conference on Software Engineering (CIbSE'12). April, 2012. Buenos Aires, Argentina. Pages 313–318. ISBN: 978-1-62993-961-2. Acceptance ratio: 27%.

10.3.4 Papers in national conferences

1. Emmanuel Arroyo-López, Teresa Ríos-Silva, Miguel Morales-Trujillo, Alejandro Rico-Martínez and Hanna Oktaba. *Expresando Nuestra Manera de Trabajo con KUALI-BEH: Lecciones Aprendidas por Tic-Tac-S.* Congreso Internacional de Investigación e Innovación en Ingeniería de Software (CONISOFT'15). April, 2015. San Luis Potosí, San Luis Potosí, Mexico.

2. Miguel Morales-Trujillo and Hanna Oktaba. *Identification and Formalization of Software Project Common Cocepts: KUALI-BEH.* Congreso Internacional de Investigación e Innovación en Ingeniería de Software (CONISOFT'13). October, 2013. Xalapa, Veracruz, Mexico.

3. José Urrutia, Rodrigo Barrera, Eraim Ruiz, Miguel Morales-Trujillo and Hanna Oktaba. *Entorno Computacional Basado en KUALI-BEH para Apoyar a Organizaciones en el Desarrollo de Software.* Congreso Internacional de Investigación e Innovación en Ingeniería de Software (CONISOFT'13). October, 2013. Xalapa, Veracruz, Mexico.

10.3.5 Papers under review in international journals

1. Miguel Morales-Trujillo, Hanna Oktaba and Francisco Hernández-Quiroz. *Towards a Formalization of a Framework to Express and Reason about Software Engineering Methods.* Formal Aspects of Computing. ISSN: 0934-5043, 2013 Impact Factor: 0.609 (Submitted April 2015).

10.4 Research contributions

The main contributions of this PhD thesis are classified and described as follows:

- **Theoretical Software Engineering:** Situating software engineering practices on universal elements will give solidity and soundness to things built with them. Formally defining practices will identify the commonalities between them, leading to a better way to collect and share knowledge (Jacobson et al., 2007).

  This PhD thesis identified the universals of software projects carrying out an in-depth study in Method Engineering, Situational Method Engineering, Process Reference Models, Process Metamodels and related Standards.

  Through the development of the research, it was confirmed that the knowledge collected in these corpus depends on a top-down approach which is then acquired by organizations, and this does not permit the organization's practitioners to express their own practices.

  In contrast, the KUALI-BEH bottom-up approach implies the expression of practitioners' tacit knowledge in order to build up the organization's way of working. It allows practitioners to express, adapt and share their ways of working as a collection of methods and practices.

  KUALI-BEH conceptual framework resulted in a valuable alternative that supports the understanding between concepts, terms and relations around software projects. As a result it contributes to the enrichment of Software Engineering theory by collecting the practitioners' knowledge.

- **Practical Software Engineering:** After validating KUALI-BEH through a collaborative workshop and three case studies, we concluded that it is a valuable alternative for practitioners and small organizations, since it provides a first step to bridge the gap between Software Engineering theory and practice. KUALI-BEH permits small organizations to create an organizational method repository comprising their knowledge, and to gradually introduce them to the adoption of standards and reference models.

KUALI-BEH enabled three entities to undertake an authoring project that led them to a software process improvement. The bottom-up strategy supported the transformation of practitioners' tacit knowledge into explicit knowledge, in other words, they achieved the representation of their own ways of working.

In consequence, with their ways of working expressed, a deeper reasoning about it was possible, allowing practitioners to adapt, share and compare it in order to improve it. As a whole, the combination of these factors derived in valuable results for the organizations that applied KUALI-BEH, becoming a solution within reach for practitioners and effectively achieving the organization's needs in the industrial context.

- **Standardization related efforts:** In 2009 SEMAT launched a Call for Action to refound Software Engineering. Months later the Object Management Group endorsed this initiative and prepared FACESEM RFP, a request for proposals that would deal with SEMAT concerns, and began the standardization process. We responded to the request as a submitter by creating the KUALI-BEH proposal, thus joining the standardization effort from the beginning.

  Three years later, the active contribution to the OMG standardization process generated valuable lessons learned, which concern the wide variety of IT standards and their lack of aligned definitions, the importance and necessity of standardization efforts despite their not being appreciated by the academic community, and the advantages and beneficial results of industrial-academic synergy.

  Also, some suggestions and improvement opportunities to make the most of the OMG standardization process were proposed.

  Finally, throughout the OMG standardization process, KUALI-BEH is making an active contribution to the identification of both the theoretical and practical universal elements. Its fusion with the ESSENCE proposal and its eventual appearance as a useful and applicable standard completed the standardization effort.

- **Education in Software Engineering:** Furthermore, KUALI-BEH metamodel has been used in an educational context with beneficial results, thus bringing Software Engineering theory and practice closer to bachelor degree students.

  The Software Engineering undergraduate course for the Bachelor's degree in Computer Science at the Science Faculty of the UNAM, has been redesigned using the KUALI-BEH approach.

  The theoretical part of the course is presented and supported with a set of practices that expresses distinct ways of working in real life. Each of these practices contains guides and techniques with the expectation of facilitating the students' understanding and application of Software Engineering theory.

  The practical part of the course is based on the practice instance execution and method enactment, following those defined in the KUALI-BEH operational view and supported by the boards defined as an alternative in managing the project developed by the students.

  The course was taught by two professors to 17 students in the last semesters of the Computer Science Bachelor's degree. The students' reactions highlighted the importance to execute real practices based on examples, and how as a team they can drive their own work using the practices as work units to be distributed within the team. From the academic perspective, the students are learning and practicing a broad scope of real software engineering in an academic environment, which will hopefully better prepare them for their inclusion in industry.

  The first courses took place in 2013 and, taking into account the results of these first courses, two more courses were carried out in 2014 with promising results. Since then, this course has been adopted the KUALI-BEH approach.

10.5 Future lines of research

The following research lines are identified as future work:

- **To enhance the robustness of KUALI-BEH:** In order to improve and up to date the proposal, more case studies should be designed and executed. KUALI-BEH should be applied and used by more practitioners from different organizations.

- **To develop a set of KUALI-BEH profiles:** KUALI-BEH proved its usefulness in industrial and educational contexts, however it was observed that some modifications had to be done in order to satisfy particular issues according to each context. At the moment the following profiles are offered:
  - **Educational.** During the bachelors' courses that were carried out using KUALI-BEH, it was observed that the approach must be prescriptive instead of descriptive mainly because of the experience level of the practitioners, in this case college students.

    Due to this fact common concepts like *Task* were not necessary because of the level of detail of practices, so it must remain as simple as possible. Also the operational boards should be adapted, in this case Method Enactment board was replaced by a KANBAN board, again looking for a simpler alternative.
  - **Agile.** To create a "light" version of KUALI-BEH in order to satisfy the agile trend to make everything simpler.
  - **Programming and Databases.** KUALI-BEH was used to describe processes related to programming and database management. According to this experience it was observed that the *Verification Criteria* had no value for practitioners, due to the fact that the verification is obtained immediately after the execution of a script, a piece of code or an user interaction end.
- **To develop and release a fully functional technological environment:** This will motivate more practitioners to use KUALI-BEH and will result in the spread of the proposal. At the same time it is desirable to add the following functionalities to the already created tools:
  - **Collaborative approach:** The technological environment should be focused on promoting interaction and collaboration of the work team by integrating collaborative elements as virtual boards and desktops. The aim is to apply technology that endorses

participation, discussion, collaboration and cooperation.

  - **Graphical manipulation of methods.** The tool should provide a functionality to compose and adapt a method graphically, using a Drag-and-Drop approach.
  - **Statistical analysis and prediction of measures.** A module that analyzes projects' data is desirable. This functionality should provide practitioners with an alternative to register the data collected during a project. It will also estimate measures based on the historical register of data collected from previous projects.
  - **Progress control of a project.** The tool can send automatically emails and reminders to work team members in order to monitor and keep informed everybody.
  - **Work products management.** The tool should interact with the work products repository of the organization in order to facilitate the version control strategy and the access to work products linked to an specific practice.

- **To create a global MPI:** In order to share and compare different ways of working it is mandatory to create a repository capable of storing methods and practices coming from everywhere. This work should start within one organization, then spreading on between other organizations. On the one hand, this network will help practitioners to understand and improve their own ways of working comparing them against others. On the other hand it will comprise a valuable source of knowledge for method engineers.

  The global MPI can serve as a bridge between theoretical and practical knowledge of Software Engineering.

Bibliography

Abad, Z. S. H., Sadi, M. H., and Ramsin, R. Towards tool support for situational engineering of agile methodologies. In *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*, pages 326–335. IEEE, 2010.

Abran, A., Cuadrado, J., García-Barriocanal, E., Mendes, O., Sánchez-Alonso, S., and Sicilia, M. Engineering the ontology for the swebok: Issues and techniques. In Calero, C., Ruiz, F., and Piattini, M., editors, *Ontologies for Software Engineering and Software Technology*, pages 120–138. Springer Berlin Heidelberg, 2006.

Alexander, C. *The Timeless Way of Building*. Oxford University Press, 1979. ISBN 978-0195024029.

Arni-Bloch, N. Towards a came tools for situational method engineering. In *Interoperability of Enterprise Software and Applications (INTEROP'05)*. IFIP-ACM, 2005.

Arroyo-López, E., Ríos-Silva, T., Morales-Trujillo, M., Rico-Martínez, A., and Oktaba, H. Expresando nuestra manera de trabajo con kuali-beh: Lecciones aprendidas por tic-tac-s. In *Memorias del Congreso Internacional de Investigación e Innovación en Ingeniería de Software*, CONISOFT '15, pages 25–32. Editorial UABC, 2015. ISBN 978-0-692-43292-1.

Baader, F. and Nutt, W. The description logic handbook: Theory, implementation, and applications. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors, *Basic Description Logics*, pages 47–100. Cambridge University Press, 2003. ISBN 0-521-78176-0.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors. *The De-*

scription Logic Handbook: Theory, implementation, and applications. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.

Barrera, R. and Oktaba, H. Repositorio de métodos y prácticas de proyectos de software para KUALI-BEH. Master's thesis, National Autonomous University of Mexico, 2014.

Basili, V., Selby, R., and Hutchens, D. Experimentation in software engineering. *IEEE Trans. Softw. Eng.*, 12(7):733–743, July 1986. ISSN 0098-5589. URL `http://dl.acm.org/citation.cfm?id=9775.9777`.

Bauer, F. Software engineering. In Freiman, C., editor, *Proceedings of the IFIP Congress (Information Processing 71)*, pages 530–538, Ljubljana, Yugoslavia, August 1971.

Becker-Kornstaedt, U., Scott, L., and Zettel, J. Process engineering with spearmint/EPG. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 791–, New York, NY, USA, 2000. ACM. ISBN 1-58113-206-9. doi: 10.1145/337180.337646. URL `http://doi.acm.org/10.1145/337180.337646`.

Belady, L. and Lehman, M. A model of large program development. *IBM Syst. J.*, 15(3):225–252, Sept. 1976. ISSN 0018-8670. doi: 10.1147/sj.153.0225. URL `http://dx.doi.org/10.1147/sj.153.0225`.

Brinkkemper, S. Method engineering: engineering of information systems development methods and tools. *Information and software technology*, 38(4):275–280, 1996.

Broy, M. Can practitioners neglect theory and theoreticians neglect practice? *Computer*, 44 (10):19–24, Oct 2011. ISSN 0018-9162. doi: 10.1109/MC.2011.305.

Calero, C., Ruiz, F., Baroni, A., Abreu, F., Brito e, and Piattini, M. An ontological approach to describe the sql:2003 object-relational features. *Computer Standards and Interfaces*, 28(6): 695–713, 2006a. ISSN 0920-5489.

Calero, C., Ruiz, F., and Piattini, M. *Ontologies for Software Engineering and Software Technology.* Springer-Verlag Berlin Heidelberg, 2006b. ISBN 3-540-34517-5.

Devanbu, P. and Jones, M. The use of description logics in kbse systems: Experience report. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pages 23–35, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-8186-5855-X.

Dias, M., Anquetil, N., and De Oliveira, K. Organizing the knowledge used in software maintenance. *Journal of Universal Computer Science*, 9(7):641–658, 2003.

Eclipse. Eclipse process framework project (EPF). `http://www.eclipse.org/epf/`, 2013.

Edwards, J. Managing software engineers and their knowledge. In Aurum, A., Jeffery, R., Wohlin, C., and Handzic, M., editors, *Managing Software Engineering Knowledge*, pages 5–27. Springer Berlin Heidelberg, 2003. ISBN 978-3-642-05573-7.

Endres, A. and Rombach, D. *A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories.* Fraunhofer IESE series on software engineering. Pearson-/Addison Wesley, 2003. ISBN 9780321154200.

Engelsman, W. and Wieringa, R. Goal-oriented requirements engineering and enterprise architecture: Two case studies and some lessons learned. In Regnell, B. and Damian, D., editors, *Requirements Engineering: Foundation for Software Quality*, volume 7195 of *Lecture Notes in Computer Science*, pages 306–320. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28713-8.

Franch, X. and Ribó, J. Using UML for modelling the static part of a software process. In *Proceedings of the 2Nd International Conference on The Unified Modeling Language: Beyond the Standard*, UML'99, pages 292–307, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3-540-66712-1. URL `http://dl.acm.org/citation.cfm?id=1767297.1767328`.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN 0-201-63361-2.

García, F., Bertoa, M., Calero, C., Vallecillo, A., Ruiz, F., and Piattini, M. Towards a consistent terminology for software measurement. *Information and Software Technology*, 48(8):631–644, 2006. ISSN 0950-5849.

Gómez Pérez, A., Fernández-López, M., and Corcho, O. *Ontological Engineering*. Springer-Verlag London, 2004. ISBN 978-1-85233-840-4.

González-Pérez, C. and Henderson-Sellers, B. An ontology for software development methodologies and endeavours. In Calero, C., Ruiz, F., and Piattini, M., editors, *Ontologies for Software Engineering and Software Technology*, pages 139–168. Springer Berlin Heidelberg, 2006.

Gruninger, M. and Lee, J. Ontology applications and design. *Communications of the ACM*, 45 (2):39–41, 2002.

Harmsen, F. and Brinkkemper, S. Design and implementation of a method base management system for a situational case environment. In *In Proceedings*, 1995.

Harmsen, F. and Saeki, M. Comparison of four method engineering languages. In *Proceedings of the IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering on Method Engineering : Principles of Method Construction and Tool Support: Principles of Method Construction and Tool Support*, pages 209–231, London, UK, UK, 1996. Chapman & Hall, Ltd. ISBN 0-412-79750-X. URL `http://dl.acm.org/citation.cfm?id=278337.278354`.

Harmsen, F., Brinkkemper, S., and Oei, H. *Situational method engineering for information system project approaches.* University of Twente, Department of Computer Science, 1994.

Harrison, R., Badoo, N., Barry, E., Biffl, S., Parra, A., Winter, B., and Wüst, J. Directions and methodologies for empirical software engineering research. *Empirical Software Engineering*, 4 (4):405–410, 1999. ISSN 1382-3256. doi: 10.1023/A:1009877923978.

Henderson-Sellers, B. Bridging metamodels and ontologies in software engineering. *Journal of Systems and Software*, 84(2):301–313, 2011.

Henderson-Sellers, B. and Ralyté, J. Situational method engineering: State-of-the-art review. *J. UCS*, 16(3):424–478, 2010.

Henderson-Sellers, B., Gonzalez-Perez, C., Mcbride, T., and Low, G. An ontology for ISO software engineering standards: 1) creating the infrastructure. *Comput. Stand. Interfaces*, 36(3):563–576, Mar. 2014. ISSN 0920-5489. doi: 10.1016/j.csi.2013.11.001. URL `http://dx.doi.org/10.1016/j.csi.2013.11.001`.

IEEE. IEEE guide for software quality assurance planning. guide 730.1-1995, Institute of Electrical and Electronics Engineers, New York, NY, 1995.

IEEE. IEEE standard for application and management of the systems engineering process. standard 1220-1998, Institute of Electrical and Electronics Engineers, New York, NY, 1998a.

IEEE. IEEE standard for functional modeling language – syntax and semantics for idef0. standard 1320.1-1998, Institute of Electrical and Electronics Engineers, New York, NY, 1998b.

IEEE. IEEE the authoritative dictionary of ieee standards terms. Dictionary 100-2000, Institute of Electrical and Electronics Engineers, New York, NY, 2000.

IEEE. IEEE standard dictionary of measures of the software aspects of dependability. standard 982.1-2005, Institute of Electrical and Electronics Engineers, New York, NY, 2005.

IEEE. IEEE standard for developing a software project life cycle process description. standard 1074-2006, Institute of Electrical and Electronics Engineers, Piscataway, NJ, 2006.

ISO. Quality management systems - fundamentals and vocabulary. standard 9000:2005, International Organization for Standardization, Geneva, Switzerland, 2005.

ISO/IEC. Information technology - process assessment. standard 15504:2004, International Organization for Standardization, Geneva, Switzerland, 2004.

ISO/IEC. Software engineering - guide to the software engineering body of knowledge (SWEBOK). standard 19759:2005, International Organization for Standardization, Geneva, Switzerland, 2005.

ISO/IEC. Systems and software engineering – measurement process. standard 15939:2007, International Organization for Standardization, Geneva, Switzerland, 2007a.

ISO/IEC. Software engineering – metamodel for development methodologies. standard 24744:2007, International Organization for Standardization, Geneva, Switzerland, 2007b.

ISO/IEC. Systems and software engineering - software life cycle processes. standard 12207:2008, International Organization for Standardization, Geneva, Switzerland, 2008a.

ISO/IEC. Systems and software engineering - system life cycle processes. standard 15288:2008, International Organization for Standardization, Geneva, Switzerland, 2008b.

ISO/IEC. Systems and software engineering - life cycle management - guidelines for process description. standard 24774:2010, International Organization for Standardization, Geneva, Switzerland, 2010.

ISO/IEC. Software engineering – lifecycle profiles for very small entities (VSEs) – management and engineering guide: Generic profile group: Basic profile. standard 29110:2011, International Organization for Standardization, Geneva, Switzerland, 2011.

Jacobson, I., Ng, P.-W., and Spence, I. Enough of processes – lets do practices. *Journal of Object Technology.*, 6(6):41–67, 2007.

Jacobson, I., Meyer, B., and Soley, R. The SEMAT initiative: A call for action, December 2009. URL `http://www.drdobbs.com/architecture-and-design/the-semat-initiative-a-call-for-action/222001342`.

Jacobson, I., Meyer, B., and Soley, R. Software engineering method and theory – A vision statement, 2010. URL `http://semat.org/wp-content/uploads/2012/03/SEMAT-vision.pdf`.

Jacobson, I., Ng, P.-W., McMahon, P., Spence, I., and Lidman, S. The essence of software engineering: The SEMAT kernel. *Queue*, 10(10):40–51, Oct. 2012. ISSN 1542-7730. doi: 10.1145/2381996.2389616. URL `http://doi.acm.org/10.1145/2381996.2389616`.

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., and Lidman, S. *The Essence of Software Engineering: Applying the SEMAT Kernel.* Addison-Wesley Professional, 1st edition, Jan. 2013. ISBN 0321885953.

Johnson, P., Ekstedt, M., and Jacobson, I. Where's the theory for software engineering? *IEEE Softw.*, 29(5):96, Sept. 2012. ISSN 0740-7459. doi: 10.1109/MS.2012.127. URL `http://dx.doi.org/10.1109/MS.2012.127`.

Kaplanski, P. Description logic as a common software engineering artifacts language. In *Information Technology, 2008. IT 2008. 1st International Conference on*, pages 1–4. IEEE, 2008.

Krdžavac, N., Gašević, D., and Devedžić, V. Model driven engineering of a tableau algorithm for description logics. *Computer Science and Information Systems/ComSIS*, 6(1):23–43, 2009.

Laporte, C., Alexandre, S., and O'Connor, R. A software engineering lifecycle standard for very small enterprises. In *Software Process Improvement*, pages 129–141. Springer, 2008.

Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A., Yost, G., and Group, O. M. O. T. P. W. The process interchange format and framework. *Knowl. Eng. Rev.*, 13(1):91–120, Mar. 1998. ISSN 0269-8889. doi: 10.1017/S0269888998001015. URL `http://dx.doi.org/10.1017/S0269888998001015`.

Medina, R. and Oktaba, H. Especificación de esquema informático para métodos y prácticas de KUALI-BEH para facilitar la verificación semi automatizada de sus características y propiedades. Master's thesis, National Autonomous University of Mexico, 2014.

Morales-Trujillo, M. and Oktaba, H. KUALI-BEH software project common concepts. Technical report, Object Management Group, Needham, MA, USA, 2012a.

Morales-Trujillo, M. and Oktaba, H. Identificación y formalización de un núcleo de conceptos comunes para proyectos de desarrollo de software. In *Proceedings of the 15th Iberoamerican Conference on Software Engineering*, CIbSE '12, pages 313–318, 2012b. ISBN 978-1-629939612.

Morales-Trujillo, M. and Oktaba, H. Identification and formalization of software project common concepts: Kuali-beh. In *Memorias del Congreso Internacional de Investigación e Innovación en Ingeniería de Software*, CONISOFT '13. Editorial UV, 2013. ISBN 978-0-615-89523-9.

Morales-Trujillo, M., Oktaba, H., and González, J. Improving software projects inception phase using games - activeaction workshop. In *Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering*, ENASE '14, pages 180–187. SCITEPRESS – Science and Technology Publications, 2014. ISBN 978-989-758-030-7.

Morales-Trujillo, M., Oktaba, H., and Piattini, M. The making of an omg standard. *Computer Satandards & Interfaces*, 2015a. ISSN 0920-5489. doi: http://dx.doi.org/10.1016/j.csi.2015.05.001.

Morales-Trujillo, M., Oktaba, H., and Piattini, M. Using technical-action-research to validate a framework for authoring software engineering methods. In *Proceedings of the 17th International Conference on Enterprise Information Systems*, ICEIS '15, pages 15–27. SCITEPRESS – Science and Technology Publications, 2015b. ISBN 978-989-758-097-0.

Naur, P. and Randell, B. *Software Engineering*. NATO, 1969. URL `http://books.google.es/books?id=mjh1NAEACAAJ`.

NIST. Integration definition for function modelling. Technical report, Gaithersburg, MD, USA, 1993.

Oktaba, H. and Dávila, M. A proposal of a software development project kernel. In *Latin American Software Engineering Symposium*. CIDENET, 2011.

Oktaba, H. and Piattini, M. *Software process improvement for small and medium enterprises: Techniques and case studies*. 2008.

OMG. Object management group. `http://www.omg.org/`.

OMG. The OMG hitchhicker's guide: A handbook for the OMG technology adoption process. Technical report, Object Management Group, Needham, MA, USA, 2008a.

OMG. Software and systems process engineering metamodel (SPEM). Technical report, Object Management Group, Needham, MA, USA, 2008b.

OMG. Case management process modeling (CMPM) RFP. Technical report, Object Management Group, Needham, MA, USA, 2009.

OMG. Business process model and notation (BPMN) 2.0. Technical report, Object Management Group, Needham, MA, USA, 2011a.

OMG. A foundation for the agile creation and enactment of software engineering methods (FACESEM) RFP. Technical report, Object Management Group, Needham, MA, USA, 2011b.

OMG. Meta object facility (MOF) core specification. Technical report, Object Management Group, Needham, MA, USA, 2011c.

OMG. Unified modeling language (UML) infrastructure. Technical report, Object Management Group, Needham, MA, USA, 2011d.

OMG. Architecture-driven modernization (ADM): Software metrics meta-model (SMM). Technical report, Object Management Group, Needham, MA, USA, 2012.

OMG. ESSENCE - kernel and language for software engineering methods (revised submission). Technical report, Object Management Group, Needham, MA, USA, 2013.

OMG. ESSENCE - kernel and language for software engineering methods. Technical report, Object Management Group, Needham, MA, USA, 2014.

Osterweil, L. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. ISBN 0-89791-216-0. URL `http://dl.acm.org/citation.cfm?id=41765.41766`.

OUP. *Oxford Concise Oxford English Dictionary.* Oxford University Press, 11th edition, 2008. ISBN 978-0199548415.

Pardo, C., Pino, F., García, F., Piattini, M., and Baldassarre, T. An ontology for the harmonization of multiple standards and models. *Comput. Stand. Interfaces*, 34(1):48–59, Jan. 2012. ISSN 0920-5489. doi: 10.1016/j.csi.2011.05.005. URL `http://dx.doi.org/10.1016/j.csi.2011.05.005`.

Pease, A. and Carrico, T. Object model working group (OMWG) core plan representation - request for comment. 1997.

Perry, D., Porter, A., and Votta, L. Empirical studies of software engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 345–355, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. doi: 10.1145/336512.336586. URL `http://doi.acm.org/10.1145/336512.336586`.

Pino, F., Pardo, C., García, F., and Piattini, M. Assessment methodology for software process improvement in small organizations. *Information and Software Technology*, 52(10):1044 –

1061, 2010. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2010.04.004. URL `http://www.sciencedirect.com/science/article/pii/S0950584910000777`.

Pino, F. J., García, F., and Piattini, M. Software process improvement in small and medium software enterprises: A systematic review. *Software Quality Control*, 16(2):237–261, June 2008. ISSN 0963-9314. doi: 10.1007/s11219-007-9038-z. URL `http://dx.doi.org/10.1007/s11219-007-9038-z`.

PLG. *Thesaurus Roget's 21st Century*. Philip Lief Group, 3rd edition, 2011. ISBN 978-0-440-24269-7.

PMI. *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*. Project Management Institute, 2004. ISBN 193069945X, 9781933890517.

Poppendieck, M. and Poppendieck, T. *Lean Software Development: An Agile Toolkit*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 0321150783.

Radice, R., Harding, J., Munnis, P., and Phillips, R. A programming process study. *IBM Systems Journal*, 38(2.3):297–307, 1999.

Rout, T. Consistency and conflict in terminology in software engineering standards. In *Software Engineering Standards, International Symposium on*, pages 67–67. IEEE Computer Society, 1999.

Ruiz, F. and Hilera, J. Using ontologies in software engineering and technology. In Calero, C., Ruiz, F., and Piattini, M., editors, *Ontologies for Software Engineering and Software Technology*, pages 62–119. Springer Berlin Heidelberg, 2006.

Runeson, P., Host, M., Rainer, A., and Regnell, B. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, 2012. ISBN 9781118181003.

Schlenoff, C., Knutilla, A., and Ray, S. Unified process specification language: Requirements for modeling process. 1996.

Schmidt-Schaubß, M. and Smolka, G. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, Feb. 1991. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90078-X.

Schneider, K. *Experience and Knowledge Management in Software Engineering*. Springer-Verlag Berlin Heidelberg, 2009. ISBN 978-3-540-95879-6.

Schramm, W. *Notes on Case Studies of Instructional Media Projects [microform] / Wilbur Schramm*. Distributed by ERIC Clearinghouse [Washington, D.C.], 1971. URL `http://www.eric.ed.gov/contentdelivery/servlet/ERICServlet?accno=ED092145`.

Schwaber, K. and Sutherland, J. The scrum guide - the definitive guide to scrum: The rules of the game. `http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011.pdf`, 2011.

SEI. Cmmi® for development, version 1.3. Technical report, Carnegie Mellon University, 2010.

Selfridge, P. G., Hoebel, L. J., and White, D. A. The sixth annual knowledge-based software engineering conference (KBSE-91). *SIGART Bull.*, 3(1):33–35, Jan. 1992. ISSN 0163-5719.

SEMAT. Software engineering method and theory. `http://www.semat.org/`.

Shingo, S. *A study of the Toyota production system: From an Industrial Engineering Viewpoint*. Productivity Press, 1989.

Sousa, K. S., Vanderdonckt, J., Henderson-Sellers, B., and Gonzalez-Perez, C. Evaluating a graphical notation for modelling software development methodologies. *J. Vis. Lang. Comput.*, 23(4):195–212, 2012.

Tapia, A. and Ibargüengoitia, G. Arquitectura de software para el entorno computacional de KUALI-BEH utilizando los métodos del software engineering institute (SEI). Master's thesis, National Autonomous University of Mexico, 2014.

Tate, A. Roots of SPAR - shared planning and activity representation. *The Knowledge Engineering Review*, 13:121–128, 3 1998. ISSN 1469-8005.

Tautz, C. and Wangenheim, C., von. *REFSENO: A Representation Formalism for Software Engineering Ontologies*. IESE-Report / Fraunhofer Einrichtung experimentelles Software Engineering. Fraunhofer-IESE, 1998. URL `http://books.google.es/books?id=NCetPgAACAAJ`.

Urrutia, J., Barrera, R., Rúiz, E., Morales-Trujillo, M., and Oktaba, H. Entorno computacional basado en kuali-beh para apoyar a organizaciones en el desarrollo de software. In *Memorias del Congreso Internacional de Investigación e Innovación en Ingeniería de Software*, CONISOFT '13. Editorial UV, 2013. ISBN 978-0-615-89523-9.

Van Strien, P. Towards a methodology of psychological practice the regulative cycle. *Theory & Psychology*, 7(5):683–700, 1997.

W3C. Web ontology language. standard, World Wide Web Consortium, Cambridge, MA, 2012.

W3C. Resource description framework. standard, World Wide Web Consortium, Cambridge, MA, 2014.

Wang, S., Jin, L., and Jin, C. Represent software process engineering metamodel in description logic. In *Proceedings of World Academy of Science, Engineering and Technology*, volume 11, 2006.

Wang, Y. *Software Engineering Foundations: A Software Science Perspective.* Auerbach Publications, Boston, MA, USA, 1st edition, 2007. ISBN 0849319315, 9780849319310.

Wieringa, R. Empirical research methods for technology validation: Scaling up to practice. *J. Syst. Softw.*, 95:19–31, Sept. 2014. ISSN 0164-1212.

Wieringa, R. and Moralı, A. Technical action research as a validation method in information systems design science. pages 220–238, 2012. doi: 10.1007/978-3-642-29863-9_17. URL `http://dx.doi.org/10.1007/978-3-642-29863-9_17`.

Yin, R. *Case Study Research: Design and Methods.* Applied Social Research Methods. SAGE Publications, 4th edition, 2009. ISBN 978-1-4129-6099-1.

Zapata-Jaramillo, C. Comparing the kernel and language submissions for the SEMAT OMG RFP: ESSENCE and KUALI-BEH. 2012.

Zhang, Y. and Zhang, W. Description logic representation for requirement specification. In Shi, Y., Albada, G., van, Dongarra, J., and Sloot, P., editors, *Computational Science – ICCS 2007*, volume 4488 of *Lecture Notes in Computer Science*, pages 1147–1154. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72585-5.

Part V

Appendixes

Appendix A

KUALI-BEH Extension to ESSENCE – Kernel and Language for Software Engineering Methods

This section defines the KUALI-BEH Extension to the Essence Kernel. KUALI-BEH Extension provides four additional Alphas to allow teams to express their Way-of-Working and the progress of their Work in software projects.

A.1 Overview

KUALI-BEH Extension amplifies the endeavor area of concern adding the following Alphas:

- Practice Authoring as a sub-ordinate Alpha of Way-of-Working
- Method Authoring as a sub-ordinate Alpha of Way-of-Working
- Practice Instance as a sub-ordinate Alpha of Work
- Method Enactment as a sub-ordinate Alpha of Work

The Practice Authoring Alpha allows the practitioners to express work units as practices. These practices can be composed as methods by the Method Authoring Alpha. Practice and Method Authoring Alphas help to articulate explicitly the practitioners' way of working. The way of working defined as practices and/or methods is executed by the organization practitioners and converted into units of Work using the Practice Instance Alpha. As a set, these practice instances define the Method Enactment that can be tracked and its progress checked.

A.2 Practice Authoring

The practice authoring provides a framework for the definition of the practitioners' different ways of working. This knowledge makes up an infrastructure of methods and practices that is defined and applied by practitioners in the organization.

A.2.1 Description

Practice Authoring: It is the defined work guidance, with a specific objective, that advises how to produce a result originated from an entry. The guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result. The completion criteria associated to the result are used to determine if the objective is achieved. Particular competences are required to perform the practice guide activities, which can be carried out optionally using tools. To evaluate the practice performance and the objectives' achievement, selected measures can be associated to it. Measures are estimated and collected during the practice execution.

A.2.1.1 Super-Ordinate Alpha

Way-of-Working

A.2.1.2 Other related Alphas

Method Authoring

A.2.2 Justification: Why Practice Authoring

Software Engineering practitioners have an implicit way of working which is constantly improving. By authoring individual practices they can express them explicitly. Even practitioners with the simplest way of working follow tacit practices.

A.2.3 Alpha states and its associations

The set of states defined for Practice Authoring Alpha are described in Table A-1, while its associations are defined in Table A-2.

Table A-1: States of Practice Authoring Alpha.

Name	Definition
Identified	The way of working to be authored as a practice is identified by the practitioners.
Expressed	The way of working is expressed as a practice using the practice template.
Agreed	The practice is agreed on by the practitioners.
In-Use	The practice is used in software projects by the practitioners as their way of working.
In-Optimization	The practice is adapted and/or improved by the practitioners based on their experience, knowledge and external influence.
Consolidated	The practice is mature and adopted by the practitioners as a routine way of working.

Table A-2: Associations of Practice Authoring Alpha with other Alphas.

Association (to Alpha)	Description
expresses (Way-of-Working)	The Practice Authoring lets practitioners express their way of working.
composes (Method Authoring)	The authored practices can be composed as a method.

A.2.4 Progressing the Practice Authoring

Practice Authoring undergoes a number of states. As indicated in Figure A-1, these states focus on the progression of a way of working while it is being integrated as a practice.

To assess the state and progress of Practice Authoring a checklist is provided in Table A-3.

A.2.5 How Practice Authoring defines the Way-of-Working

In order to define their way of working, the practitioners have to identify the desired objective and the way to produce a result originated from an entry. The result should accomplish laid down completion criteria evaluated by the practitioner's judgment. With the aim to evaluate the

Table A-3: Checklist for Practice Authoring Alpha.

State	Checklist
Identified	• The practitioners have recognized the need to express their tacit way of working as an explicit work unit. • The practitioners have defined the work unit scope to be authored as a practice.
Expressed	• Each of the way of working elements has been identified and mapped to the practice template elements. • The way of working elements have been documented in the practice template.
Agreed	• The expressed practice has been revised and accustomed by practitioners. • The expressed practice has been accepted by the practitioners as their explicit way of working.
In-Use	• The agreed practice has been applied by practitioners in software projects.
In-Optimization	• The in-use practice has been modified by practitioners based on the experience of use and/or the new knowledge acquired.
Consolidated	• The optimized practice has been regularly used by practitioners. • The optimized practice has been stabilized and does not suffer frequent changes.

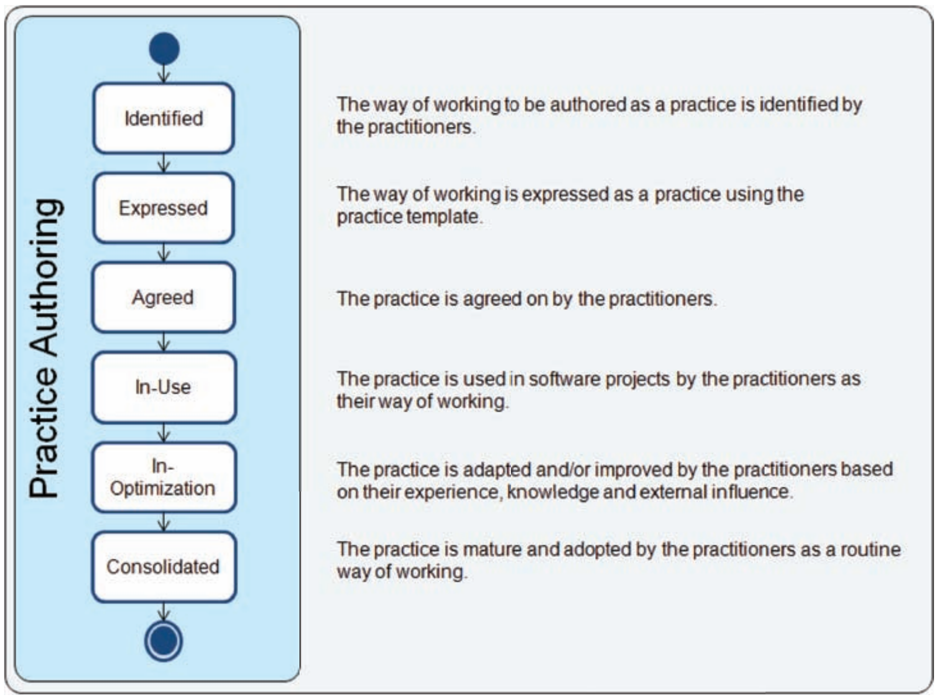

Practice Authoring
Identified
Expressed
Agreed
In-Use
In-Optimization
Consolidated
The way of working to be authored as a practice is identified by the practitioners.
The way of working is expressed as a practice using the practice template.
The practice is agreed on by the practitioners.
The practice is used in software projects by the practitioners as their way of working.
The practice is adapted and/or improved by the practitioners based on their experience, knowledge and external influence.
The practice is mature and adopted by the practitioners as a routine way of working.

Figure A-1: Practice Authoring Alpha and its states.

practice performance, measures to be collected during the execution of the practice are defined. The entries and results can be represented as work products, such as documents, diagrams or code, as conditions, such as particular situations, for example the stakeholder's availability to be interviewed or as Alpha states. Each practice contains work guide, that is, a set of activities that transform entries into results. In addition, the activities are broken down into particular tasks. The guide activities can be carried out using particular tools. Applying the guide in a proper way requires specific competences of the practitioners involved in the software project. As a whole, a set of practices can be comprised as a method that produces an expected software product responding to particular stakeholder needs and under specific conditions.

A.3 Method Authoring

The method authoring provides a framework for the definition of the practitioners' different ways of working using the authored practices to compose it. This knowledge makes up an infrastructure of methods and practices that can be defined and applied by practitioners of the organization in software project endeavors.

A.3.1 Description

Method Authoring: A method is an articulation of a coherent, consistent and sufficient set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.

A.3.1.1 Super-Ordinate Alpha

Way-of-Working

A.3.1.2 Other related Alpha

Practice Authoring

A.3.2 Justification: Why Method Authoring

Software Engineering practitioners have an implicit way of working to accomplish their different types of endeavors. By authoring methods they can express them explicitly. Even practitioners with the simplest way of working follow tacit methods as a composition of agreed practices.

A.3.3 Alpha states and its associations

The set of states defined for Practice Authoring Alpha are described in Table A-2, while its associations are defined in Table A-5.

Table A-4: States of Method Authoring Alpha.

Name	Definition
Identified	Individual practices, needed to accomplish an endeavor, to be authored as a method are selected by the practitioners.
Integrated	The method is integrated as a composition of agreed practices.
Well-Formed	The method is agreed on by the practitioners and accomplishes the properties of coherency, consistency and sufficiency.
In-Use	The method is used in software projects by the practitioners.
In-Optimization	The method is adapted and/or improved by the practitioners based on their experience and external influence.
Consolidated	The method is mature and adopted by practitioners as a routine way of working.

Table A-5: Associations of Method Authoring Alpha with other Alphas.

Association (to Alpha)	Description
defines (Way-of-Working)	The progress of the Method Authoring defines the maturity of the practitioners' way of working.
composes (Method Authoring)	The authored practices can be composed as a method.

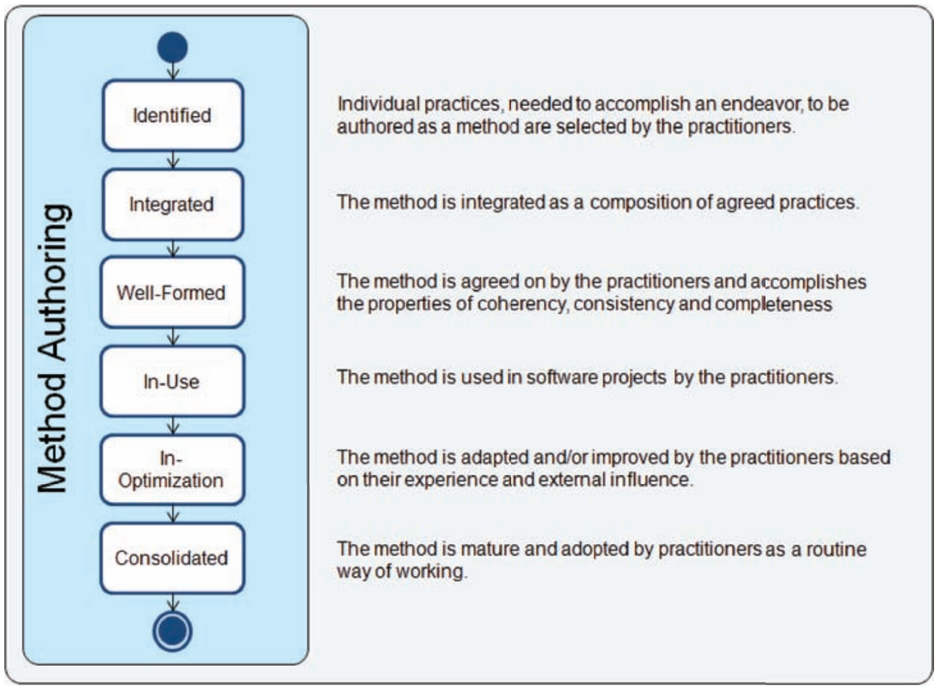

Method Authoring
Identified
Integrated
Well-Formed
In-Use
In-Optimization
Consolidated
Individual practices, needed to accomplish an endeavor, to be authored as a method are selected by the practitioners.
The method is integrated as a composition of agreed practices.
The method is agreed on by the practitioners and accomplishes the properties of coherency, consistency and completeness
The method is used in software projects by the practitioners.
The method is adapted and/or improved by the practitioners based on their experience and external influence.
The method is mature and adopted by practitioners as a routine way of working.

Figure A-2: Method Authoring Alpha and its states.

A.3.4 Progressing the Method Authoring

Method Authoring undergoes a number of states. As indicated in Figure A-2, these states are selected, integrated, well formed, in use, in optimization and consolidated. These states show the progression of the method's maturity and stability, from the initial integration till the routine use by the practitioners.

A method is ready to be used in software projects when its definition reaches the well formed state. It means that its set of practices should preserve the properties of coherency, consistency and sufficiency to allow the achievement of a method purpose.

To assess the state and progress of Method Authoring a checklist is provided in Table A-6.

A.3.5 How Method Authoring defines the Way-of-Working

In order to form a method, practitioners have to define its purpose, considering the specific stakeholder needs and the desired characteristics of the software product. In Software Engineering context, a method pursues a purpose related to developing, maintaining or integrating a software product. The set of practices that makes up a method should contribute directly to the achievement of this purpose.

A.4 Practice Instance

A.4.1 Description

Practice Instance: During the enactment of a method by practitioners, each practice is initially instantiated as work to be done. Later it changes its state to can start, in execution, stand by or in verification until it is finished or cancelled.

A.4.1.1 Super-Ordinate Alpha

Work

A.4.1.2 Other related Alpha

Method Enactment

A.4.2 Justification: Why Practice Instance

Practitioners execute work units in order to achieve a specific objective. This work, even the simplest, is tracked and practitioners monitor its progress and verify its completion. Also, the temporal suspension or cancellation of the work corresponds to the everyday practitioner's experience.

A.4.3 Checking the progress of Practice Instance states

To assess the state and progress of Practice Instance a checklist is provided in Table A-7.

Table A-6: Checklist for Method Authoring Alpha.

State	Checklist
Identified	• The practitioners have recognized the need to interrelate their individual agreed practices to accomplish software projects. • The practitioners have defined the purpose, entry and result of the method in the template. • The practitioners have identified the agreed practices to be integrated as a method.
Integrated	• Each of the selected agreed practices have been added to the method template.
Well-Formed	• The integrated method has accomplished the coherence, consistency and sufficiency properties. • The integrated method has been revised and customized by practitioners. • The integrated method has been accepted by the practitioners as their explicit way of working.
In-Use	• The well-formed method is applied in software projects by practitioners.
In-Optimization	• The in-use method has been modified by practitioners based on the experience of use and/or the new knowledge acquired.
Consolidated	• The optimized method has been used by practitioners regularly. • The optimized method has been stabilized and does not suffer frequent changes.

Table A-7: Checklist for Practice Instance Alpha.

State	Checklist
Instantiated	• The practitioners have identified the work to be done. • The needed work unit has been created as the practice instance. • The practice instance measures have been optionally estimated by practitioners.
Can-Start	• The required practice instance entry has been created and assigned. • The practice instance measures have been estimated.
In-Execution	• The practitioners have chosen a practice instance that can start. • The practitioners responsible for the practice instance have been agreed upon • The practitioners are working on the practice instance following the guide.
Stand-By	• The execution of the practice instance has been interrupted. • The practitioners have paused any work related to the practice instance.
In-Verification	• The practitioners have produced a result after executing the practice instance. • The practitioners are verifying the result using the related completion criteria.
Cancelled	• The practitioners have stopped permanently the practice instance work. • The associated items of the practice instances have been quit.
Finished	• The practitioners have finalized the practice instance work. • The practitioners have produced a result, which was verified as correct.

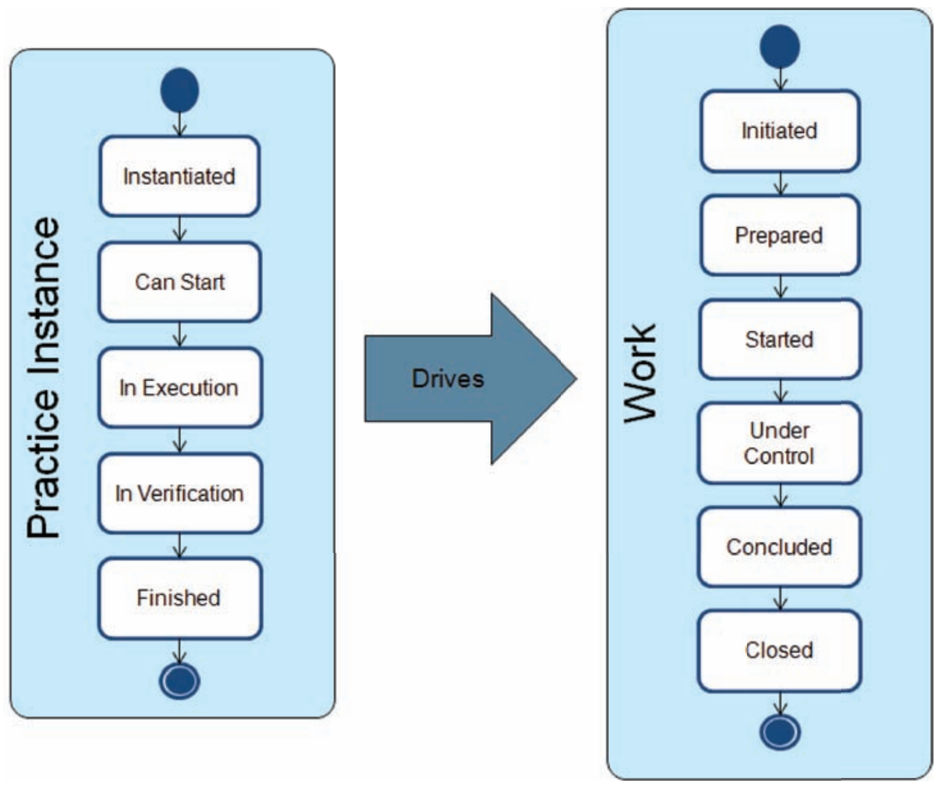


Figure A-3: Practice Instance drives the progress of the Work.

A.4.4 How Practice Instance drives the Work

The Work is driven by Practice Instance as shown in Figure A-3.

A.5 Method Enactment

A.5.1 Description

Method Enactment: It occurs in the context of a software project execution. Before starting the method enactment, the practitioners assigned to the software project get to know the stakeholder needs and are informed about the software project conditions. In case of a maintenance or software integration project, the already existent software product(s) should also be available.

A.5.1.1 Super-Ordinate Alpha

Work

A.5.1.2 Other related Alpha

Practice Instance

A.5.2 Justification: Why Method Enactment

Practitioners execute software projects following a set of practices (method) in order to achieve a specific purpose. This work, even the simplest, is tracked and its progress is monitored by practitioners.

A.5.3 Checking the progress of a Method Enactment

To assess the state and progress of Method Enactment a checklist is provided in Table A-8.

A.5.4 How Method Enactment drives the Work

The Work is driven by Method Enactment as shown in Figure A-4.

Table A-8: Checklist for Method Enactment Alpha.

State	Checklist
Selected	• The practitioners have selected a well-formed method from the methods and practices infrastructure. • The practitioners have fulfilled the required competencies specified in the method practices guides.
Adapted	• The practitioners have analyzed the stakeholder needs and conditions of the software project. • The practitioners have adapted the selected method. • Each of the practices of the method has been instantiated as work units planned to be executed during the software project.
Ready-to-Begin	• The method has at least one practice instance in Can Start state. • The method and the practitioners are ready to begin the work.
In-Progress	• The practitioners are applying the method. • The practitioners have paused any work related to the practice instance.
Progress-Snapshot	• The practitioners are analyzing the method execution context. • The practitioners are discussing and taking decisions about the work continuation as it was planned or if the method requires an adaptation.
Cancelled	• The practitioners have stopped permanently the method execution. • The associated items of the method have been quit. • The result has not been produced.
Finished	• The practitioners have finalized their work. • The practitioners have produced a result that can be delivered.

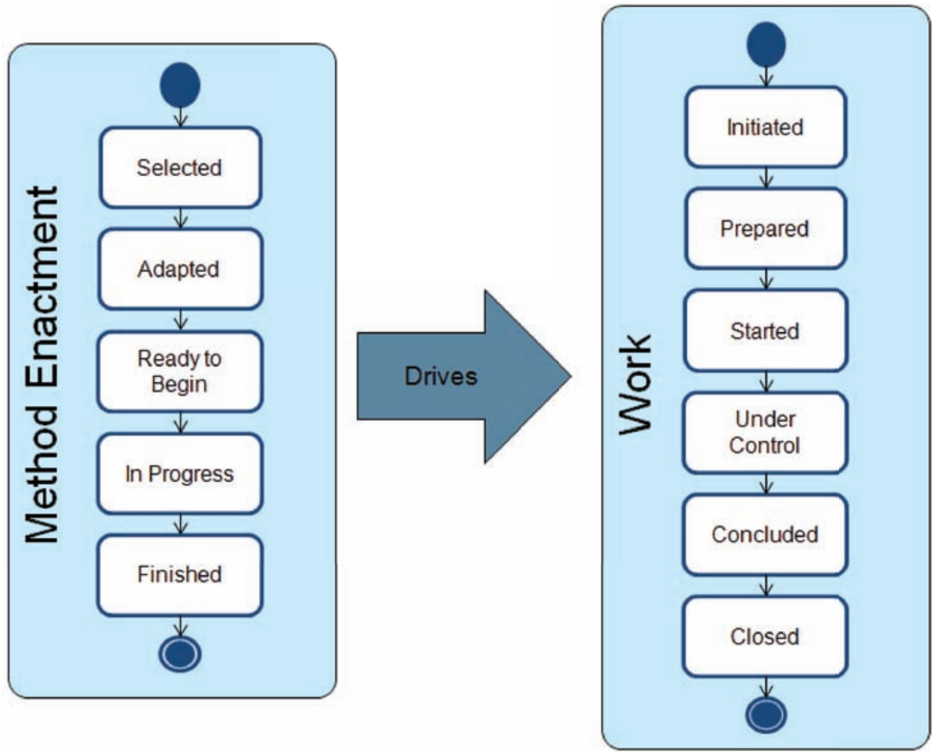


Figure A-4: Method Enactment drives the progress of the Work.

Appendix B

Sources and terms considered for KUALI-BEH common concepts definitions

B.1 Activity

Activity (OUP, 2008) –

1. A condition in which things are happening or being done.
2. An action taken in pursuit of an objective.

Activity (PLG, 2011) – State of being active.

Activity (ISO/IEC, 2011) – A set of cohesive tasks. Task is a requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more objectives of a process. A process activity is the first level of process workflow decomposition and the second one is a task.

Activity (ISO/IEC, 2008a) – Set of cohesive tasks of a process.

Activity (OMG, 2011b) – An activity is a set of cohesive tasks intended to contribute to the achievement of one or more objectives. An activity is the first level of method workflow decomposition and the second one is a task.

Activity (IEEE, 2000) – A defined body of work to be performed, including its required input and output information.

B.2 Coherent

Coherent (OUP, 2008) –

1. (of an argument or theory) logical and consistent.
2. Holding together to form a whole.

Coherent (PLG, 2011) – Understandable.

B.3 Condition

Condition (OUP, 2008) –

1. The state of something or someone, with regard to appearance, fitness, or working order.
2. Circumstances affecting the functioning or existence of something.
3. A state of affairs that must exist before something else is possible.

Condition (PLG, 2011) – Circumstances, state, status, action.

B.4 Consistent

Consistent (OUP, 2008) –

1. Acting or done in the same way over time, especially so as to be fair or accurate.
2. (usu. consistent with) compatible or in agreement.
3. Not containing any logical contradictions.

Consistent (PLG, 2011) – Constant, regular.
Consistency (IEEE, 2000) – The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.

Consistency check (IEEE, 2000) – A check that verifies that an item of data is compatible with certain rules specified for that data.

B.5 Guide

Guide (OUP, 2008) – A directing principle or standard.
Guide (PLG, 2011) – Paradigm, pattern, advice.

B.6 Input

Input (IEEE, 2000) –

1. Pertaining to data received from an external source.
2. The data to be processed.

Input (IEEE, 1998b) – In an IDEF0 model, that which is transformed by a function into output.

B.7 Knowledge and Skills

Knowledge (OUP, 2008) –

1. Information and skills acquired through experience or education.
2. Awareness or familiarity gained by experience.

Knowledge (PLG, 2011) – Person's understanding; information, ability, attainments.
Skill (OUP, 2008) –

1. The ability to do something well; expertise or dexterity.
2. Train (a worker) to do a particular task.

Skill (PLG, 2011) – Ability, talent to do something, competence.

B.8 Measure

Measure (ISO/IEC, 2005) – Measures can be used to assess or to quantitatively estimate various aspects. Measures can be derived from the attributes of the software, the maintenance process, and personnel, including size, complexity, quality, understandability, maintainability, and effort.

Measure (IEEE, 2005) – The number or symbol assigned to an entity by a mapping from the empirical world to the formal, relational world in order to characterize an attribute.

Measure (OUP, 2008) –

1. A unit or standard of measurement.
2. A system of measurement.

Measure (PLG, 2011) – Magnitude, dimension.

Measure (ISO/IEC, 2007a) – Variable to which a value is assigned as the result of measurement.

B.9 Method

Method (OUP, 2008) –

1. A particular procedure for accomplishing or approaching something.
2. Orderliness of thought or behavior.

Method (PLG, 2011) – Means, procedure.

Method (OMG, 2011b) – A method is a systematic way of doing things in a particular discipline. Software engineering methods support tasks such as the development of a new software system, the maintenance of an existing system or even the integration of an entire enterprise system architecture. Methods at this level may be considered as composed from well-defined practices. A method may be considered to be simply a composite practice targeted at the level of support of an entire discipline.

Method (IEEE, 1998a) – A formal, well-documented approach for accomplishing a task, activity, or process step governed by decision rules to provide a description of the form or representation of the outputs.

Methodology (IEEE, 1995) – A comprehensive, integrated series of techniques or methods creating a general systems theory of how a class of thought-intensive work ought to be performed.
Process (ISO, 2005) – Set of interrelated or interacting activities which transforms inputs into outputs.
Purpose statement (IEEE, 1998b) – A brief statement of the reason for an IDEF0 model's existence that is presented in the A-0 context diagram of the model.

B.10 Methods and Practices Infrastructure

Practice Infrastructure (OMG, 2011b) – A practice infrastructure would enable software developers to more quickly understand, compose and compare individual practices and entire methods. It could also form the basis for the appropriate governance of software organizations, while allowing their developers the freedom to use their preferred practices, composed with those of their organizations. Further, it would allow the evaluation and validation of comparable method and process elements, guide practical research to useful results and act as a common context for training and education.

B.11 Pattern

Pattern (Gamma et al., 1995) – A design pattern describes the problem, a solution to the problem consisting of a general arrangement of objects and classes, when to apply the solution, and the consequences of applying the solution.
Pattern (Alexander, 1979) – Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

B.12 Practice

Practice (OUP, 2008) –

1. The actual application or use of a plan or method, as opposed to the theories relating to it.

2. The customary or expected procedure or way of doing something.

Practice (PLG, 2011) – Routine, usual procedure.

Practice (OMG, 2011b) – A practice is a general, repeatable approach to doing something with a specific purpose in mind, providing a systematic and verifiable way of addressing a particular aspect of the work at hand. It should have a clear goal expressed in terms of the results its use will achieve and provide guidance on what is to be done to achieve the goal and to verify that it has been achieved. Such practices may include specific approaches for software design, coding, testing at various levels, integration, organizing and managing the development team.

Practice (IEEE, 2000) – Recommended approach, employed to prescribe a disciplined, uniform approach to the software life cycle.

Objectives (IEEE, 2000) – The desired goals and results of the evaluation/selection process in terms relevant to the organization(s) involved.

B.13 Practitioner

Practitioner (OUP, 2008) – A person actively engaged in an art, discipline, or profession, especially medicine.

Practitioner (PLG, 2011) – Professional, expert, specialist.

Judgment (OUP, 2008) – The ability to make considered decisions or form sensible opinions.

Judgment (PLG, 2011) – Discernment, experience, perception.

B.14 Project Conditions

Condition (OUP, 2008) –

1. The state of something or someone, with regard to appearance, fitness, or working order.

2. Circumstances affecting the functioning or existence of something.

3. A state of affairs that must exist before something else is possible.

Condition (PLG, 2011) – Circumstances, state, status, action.

B.15 Result

Output (IEEE, 2000) – Data that have been processed.

B.16 Similar

Similar (OUP, 2008) – Of the same kind in appearance, character, or quantity, without being identical.
Similar (PLG, 2011) – Analogous, coincident, congruent, matching.

B.17 Software Product

Software product (ISO/IEC, 2008a) – Set of computer programs, procedures, and possibly associated documentation and data.

B.18 Software Project

Project (PMI, 2004) – A temporary endeavor undertaken to create a unique product, service, or result.
Project (ISO/IEC, 2008a) – Endeavour with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements.

B.19 Stakeholder

Stakeholder (ISO/IEC, 2008a) – Is an individual or organization having a right, share, claim or interest in a system or in its possession of characteristics that meet their needs and expectations.

B.20 Stakeholder Needs

Need (OUP, 2008) –

1. Circumstances in which something is necessary; necessity.

2. A thing that is wanted or required.

Need (PLG, 2011) – Want, requirement, requisite, demand, exigency.

B.21 Sufficient

Complete (OUP, 2008) –

1. Having all the necessary or appropriate parts; entire.
2. Having run its full course; finished.

Complete (PLG, 2011) – Total, not lacking.

NOTE: Initially this property was named Complete, however, during the OMG standardization process it was changed to Sufficient, while the definition of the property remained unchanged.

B.22 Task

Task (OUP, 2008) – A piece of work.
Task (PLG, 2011) – Job or chore, often assigned.
Task (OMG, 2011b) – Task is a required, recommended or permitted action.
Task (ISO/IEC, 2008a) – Requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more outcomes of a process.
Task (IEEE, 2000) – The smallest unit of work subject to management accountability. A task is a well-defined work assignment for one or more project members. Related tasks are usually grouped to form activities.

B.23 Tool

Tool (OUP, 2008) – A device or implement, typically hand-held, used to carry out a particular function.
Tool (PLG, 2011) – Device, apparatus, instrument.

B.24 Verification Criteria

Acceptance criteria (IEEE, 2000) – The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity.

Verification (ISO, 2005) – Confirmation, through the provision of objective evidence, that specified requirements have been fulfilled.

Verification (ISO/IEC, 2008a) – Verification in a life cycle context is a set of activities that compares a product of the life cycle against the required characteristics for that product. This may include, but is not limited to, specified requirements, design description and the system itself.

Criterion (OUP, 2008) – A standard by which something can be judged or decided.

Criterion (PLG, 2011) – Test, gauge for judgment.

Criteria (IEEE, 2000) – Parameters against which the CASE tool is evaluated, and upon which selection decisions are made.

B.25 Work Product

Work product (IEEE, 2000) – Any tangible item produced during the process of developing or modifying software.

Input Products (ISO/IEC, 2011) – Products required to perform the process and its corresponding source, which can be another process or an external entity to the project.

Output Products (ISO/IEC, 2011) – Products generated by the process and its corresponding destination, which can be another process or an external entity to the project.

Internal Products (ISO/IEC, 2011) – Products generated and consumed by the process.

Product (ISO, 2005) – Result of a process.

B.26 Work Team

None

Appendix C

Issues to be discussed

Why KUALI-BEH fits in Agile Creation and Enactment of Software Engineering Methods:

- Practitioners can start defining individual useful practices and then combine them in methods (coherent, consistent and sufficient set of practices). The traditional approach is to begin with difficult-to-integrate processes, while the "agile" approach is to collect several practices (advises or techniques) not necessarily consistent or sufficient.
- Method improvement can be done "offline" through modifying the organization's Methods and Practices Infrastructure, or "online" applying method adaptation during its enactment. We think that the online adaptation adds real agility to the software project execution.
- Our proposal empowers the work team, since the main decisions on what to do, how to do it, who will do it, effort estimations, etc. are in their hands. So we tried to follow the first principle of the Agile Manifesto "individuals and interactions over processes and tools".

C.1 Alternative naming issue

Our proposal does not use the "kernel" as a key word. We prefer to talk about *software project common concepts* because it is more understandable for Software Engineering practitioners, as demonstrated in chapters 7 and 8.

The following words or expressions can be considered as alternatives for software project common concepts of KUALI-BEH:

- Sufficient – complete
- Guide – guidance
- Input – entry
- Knowledge and Skills – competences
- Method – process – methodology
- Methods and Practices Infrastructure – organization's base of knowledge
- Practice – technique – work unit
- Practitioner – software engineer
- Project conditions – project constrains
- Software Product – software system
- Stakeholder – customer
- Stakeholder needs – customer needs – customer requirements – customer value
- Work Product – artifact

C.2 SPEM issue

We do not use SPEM 2.0 to define KUALI-BEH framework because we want to simplify the proposal as much as possible, in order to make it clear for the practitioners from the first approach and get their acceptance. KUALI-BEH at this point is not orthogonal or opposite to SPEM 2.0.

A deeper analysis shows that the difference between *process* and *method* in SPEM 2.0 is not clear. We agree with SPEM 2.0 proposal on *activity*, *guide*, *work product*, *tool* or *role* level of abstraction, however more abstract concepts are not easy to understand and some differences between them can be identified. Table C-1 presents a likely mapping between KUALI-BEH common concepts and SPEM 2.0 elements. The main differences between concepts, if exist, are shown in the additional column.

Table C-1: KUALI-BEH and SPEM 2.0 concepts.

SPEM 2.0	**KUALI-BEH**	**Differences identified**
Activity	Activity	-
Artifact	Work Product	-
Deliverable	Result	Not all the results are deliverable, but all the deliverables can be results. The use of the term Result makes simpler the proposal.
Guidance	Guide	-
Method Library	MPI	The Methods and Practices Infrastructure can contain the Method Library and more elements.
Metric	Measures	-
Milestone	Objective/Purpose	The usage of two terms, instead of one, to define a goal, expects to make a difference between method and practice concepts.
Outcome	Result	-
Role Definition	Knowledge and Skills	Define the knowledge and skills required to perform a guide, gives flexibility to the organization to organize their human resources as roles or something else.
Step	Task	Both concepts define the smallest action done by a practitioner, is a naming difference only.
Task Definition	Practice	Both concepts are similar in level of abstraction, but the practice concept is fundamental in the RFP.
Tool Definition	Tool	-

This proposal intends to be a simple standard that supports the majority of existent methods and practices in-use in the industry nowadays. For that reason we have tried to preserve a minimal core that will support them. As shown in this thesis, using KUALI-BEH permitted to model existing ISO/IEC-style and Agile-style practices and methods; moreover, it remains concordant with SPEM 2.0, making it possible to reuse SPEM 2.0 metamodel.

C.3 MOF issue

Knowledge can be represented on different levels of abstraction. For the purpose of this proposal three knowledge levels were used, the epistemological level, the conceptual level, and the linguistic level.

According to (Tautz and von Wangenheim, 1998) the knowledge levels mentioned above are defined as follows:

- The epistemological level defines the epistemistic primitives such as concepts, attributes, relationships, etc. Thus, the epistemological level is domain-independent.

- The conceptual level defines the standard vocabulary. It is domain-specific. Exemplary constructs of this level (for the software engineering domain) are process models, measurement plans, code modules, lessons learned, etc. As an explicit specification of a conceptualization, ontology is always defined on this level. Thus, ontology can be defined using epistemistic primitives.

- Finally, the linguistic level defines concrete instances of the constructs defined on the conceptual level. It is domain- and context-specific. An exemplary construct on this level (for a particular software development organization) is a concrete measurement plan for measuring the effort of project X at company Y.

On one hand, REFSENO makes possible the representation of this kind of knowledge, formalizing it as ontology. REFSENO is a framework to conceptualize knowledge.

On the other hand, Meta-Object Facility (MOF) (OMG, 2011c) is a model to create models. It provides a metadata management framework, and a set of metadata services to enable the development and interoperability of model and metadata driven systems.

Taking into account that the purpose of this proposal is to conceptualize a specific domain, identifying its concepts and relationships, the submission team decided to develop an ontology instead of a metamodel.

Nevertheless, the KUALI-BEH ontology can be mapped to the levels M1 and M2 of MOF. Table C-2 presents the mapping between MOF layers and the KUALI-BEH ontology.

Table C-2: MOF layers and KUALI-BEH ontology.

Layer	**MOF**	**KB Language**
M3	Meta-meta-model	-
M2	Meta-model	UML Class diagram
M1	Model	Glossary of concepts, Relationships and Attributes
M0	Data	Practitioners applying KUALI-BEH to describe their way of working

Appendix D

Acronyms

This section contains the list of acronyms used in this thesis.

AB	Architectural Board
ABox	Assertional Knowledge
ADTF	Analysis and Design Task Force
AESIG	Architecture Ecosystem Special Interest Group
ALPHA	Abstract-Level Project Health Attributes
BC	Business Committee
BoD	Board of Directors
BPMN	Business Process Modeling Notation
CAME	Computer-Aided Methods Engineering
CIbSE	Iberoamerican Conference on Software Engineering
CMMI	Capability Maturity Model Integration
CMPM	Case Management Process Modeling
CONACyT	Consejo Nacional de Ciencia y Tecnología

CONISOFT	Congreso Internacional de Investigación e Innovación en Ingeniería de Software
CORBA	Common Object Request Broker Architecture
CPR	Core Plan Presentation
CS	Case Study
DL	Description Logics
DT	Development Team
ENASE	International Conference on Evaluation of Novel Approaches to Software Engineering
EPF	Eclipse Process Framework
ETVX	Entry-Task-Validation-Exit
FACESEM	A Foundation for the Agile Creation and Enactment of Software Engineering Methods
FEDER	Fondo Europeo de Desarrollo Regional
FOL	First Order Logic
FTF	Finalization Task Force
IDEF0	Integration Definition for Function Modelling
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ITSSLP	Instituto Tecnológico Superior de San Luis Potosí
JSF	Java Server Faces
JTC1	Joint Technical Committee 1 of the ISO/IEC
KB-A	KUALI-BEH Algebra

KB-K	KUALI-BEH Knowledge
KB-O	KUALI-BEH Ontology
KBMetMan	KUALI-BEH Method Manager Tool
KBOView	KUALI-BEH Operational View Tool
KBProjMan	KUALI-BEH Project Manager Tool
KBSE	Knowledge-Based Software Engineering
KBSView	KUALI-BEH Static View Tool
KBTool	KUALI-BEH Technological Environment
KR	Knowledge Representation
LASES	Latin American Symposium of Software Engineering
LOI	Letter of Intent
MOF	Meta-Object Facility
MPI	Methods and Practices Infrastructure
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standards and Technology
OMG	Object Management Group
OWL	Web Ontology Language
PAEP	Programa de Apoyo a Estudiantes de Posgrado
PIF	Process Interchange Format
PMBOK	Project Management Body of Knowledge
PMI	Project Management Institute

PSL	Process Specification Language
PTC	Platform Technical Committee
RDF	Resource Description Framework
REFSENO	Representation Formalism for Software Engineering Ontologies
RFP	Request for Proposals
RFR	Request for Retirement
RTF	Revision Task Force
SC7	Subcommittee 7 of the ISO/IEC JTC1
SDPK	Software Development Projects Kernel
SEI	Software Engineering Institute
SEMAT	Software Engineering Method and Theory
SEMDM	Software Engineering – Metamodel for Development Methodologies
SMM	Structured Metrics Metamodel
SPAR	Shared Planning and Activity Representation
SPEM	Software and Systems Process Engineering Meta-model
SWEBOK	Software Engineering Body of Knowledge
TAR	Technical-Action-Research
TBox	Terminological Knowledge
TTF	Testing Task Force
UML	Unified Modeling Language
UNAM	Universidad Nacional Autónoma de México

URI	Uniform Resource Identifier
V2R	Voting for Recommendation
V2V	Vote-to-Vote
VSE	Very Small Entity
W3C	World Wide Web Consortium
WT	Work Team
XMI	XML Metadata Interchange
XML	eXtensible Markup Language