Date: August 2012

KUALI-BEH: Software Project Common Concepts

Revised Submission – Version 1.1

In response to: Foundation for the Agile Creation and Enactment of Software Engineering Methods (FACESEM) RFP (OMG Document ad/2011-06-26)

OMG Document Number: ad/2012-02-06

Submission Team

OMG Submitters:

Universidad Nacional Autónoma de México (UNAM)

Supporting Organizations:

Graduate Science and Engineering Computing, National Autonomous University of Mexico (UNAM)
Science Faculty, National Autonomous University of Mexico (UNAM)
General Direction of Computing and Information Technologies and Communication (DGTIC), National Autonomous University of Mexico (UNAM)
Alarcos Research Group, University of Castilla – La Mancha (UCLM)
Magnabyte
JPE Consultores
Ultrasist
Software Gurú

Authors of this proposal:
Hanna J. Oktaba, Miguel Ehécatl Morales Trujillo and Magdalena Dávila Muñoz.

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A Specifications Catalog is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.)

Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text
Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.
`Courier - 10 pt. Bold:` Programming language elements.
Helvetica/Arial - 10 pt: Exceptions

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

1 Scope

KUALI-BEH describes the kernel of common concepts involved in software projects and their relationships. The static view of KUALI-BEH allows the definition of methods and practices that are useful for organizations dedicated to software development, maintenance or integration. The KUALI-BEH operational view describes the method enactment during the execution of a software project.

The KUALI-BEH common concepts are applicable to define methods and practices independently of the size and complexity of the projects, the lifecycle model or the technology used.

Software Engineering practitioners, actively involved in software projects, are the target audience of this document. Also method engineers, in charge of the existent and recommended working definitions, are another group who may benefit from this proposal.

2 Conformance

KUALI-BEH is conformant to the 6.5 Mandatory Requirements of A Foundation for the Agile Creation and Enactment of Software Engineering Methods RFP [1].

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply:

- A Foundation for the Agile Creation and Enactment of Software Engineering Methods [1].

4 Terms and Definitions

For the purposes of this specification, the terms and definitions given in the normative reference and the following apply.

BEH
Mayan word meaning *way*, *course* or *path*.

KUALI
Nahuatl word meaning good, fine or appropriate.

MPI
Methods and Practices Infrastructure.

WT
Work Team.

5 Symbols

There are no specific symbols associated with this specification.

6 Additional Information

6.1 Changes to Adopted OMG Specification

There are no specific changes to adopted OMG specifications.

6.2 How to Read this Specification

Section 7 presents an overview of the KUALI-BEH proposal. Section 8 describes the static view that introduces and defines the kernel of common concepts involved in software projects. An example of the method and practice definitions is provided. The operational view, focusing on the method enactment during the execution of a software project, is presented in section 9. This section also includes an example.

The chapters are organized in a logical manner and can be read sequentially.

6.3 Submitting Organizations

The following organizations submitted this specification:

- Universidad Nacional Autónoma de México (UNAM)

6.4 Supporting Organizations

The following organizations and companies supported this specification:

- Graduate Science and Engineering Computing, National Autonomous University of Mexico (UNAM)
- Science Faculty, National Autonomous University of Mexico (UNAM)
- General Direction of Computing and Information Technologies and Communication (DGTIC), National Autonomous University of Mexico (UNAM)
- Alarcos Research Group, University of Castilla – La Mancha (UCLM)
- Magnabyte
- JPE Consultores
- Ultrasist
- Software Gurú

6.5 Submission Contacts

- Hanna J. Oktaba, UNAM, hanna.oktaba@ciencias.unam.mx
- Miguel Ehécatl Morales Trujillo, UNAM, migmor@ciencias.unam.mx

6.6 Acknowledgements

The following people contributed valuable ideas and feedback that improved this specification: Mario Piattini Velthuis, Francisco Hernández Quiroz, María Guadalupe Ibargüengoitia González, Jorge Barrón Machado, María Teresa Ventura Miranda, Liliana Rangel Cano, Nubia Fernández, María de los Ángeles Sánchez Zarazua, Luis Daniel Barajas González, Sergio Eduardo Muñoz Siller, Elliot Iván Armenta Villegas, María de los Ángeles Ramírez, Miguel Ángel Peralta Martínez and José León González.

In addition these people collaborated in one way or another to this specification: Rodrigo Barrera Hernández, José Luis Urrutia Velázquez, Eraim Ruíz Sánchez, Álvaro Antonio Saldaña Nava, Alberto Tapia, Hugo Rojas Martínez, Evaristo Fernández Perea and Octavio Orozco y Orozco.

6.7 Status of the Document

This document is a revised specification for a further review and comment by OMG members.

6.8 Responses to RFP Requirements

See Annexes A, B, C, D and E.

7 KUALI-BEH Overview

The KUALI-BEH: Software Project Common Concepts has been developed as a proposal responding to the RFP *A Foundation for the Agile Creation and Enactment of Software Engineering Methods*. KUALI-BEH is based on the knowledge obtained from recognized sources and the experience of the definition of software development standards [2] [3] [4].

KUALI-BEH is composed of two views: the static and the operational. The KUALI-BEH static and operational views are the kernel of the software project common concepts.

The static view provides a framework for the definition of the practitioners' different ways of working. These ways of working are arranged as methods composed by practices. This knowledge makes up an infrastructure of methods and practices that can be applied by practitioners.

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

Figure 1 shows the global structure that makes up the KUALI-BEH proposal.

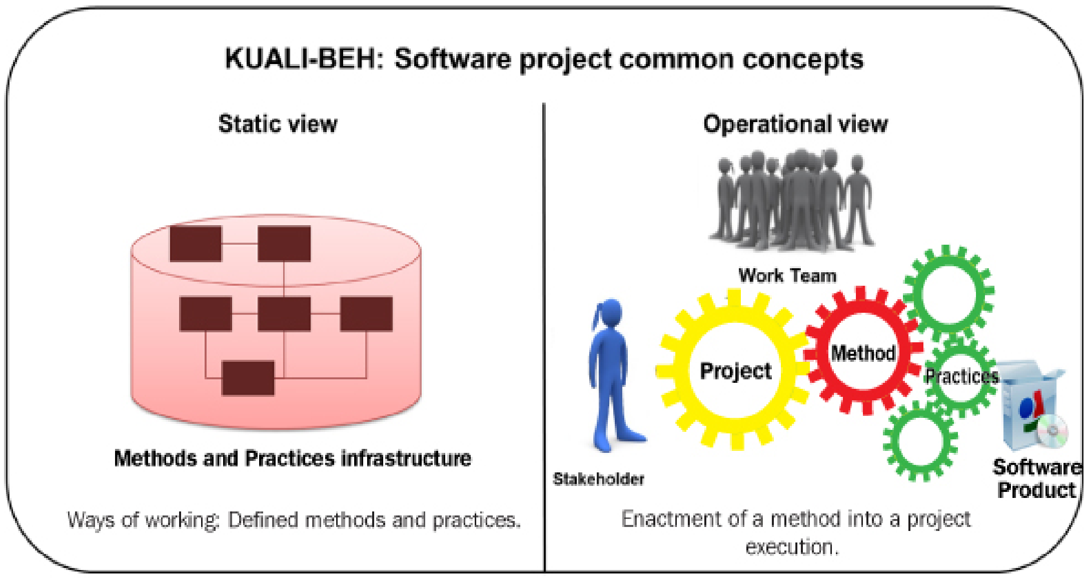

KUALI-BEH: Software project common concepts
Static view
Operational view
Work Team
Methods and Practices infrastructure
Stakeholder
Project
Method
Practices
Software Product
Ways of working: Defined methods and practices.
Enactment of a method into a project execution.

Figure 1 – KUALI-BEH Static and Operational view

8 KUALI-BEH Static View

KUALI-BEH static view describes the software project common concepts. Section 8.1 presents a general outline of this view. The software project common concepts definitions are presented in section 8.2. The concepts templates and graphical representations for practitioners are proposed in sections 8.3 and 8.4 respectively. Finally, section 8.5 shows static view examples.

8.1 Induction to Software Project Common Concepts

KUALI-BEH static view describes common concepts involved in *software projects* and their relationships. These common concepts are written in italics.

A *software project* is an effort of a group of Software Engineering *practitioners* aiming at developing, maintaining or integrating *software products*. Typically a *software project* is originated by the needs of an individual or an organization, a *stakeholder*. The *stakeholder needs* are expressed to a *work team*, composed by *practitioners*, under some restrictions called *project conditions*.

While *work teams* are developing *projects*, they are creating their own ways of working according to their own *knowledge and skills*. These ways of working comprise *practices* and compose different *methods*. Thus, a *practice* is a set of *activities* and *tasks* which has been used repeatedly in *software projects* and has proven its usefulness.

A collection of *practices* can be structured 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 a 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.

Now, let’s review the essential elements required to express the *method* and *practice* concepts.

In order to define a *method*, we have to define its purpose, considering the *stakeholder needs* characteristics and the desired *software product*. In this 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 to the achievement of this purpose.

Each *practice* has the objective to produce a *result* originated from an *input*. The result should accomplish laid down *verification criteria* that are evaluated by the practitioner’s judgment. With the aim of evaluating the performance of a practice, 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, for example the stakeholder’s availability to be interviewed.

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

As a whole, the set of *practices* that comprises a *method* must be coherent, consistent and complete. In other words, a set of *practices* is coherent if the objective of each practice contributes to the entire *method* purpose. It is consistent if each of its *inputs* and *results* are interrelated and useful. Finally, it is complete if the achievement of all *practice* objectives fulfills entirely the *method* purpose and produces expected *software product*.

Figure 17 (section 10) shows the common concepts as a UML class diagram. Each common concept is represented as a class, and their logical connections as relationships.

8.2 Software Project Common Concepts Definitions

The definitions of KUALI-BEH common concepts and other related terms are presented in this section.

8.2.1 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 definitions of the common concept related to software project are presented in the following subsections.

8.2.1.1 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.

8.2.1.2 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.

8.2.1.3 Stakeholder Needs

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

8.2.1.4 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.

8.2.1.5 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.

8.2.1.6 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.

8.2.2 Method Definition

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

8.2.3 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 subsections present the definitions of the common concept related to practice.

8.2.3.1 Input

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

8.2.3.2 Result

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

8.2.3.3 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.

8.2.3.4 Activity

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

8.2.3.5 Task

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

8.2.3.6 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.

8.2.3.7 Work Product

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

8.2.3.8 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.

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

8.2.4 Method Properties

The set of method practices should preserve the properties of coherency, consistency and completeness to allow the achievement of a method purpose.

8.2.4.1 Coherent Set of Practices
A set of method practices is *coherent* if each practice objective contributes to achieve the method purpose.

Figure 2 illustrates a coherent set of practices. Graphical symbol M represents a method and P a practice (see section 8.4)

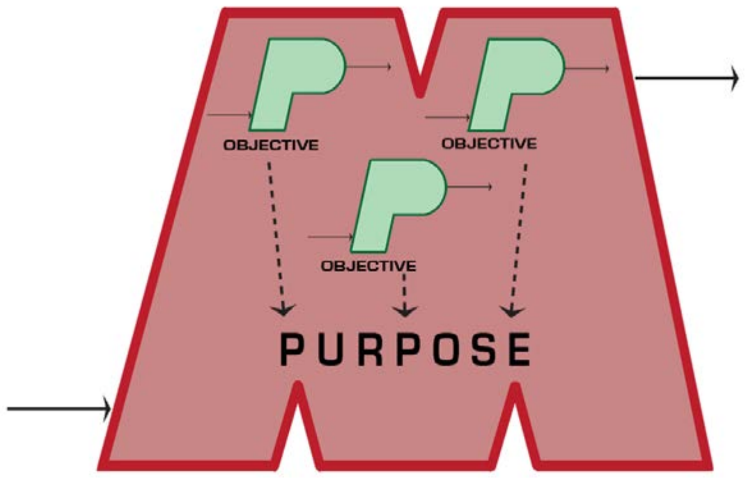


Figure 2 – Coherent set of practices

8.2.4.2 Consistent Set of Practices
A set of method practices is *consistent* if:

- there exists at least one practice which input is similar with the method's input and at least one practice which 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.

Figure 3 illustrates a consistent set of practices.

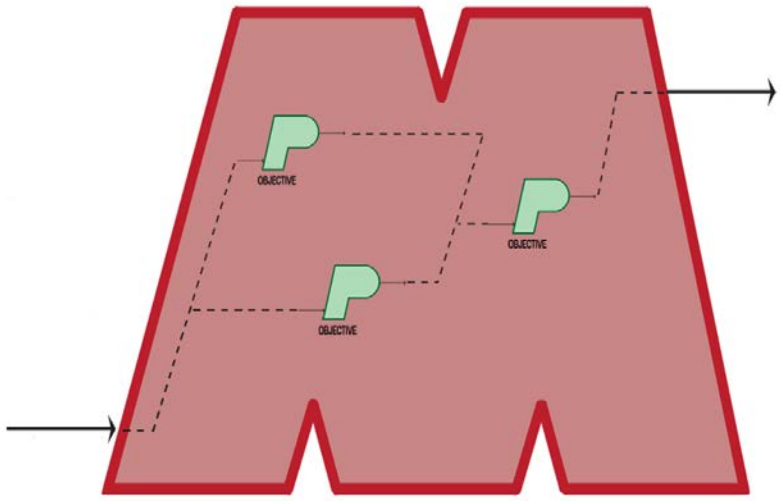


Figure 3 – Consistent set of practices

8.2.4.2.1 Similar

Two or more elements are *similar*, if according to the practitioner's judgment their characteristics are analogous.

8.2.4.3 Complete Set of Practices

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

Figure 4 illustrates a complete set of practices.

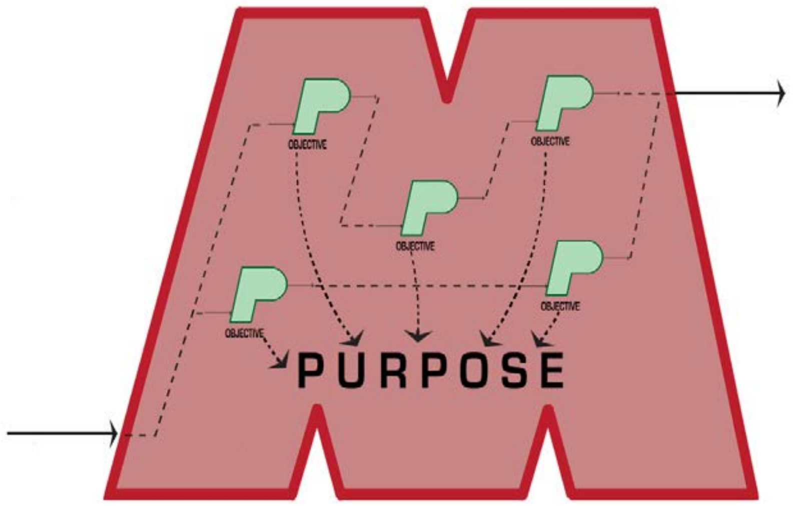


Figure 4 – Complete set of practices

8.2.5 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.

8.2.5.1 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.

8.2.5.2 Practice Patterns

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.

8.2.6 Methods and Practices Infrastructure Operations

8.2.6.1 Composition

Composition of practices consists in putting together practices in order to make up a method with a specific purpose, to form a family with a particular objective or to create a pattern as a reusable solution.

The practices are taken from MPI and organized according to the practitioner's judgment. The composition operation can also be applied to methods, families of practices and practice patterns.

Figure 5 illustrates the composition of practices to make up a method.

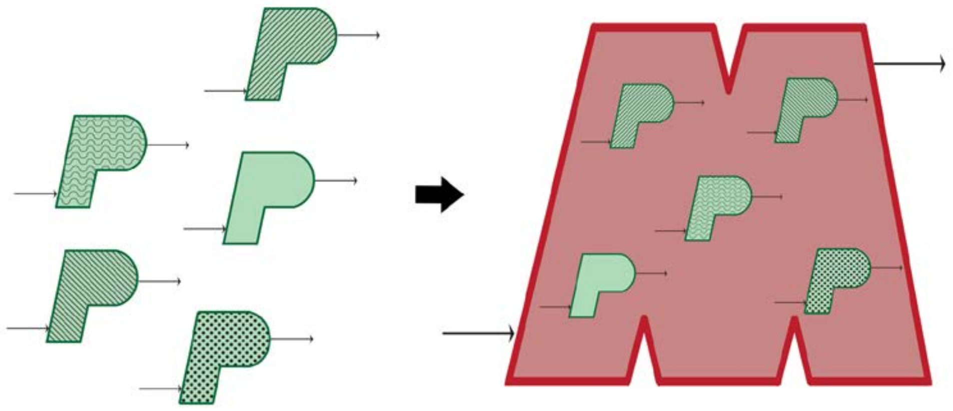


Figure 5 – Practices composition

8.2.6.2 Modification

A practice modification consists in the adjustment or change, done 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 a part of a practice.

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

Figure 6 illustrates the modification of a practice.

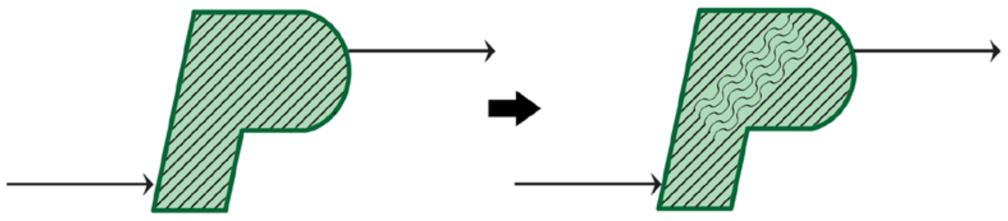


Figure 6 – Practice modification

8.3 Software Project Common Concepts Templates

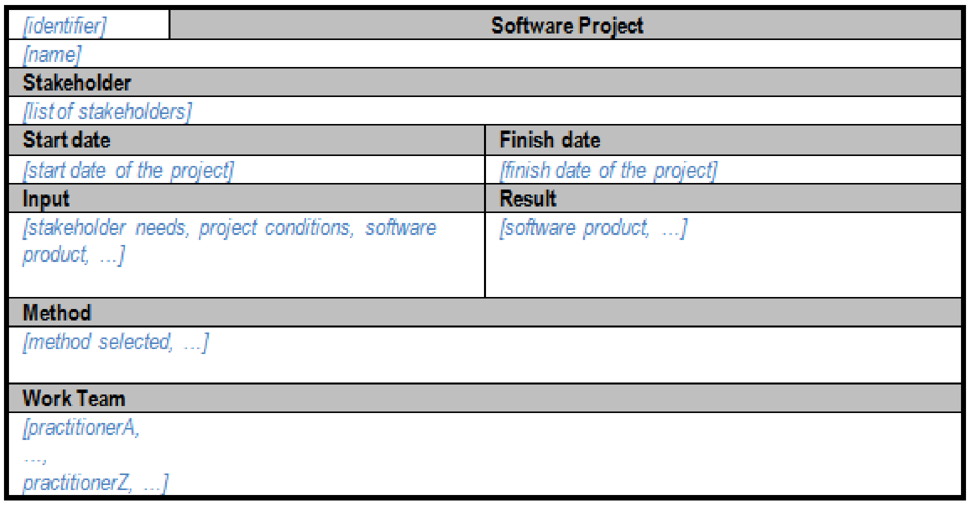
The methods and practices infrastructure and its content are extensible and adaptable in order to support the needs of a wide variety of methods and practices and to allow flexibility in the definition and application of these methods by practitioners in a work team.

Practitioners can use a set of templates to extend the methods and practices infrastructure and to register software projects' basic information. The templates are expected to be filled in when the common concepts are instantiated. The templates to capture particular software project information and definitions of method and practices are provided.

8.3.1 Software Project Template Structure

Practitioners can instantiate the software project common concept using the template shown in Table 1. The template includes the information and data required by the software project concept.

Table 1 — Software Project template



[identifier]	**Software Project**
[name]	
Stakeholder	
[list of stakeholders]	
Start date	**Finish date**
[start date of the project]	*[finish date of the project]*
Input	**Result**
[stakeholder needs, project conditions, software product, …]	*[software product, …]*
Method	
[method selected, …]	
Work Team	
[practitionerA, …, practitionerZ, …]	

8.3.2 Method Template Structure

Practitioners can instantiate the method common concept using the template shown in Table 2. The template asks for the information and data required by the method concept. 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.

Table 2 — Method template

[identifier]	Method
[name]	
Purpose	
[purpose]	
Input	Result
[stakeholder needs, project conditions,…]	*[software product,…]*
Practices	
[practiceRequirements, …, practiceDelivery, …]	

8.3.3 Practice Template Structure

Practitioners can instantiate the practice common concept using the template shown in Table 3. The template asks for the information and data required by the practice concept. These data have to be collected by the practitioners according with their experience and knowledge. The filled in template will be stored in the organizational methods and practices infrastructure.

Table 3 — Practice template

[identifier]	Practice
[name]	
Objective	
[objective]	
Input	**Result**
[expected characteristics,...]	[expected characteristics,...]
Guide	
Activities	
[list of activities]	
Tasks (optional)	
[toDoThis, ..., toDoThat, ...]	
Tools (optional)	
[list of proposed tools]	
Knowledge and Skills	
[abilities, competences, attainments,...]	
Verification Criteria	
[criterionA, criterionB,...]	
Measures	
[mesaureA, measureB, ...]	

8.4 Software Project Common Concepts 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 7 shows the software project representation as letter J.

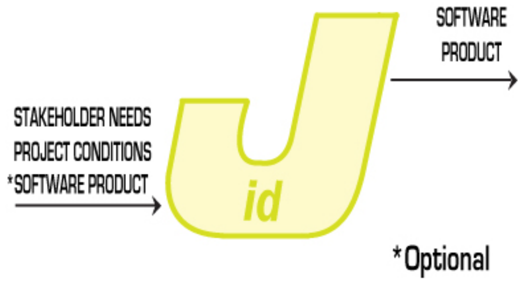


Figure 7 – Project symbol

The letter M is used to represent graphically a method, together with its input and result. See Figure 8.

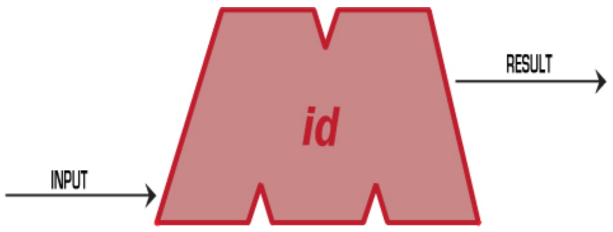


Figure 8 – Method symbol

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

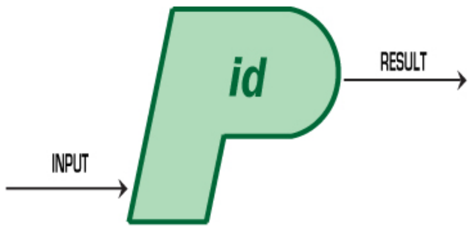


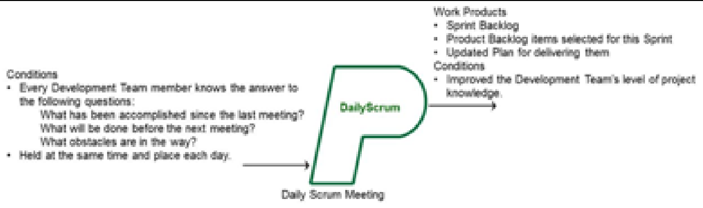
Figure 9 – Practice symbol

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.

8.5 Static View Example

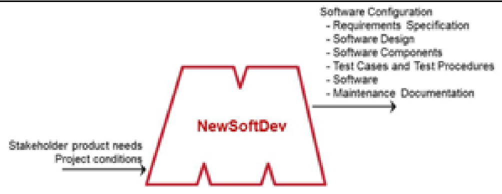
To illustrate the use of the KUALI-BEH practice template, the Daily Scrum Meeting event was chosen (see Table 4). The content of the practice is based on *The Scrum Guide -The Definitive Guide to Scrum: The Rules of the Game* [13], developed and sustained by Ken Schwaber and Jeff Sutherland. The remaining Scrum events practice templates can be found in Annex E.

Table 4 — Daily Scrum Meeting practice

DailyScrum	**Practice**
Daily Scrum Meeting	
 Conditions • Every Development Team member knows the answer to the following questions: What has been accomplished since the last meeting? What will be done before the next meeting? What obstacles are in the way? • Held at the same time and place each day. DailyScrum Daily Scrum Meeting Work Products • Sprint Backlog • Product Backlog items selected for this Sprint • Updated Plan for delivering them Conditions • Improved the Development Team's level of project knowledge.	
Objective	
Development Team meeting to synchronize activities and create (adapt) a plan for the next 24 hours. To assess progress toward the Sprint Goal and to assess how progress is trending toward completing the work in the Sprint Backlog.	
Input	**Result**
Conditions • *Every Development Team member knows the answer to the following questions:* *What has been accomplished since the last meeting?* *What will be done before the next meeting?* *What obstacles are in the way?* • *Held at the same time and place each day.*	*Work products* • *Sprint Backlog* • *Product Backlog items selected for this Sprint* • *Updated Plan for delivering them* *Conditions* • *Improved the Development Team's level of project knowledge.*
Guide	
Activities	
1. *During the meeting, each Development Team member explains:* *What has been accomplished since the last meeting?* *What will be done before the next meeting?* *What obstacles are in the way?* 2. *The Development Team often meets immediately after the Daily Scrum to re-plan the rest of the Sprint's work.*	
Knowledge and Skills	
Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.	
Verification Criteria	
Development Team should be able to explain to the Product Owner and Scrum Master how they try to work together as a self-organizing team to accomplish the goal and create the anticipated increment in the remainder of the Sprint.	
Measures	
Meeting duration [suggested time-box 15 minutes].	

To illustrate how the KUALI-BEH method template can be used, the Software Implementation process activities of the ISO/IEC 29110 5-1-2 Basic profile were chosen (see Table 5). The complete definition of the NewSoftDev practice templates can be found in Annex E.

Table 5 — NewSoftDev method

NewSoftDev	**Method**
Method for developing a new software product.	
 Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software - Maintenance Documentation NewSoftDev Stakeholder product needs Project conditions	
Purpose	
Systematically perform the analysis, design, construction, integration and tests activities for new software products according to the specified requirements.	
Input	**Result**
Stakeholders product needs:	*Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components* - *Software* - *Test Cases and Test Procedures* - *Test Report* - *Maintenance Documentation*
Statement of Work • *Product description: purpose of the product and general customer requirements.* • *Scope description of what is included and what is not* • *Project objectives* • *List of products to be delivered to customer*	
Project conditions:	
• *Project conditions established by the customer.* • *Schedule of the Project.* • *Identification of Project Risks.*	
Practices	
Software Requirements Analysis (SRA) *Software Architectural and Detailed Design (SADD)* *Software Construction (SC)* *Software Integration and Tests (SIT)* *Product Delivery (PD)*	

9 KUALI-BEH Operational View

KUALI-BEH operational view describes the software project execution. Section 9.1 presents a general outline of this view. The practice instance lifecycle is presented in section 9.2. The method enactment and adaptation during a software project execution are described in section 9.3 and 9.4 respectively. A method and practice instance boards for practitioners are proposed in section 9.5. Finally, section 9.6 shows an operational view example.

9.1 Induction to Software Project Execution

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

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 existent *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 successfully the selected *method*, the *work team* has to fulfill the *knowledge and skills* requirements specified in the *practices guide*. If it 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 done during the *software project* execution. To reach this goal, the *work team* has to analyze the *practices* of the selected *method* and, if necessary, apply the *practice* substitution, concatenation, splitting or merging. In other words, one *practice* can be substituted by an equivalent one (substitution), two *practices* can be juxtaposed (concatenation), one *practice* can be divided into two *practices* (splitting) or two *practices* can be integrated in one (merging).

The consistency, coherence and completeness properties of the original set of *practices* have to 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*. As a result, the *method* 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.

To start the *practice* instance execution, the *work team* has to estimate the measures associated to the *practice,* agree on the work distribution, on who is responsible for it and begin to work. This means that the practice instance changes to an **in-execution** state and the *method* enactment changes to an **in-progress** state.

During the *practice* instance execution, the *work team* can decide to interrupt it, so the *practice* instance changes to a **stand-by** state. At some point, the *work team* can decide to restart and the *practice* instance changes again to an **in-execution** state.

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

If the *work team* verifies the *result* as correct, the *practice* instance is **finished**. If it is not the case, the *work team* should correct the *result* and the *practice* instance goes back again to the **in-execution** state. In some cases, the *work team* can decide to cancel the *practice* instance. If the practice is **finished** or **cancelled**, the measures associated to 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 adaptation of *method practices*; taking into account the *practice* instance cancelation, the *stakeholder needs* change requests, the changes to the *project conditions*, or anything else that can affect the project.

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

9.2 Practice Instance Lifecycle

During the enactment of a method by a work team (WT), each practice is initially instantiated, later is constantly changing its state until it is finished or canceled. The valid practice instance states during their lifecycle are shown in Table 6.

Table 6 — Practice instance lifecycle states

Practice Instance 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.

The transitions between practice instance states are described in Table 7.

Table 7 — Practice instance lifecycle transitions

From Practice Instance State	Event that causes the transition	To Practice Instance 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

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

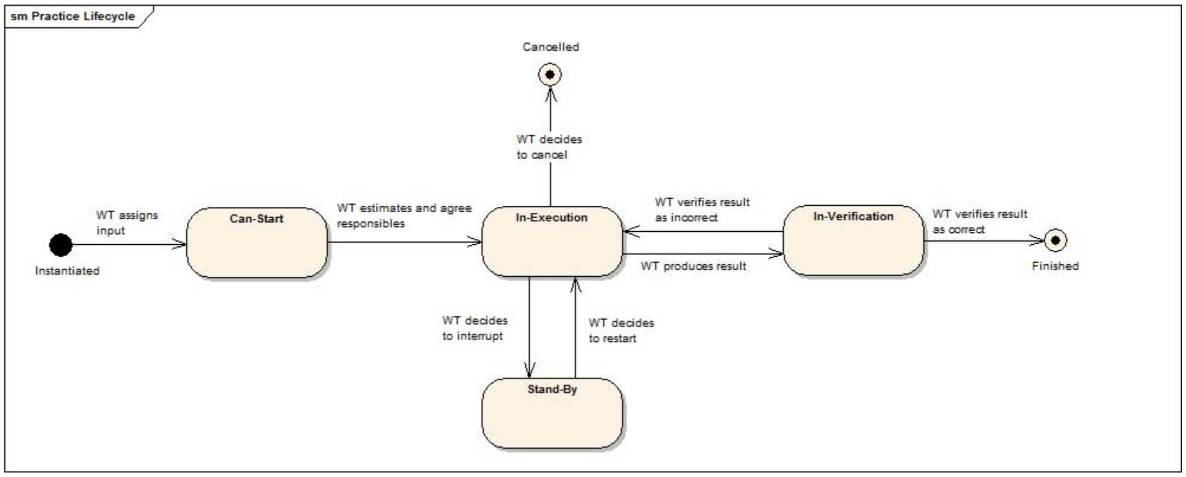


Figure 10 – Practice instance states and transitions

9.3 Method Enactment

A method enactment occurs in the context of a software project execution. Before starting the method enactment, the assigned to the software project work team gets to know the stakeholder needs and is 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.

The valid states of a method enactment, done by a work team during the project execution, are shown in Table 8.

Table 8 — Method enactment states

Method Enactment 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 9.

Table 9 — Method enactment transitions

From Method Enactment State	Event that causes the transition	To Method Enactment 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 merging. 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

The method enactment can reach more than one state at the same time, caused by the behavior of the practice instances lifecycle. For example, in some moment, a group of practice instances can be in execution state, other practices in can start state and others are finished, causing that the method enactment reaches different states at the same time. So, the method enactment behavior can be represented as a variation of a non-deterministic finite-state machine.

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

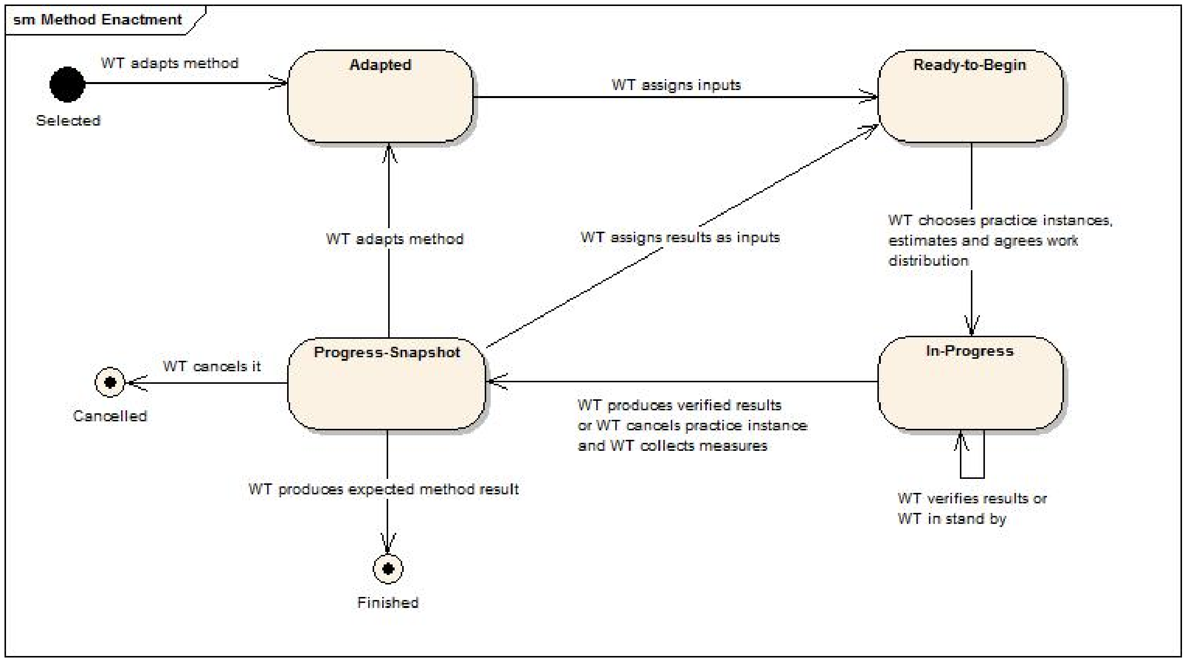

sm Method Enactment
WT adapts method
Selected
Adapted
WT assigns inputs
Ready-to-Begin
WT adapts method
WT assigns results as inputs
WT chooses practice instances, estimates and agrees work distribution
Progress-Snapshot
In-Progress
WT cancels it
Cancelled
WT produces verified results or WT cancels practice instance and WT collects measures
WT produces expected method result
WT verifies results or WT in stand by
Finished

Figure 11 – Enactment of a method

9.4 Method Adaptation

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

The purpose of adapting a method is to identify and/or modify the work units need to be done during the software project execution. To reach this goal the following actions should be done:

- WT has to analyze the practices of the selected method or the remaining practice instances and, if necessary, apply the practice substitution, concatenation, splitting or merging.
- 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.

The practice substitution, concatenation, splitting and merging are defined in the next subsections.

9.4.1 Practice Notation

Let’s define a practice *P* as a triple formed by an Input (I), an Objective (O) and a Result (R)

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

9.4.2 Substitution of Practices

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

$$\begin{aligned}&\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ practices,}\\ &P_1 \text{ can be } \textit{substituted} \text{ by } P_2 \text{ if and only if:}\\ &\quad P_1 \text{ is equivalent to } P_2\end{aligned}$$

The equivalence between practices holds when similar results are reached starting from similar inputs and similar objectives are fulfilled.

$$\begin{aligned}&\text{A practice } P \text{ is } \textit{equivalent} \text{ to a practice } P' \text{ if and only if:}\\ &\quad I \text{ is similar to } I' \text{ and}\\ &\quad R \text{ is similar to } R' \text{ and}\\ &\quad O \text{ is similar to O}'\end{aligned}$$

Notice that similarity is recognized and dictated by the practitioner’s judgment.

Figure 12 illustrates the substitution of a practice.

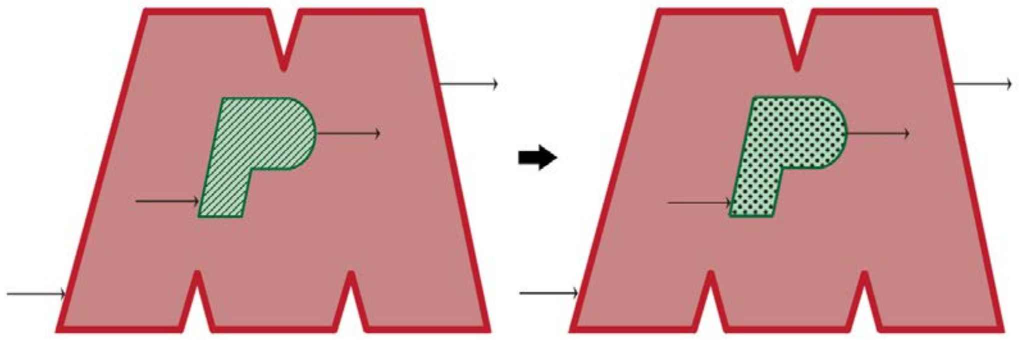


Figure 12 – Practice substitution

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

9.4.3 Concatenation of Practices

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 resulting objective will be the union of both original objectives.

Formally, the concatenation operation is defined as follows:

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

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

Figure 13 illustrates the concatenation of practices.

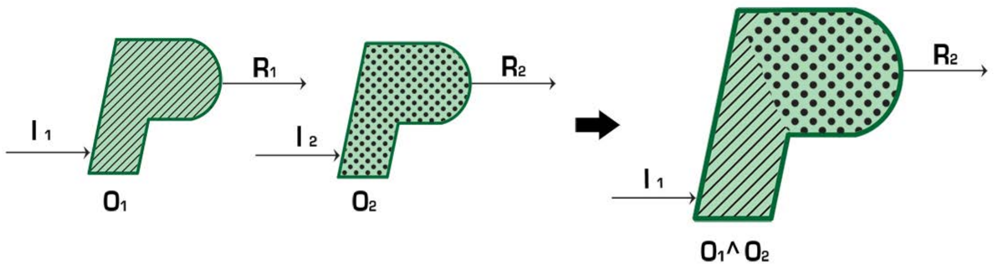


Figure 13 – Practice concatenation

9.4.4 Splitting of Practices

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

Formally, the splitting operation is defined as follows:

Let $P_1 = (I_1, O_1, R_1)$ and $P_2 = (I_2, O_2, R_2)$ practices.
P_1 and P_2 are a correct *split* of $P = (I, O, R)$ if:
I_1 union I_2 is similar to I and
R_1 union R_2 is similar to R and
O_1 and $O_2 = O$

Figure 14 illustrates the splitting of a practice.

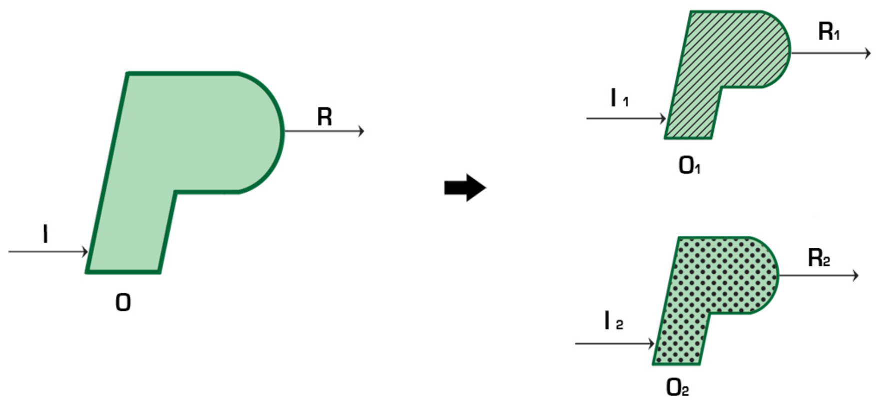


Figure 14 – Practice splitting

9.4.5 Merging of Practices

A practice merging consists in bringing two different practices into one. The resulting practice preserves the original objectives accomplishment and an integrated guide. The integrated guide is formed by the activities of both original practices merged into a new one.

Formally, the merging operation is defined as follows:

Let $P_1 = (I_1, O_1, R_1)$ and $P_2 = (I_2, O_2, R_2)$ practices.
$P = (I, O, R)$ is a correct *merge* of P_1 and P_2 if:
I is similar to I_1 union I_2 and
R is similar to R_1 union R_2 and
$O = O_1$ and O_2

If operations of practice substitution, concatenation, splitting and merging are applied strictly following the mentioned rules, the original properties of the method coherency, consistency and completeness are preserved.

Figure 15 illustrates the merging of practices.

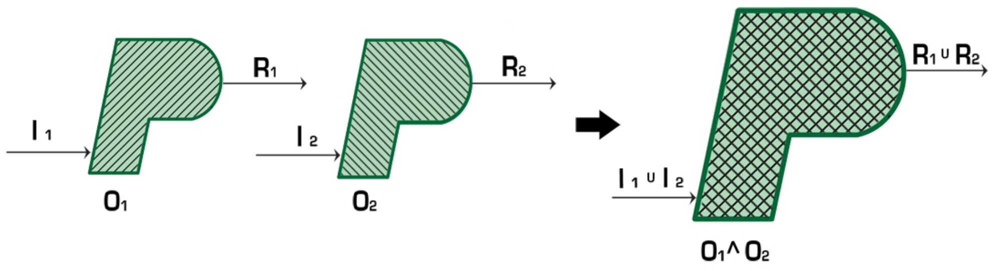


Figure 15 – Practice merging

9.5 Method Enactment and Practice Instance Boards

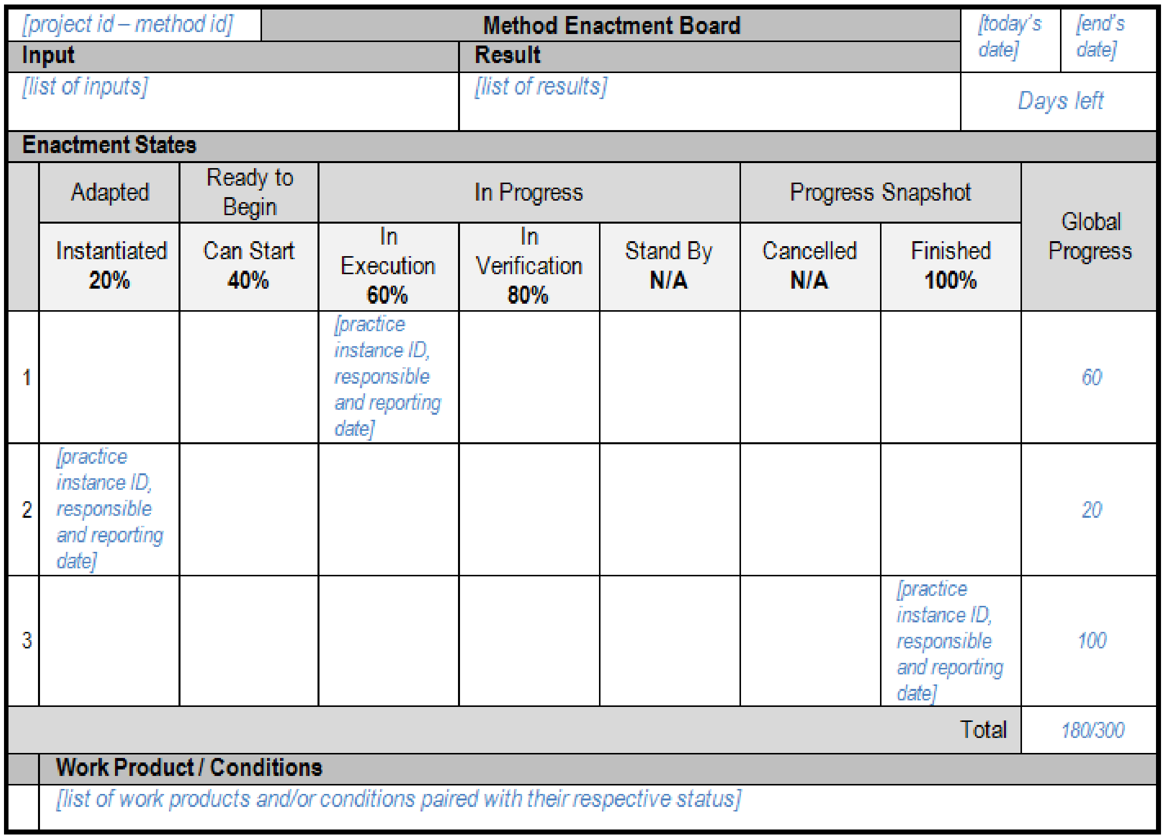
During the execution of a project, the work team needs to visualize the project´s on-going performance. The method enactment and the practice instance boards are used to display project relevant information. In the next subsections each board is presented in detail.

9.5.1 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 10 shows a proposed board for the method enactment.

Table 10 — Method enactment board



[project id – method id]		Method Enactment Board					[today's date]	[end's date]
Input			**Result**					
[list of inputs]			[list of results]				Days left	
Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated **20%**	Can Start **40%**	In Execution **60%**	In Verification **80%**	Stand By **N/A**	Cancelled **N/A**	Finished **100%**	
1			[practice instance ID, responsible and reporting date]					60
2	[practice instance ID, responsible and reporting date]							20
3							[practice instance ID, responsible and reporting date]	100
							Total	180/300
	Work Product / Conditions							
	[list of work products and/or conditions paired with their respective status]							

9.5.2 Practice Instance Board

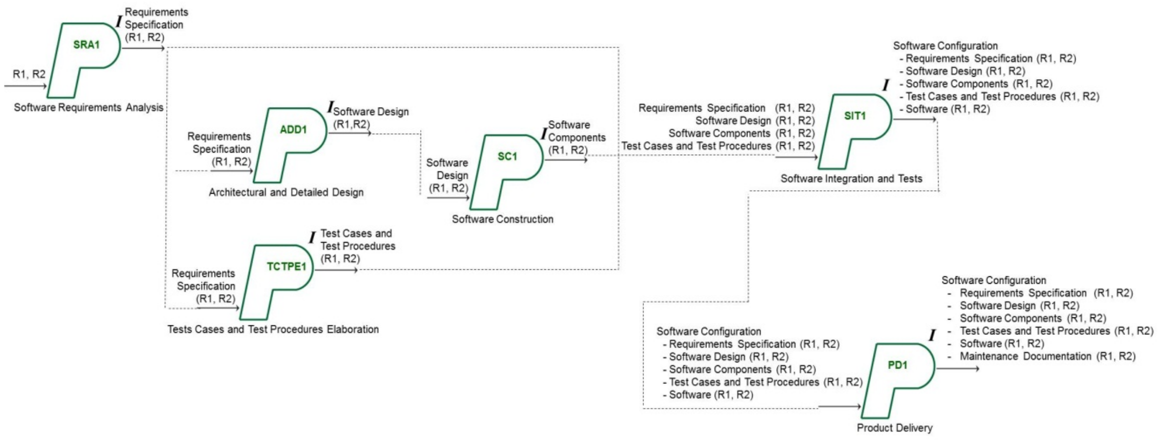
The practice instance board reflects the practice state at one particular moment. Each practice instance board also represents the responsible for it work team and associated to it measures. A numerical value together with the estimated and actual start and end dates can be associated to each practice instance state in order to calculate its progress. Table 11 shows a proposed board for practice instances.

Table 11 — Practice instance board

[project id – method id – practice id]	**Practice Instance Board**					
Input			**Result**			
[list of inputs]			*[list of results]*			
Work Team Practitioners			**Measures**			
[list of responsible practitioners]			Estimated		Actual	
			[list of measures estimations]		*[list of actual measures]*	
Activity Progress						
Activities		Progress	Responsible	Comments		
[activity 1]		*[numerical value]*	*[work team practitioner]*	*[comments and important notes]*		
Practice Instance States						
Instantiated **20%**	Can Start **40%**	In Execution **60%**	In Verification **80%**	Stand By **N/A**	Cancelled **N/A**	Finished **100%**

9.6 Operational View Example

The KUALI-BEH graphical practice symbols can be used to represent the planned practice instances and their input – result dependencies during the method enactment. Figure 16 shows the example.

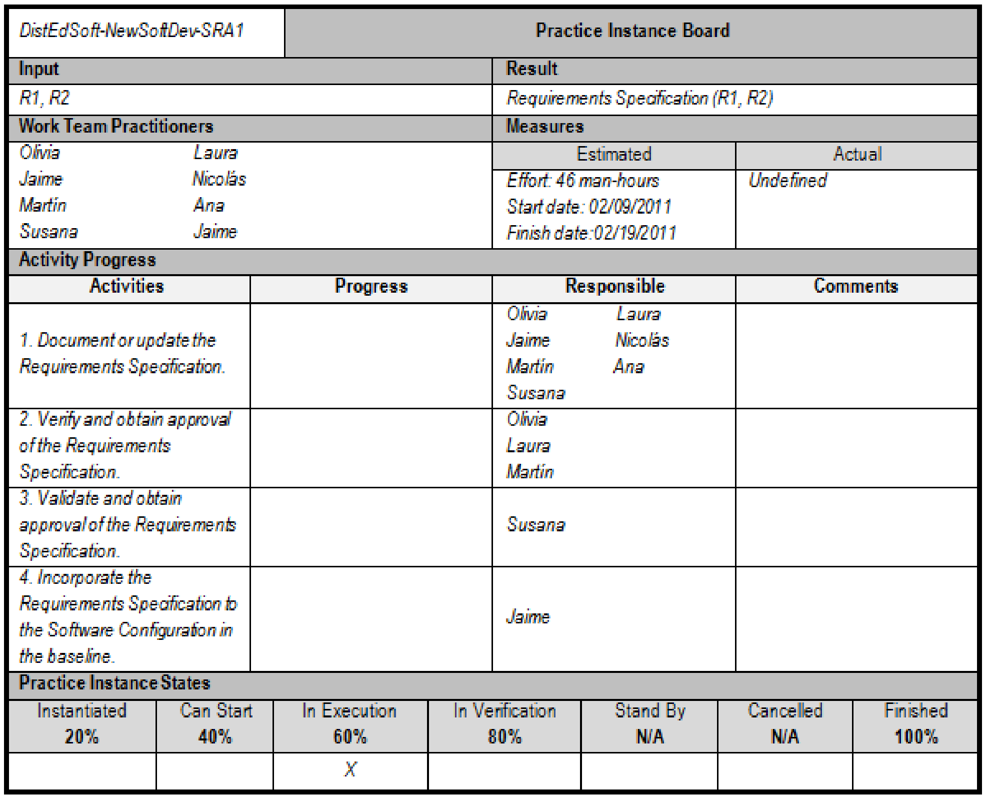


SRA1
R1, R2
I Requirements Specification (R1, R2)
Software Requirements Analysis
ADD1
Requirements Specification (R1, R2)
I Software Design (R1,R2)
Architectural and Detailed Design
SC1
Software Design (R1, R2)
I Software Components (R1, R2)
Software Construction
TCTPE1
Requirements Specification (R1, R2)
I Test Cases and Test Procedures (R1, R2)
Tests Cases and Test Procedures Elaboration
SIT1
Requirements Specification (R1, R2)
Software Design (R1, R2)
Software Components (R1, R2)
Test Cases and Test Procedures (R1, R2)
I Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
Software Integration and Tests
PD1
Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
I Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
- Maintenance Documentation (R1, R2)
Product Delivery

Figure 16 – Example of the adapted practices instances of NewSoftDev method

Table 12 shows the example of a practice instance board in In-Execution state.

Table 12 — SRA1 instance board in In-Execution state



DistEdSoft-NewSoftDev-SRA1	**Practice Instance Board**		
Input		**Result**	
R1, R2		*Requirements Specification (R1, R2)*	
Work Team Practitioners		**Measures**	
Olivia	*Laura*	Estimated	Actual
Jaime	*Nicolás*	*Effort: 46 man-hours*	*Undefined*
Martín	*Ana*	*Start date: 02/09/2011*	
Susana	*Jaime*	*Finish date:02/19/2011*	
Activity Progress			
Activities	**Progress**	**Responsible**	**Comments**
1. Document or update the Requirements Specification.		*Olivia Laura Jaime Nicolás Martín Ana Susana*	
2. Verify and obtain approval of the Requirements Specification.		*Olivia Laura Martín*	
3. Validate and obtain approval of the Requirements Specification.		*Susana*	
4. Incorporate the Requirements Specification to the Software Configuration in the baseline.		*Jaime*	

Practice Instance States						
Instantiated **20%**	Can Start **40%**	In Execution **60%**	In Verification **80%**	Stand By **N/A**	Cancelled **N/A**	Finished **100%**
		X				

Table 13 shows the example of a method enactment board. A more detailed example of the method enactment in the context of specific project is presented in Annex E.

Table 13 — Example of the NewSoftDev method enactment board

DistEdSoft-NewSoftDev	Method Enactment Board						*02/07/2011*	*08/15/2011*
Input			Result				118 days left.	
Stakeholders product needs:			*DistEdSoft Software Configuration* *- Requirements Specification* *- Software Design* *- Software Components Software* *- Test Cases and Test Procedures* *- Test Report* *- Maintenance Documentation*					
Statement of Work *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*								
Project conditions:								
Project conditions established by the customer *C1. Enrollment and On-line courses are the highest priority requirements.* *C2. Delivery deadline of the highest priority requirements cannot be changed.*								
Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
1st. increment								
1							*SRA1*	100
2			*ADD1*					60
3			*TCTPE1*					60
4	*SC1*							20
5	*SIT1*							20
6	*PD1*							20
2nd. Increment								
7	*SRA2*							20
8	*SADD2*							20
9	*SC2*							20
10	*SIT2*							20
11	*PD2*							20
							Total	380/1100
	Work Product / Conditions							
	Statement of Work (R1, R2, R3 and R4) –Agreed *Requirements Specification (R1, R2) -Validated*							

10 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. This ontology is supposed to be used by the method engineers as the means of description, analysis and reasoning about software projects and the information related to them. Section 10.1 presents a general background and the KUALI-BEH ontology requirements specification. The KUALI-BEH ontology definition is presented in section 10.2.

10.1 Ontology Background

The KUALI-BEH ontology has been developed using the Representation Formalism for Software Engineering Ontologies (REFSENO) [10]. 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 [10].

REFSENO provides constructs to define concepts with their attributes and the relationships between them. REFSENO is based on the construction of three tables using 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.

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 [10]. Table 14 defines the KUALI-BEH ontology requirements specification.

Table 14 — Ontology Requirements Specification

KUALI-BEH Ontology Requirements Specification	
Domain	Software Projects
Date	June 23, 2012
Conceptualized by	Miguel Morales Trujillo and Hanna Oktaba.
Purpose	Describe the common concepts involved in software projects and their relationships.
Level of Formality	Semi-formal (UML Diagrams, text and tables REFSENO).
Scope	List of concepts: • Software Project • Methods and Practices Infrastructure • Pattern • Method • Practice • Guide • Activity • Task • Tool • Input • Result • Condition • Work Product • Work Team • Practitioner • Knowledge and Skills • Stakeholder • Project Conditions • Stakeholder Needs • Software Product
	Instances: none
	Attributes: • Purpose (Method) • Objective (Practice) • Verification Criteria (Practice) • Measures (Practice) • Status (Work Product)
Source of Knowledge	See References section.

10.2 Ontology Definition

After establishing the ontology requirements specification, REFSENO suggests a process model for developing the ontology itself. Therefore, the suggested process model and a UML class diagram have been used to develop the KUALI-BEH ontology. 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 based on UML and a textual semi-formal representation of knowledge based on REFSENO. Figure 17 shows the UML class diagram used to develop the ontology.

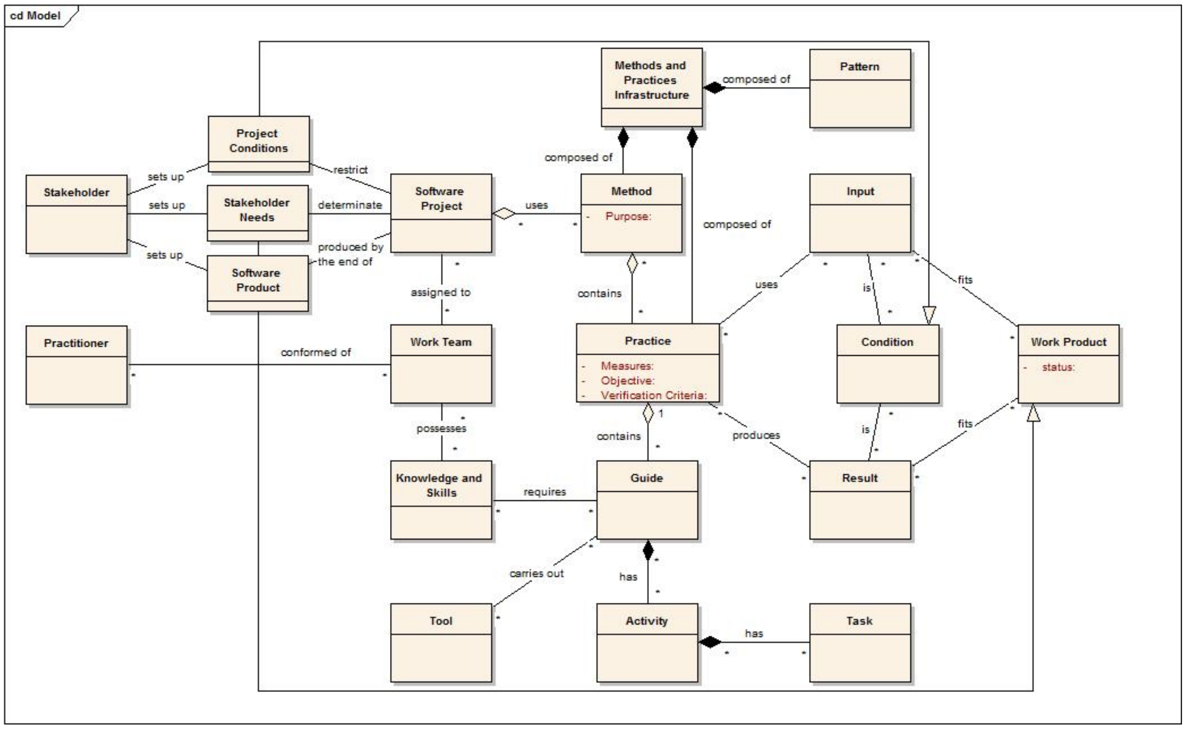


Figure 17 – Software project common concepts and their relationships and attributes

10.2.1 Concept Glossary

The concept glossary lists alphabetically all concepts of the ontology. One row of the concept 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 Annex D where the sources considered and used to create the respective definition are located.

Table 15 presents the glossary of concepts that form the KUALI-BEH ontology.

Table 15 — Ontology Glossary of Concepts

KUALI-BEH Ontology 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.	D.13
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.	D.17
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	D.12

KUALI-BEH Ontology Concepts Glossary			
Name	Definition	Example	References
		approval of the Requirements Specification.	
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.	D.10
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.	D.15
Method	A method is an articulation of a coherent, consistent and complete set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.	Software Implementation	D.8
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	D.19
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	D.20
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	D.9
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.	Hanna, Miguel	D.7
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	D.5

KUALI-BEH Ontology Concepts Glossary			
Name	Definition	Example	References
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: - Functionality - User interface - External interfaces - Legal and regulative Each requirement is identified, unique and it is verifiable or can be assessed.	D.11
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	D.3
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	D.1
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	D.2
Stakeholder Needs	The stakeholder needs are the representation of requirements, demands or exigencies expressed by the stakeholders to the work team.	The Client Needs	D.4
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.	D.14
Tool	A tool is a device used to carry out a particular function.	Enterprise Architect	D.18
Work Product	A work product is an artifact utilized or generated by a practice. It could have a status associated.	Requirements Specification	D.16
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	D.6

10.2.2 Relationships

A relationship models the way in which a particular software engineering entity is related to other software engineering entities. The relationships 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 16 presents the relationships that form the KUALI-BEH ontology.

Table 16 — Ontology Relationships

KUALI-BEH Ontology Relationships		
Name	Concepts (Cardinality)	Description
Assigned to	Work Team (*) – Software Project (*)	A work team is assigned to a software project.
Carries out	Tool (*) – Guide (*)	A tool carries out a guide.
Composed of	Methods and Practices Infrastructure (*) – Pattern (*)	A methods and practices infrastructure is composed of patterns.
Composed of	Methods and Practices Infrastructure (*) – Method (*)	A methods and practices infrastructure is composed of methods.
Composed of	Methods and Practices Infrastructure (*) – Practice (*)	A methods and practices infrastructure is composed of practices.
Conformed of	Work Team (*) – Practitioner (*)	A work team is conformed of practitioners.
Contains	Method (*) – Practice (*)	A method contains practices.
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.
Has	Guide (*) – Activity (*)	A guide has activities.
Has	Activity (*) – Task (*)	An activity has tasks.
Is	Condition (*) – Input (*)	A condition is an input.
Is	Condition (*) –Result (*)	A condition is a result.
Possesses	Work Team (*) – Knowledge and Skills (*)	A work team possesses knowledge and skills.
Produced by the end of	Software Product (*) – Software Project (1)	A software product is produced by the end of a software project.
Produces	Practice (*) – Result (*)	A practice produces a result.
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.
Uses	Software Project (*) – Method (*)	A software project uses a method.
Uses	Practice (*) – Input (*)	A practice uses an input.

10.2.3 Attributes

An attribute is represented using the concept attribute table. The concept attribute table 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 17 presents the attributes that form the KUALI-BEH ontology.

Table 17 — Ontology Attributes

KUALI-BEH Ontology Attributes				
Attribute (of Concept)	Description	Mandatory	Type	Cardinality
Measures (Practice)	List of standard units used to evaluate the practice performance and the objectives' achievement.	No	Text	1..*
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
Verification Criteria (Practice)	List of criteria associated to a result used to determine if a particular objective is achieved.	Yes	Text	1..*

Annex A: Mandatory Requirements
(Informative)

KUALI-BEH proposal satisfies the specific requirements stated in Chapter 6 of [1]. Table 18 presents the rationale to support the statement. Check the specific section marked in parenthesis; besides, feel free to contact any of the submission authors for further information about this issue.

Table 18 — Mandatory requirements and correspondent sections

6.5 Mandatory Requirements

6.5.1 The Kernel

6.5.1.1 Domain model

The KUALI-BEH proposal is represented as a domain model of 20 essential concepts of software engineering, their attributes and relationships.

KUALI-BEH includes the definition of each concept (8.2). The concepts are considered common for software projects because they were identified through concepts generalization found in models and standards related to software development. Moreover, these concepts have been used in several projects observed along the 30 years of academic and industry experience.

The following projects support the experience of one of the authors and has contributed to define the concepts of KUALI-BEH:

- **MoProSoft**: The process reference model for Mexican software organizations was published in 2003. In 2005 was declared as national standard NMX-I-059-NYCE-2005. At the moment more than 300 Mexican organizations have adopted the national standard.
- **COMPETISOFT Project (2006-2008)**: Taking as basis MoProSoft, this project provided a common framework suitable for small Latin American organizations dedicated to software development.
- **ISO/IEC 29110:2011**. Again with MoProSoft as basis the ISO/IEC TR 29110 Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 5-1-2: Management and Engineering Guide - Basic VSE Profile was developed.

6.5.1.2 Key conceptual elements

- **System**: the related concept is Software Product (8.2.1.2).
- **Functionality**: the related concept is Stakeholder Needs (8.2.1.3).

- **People**: the related concepts are Stakeholder (8.2.1.1), Practitioner (8.2.1.6) and Work Team (8.2.1.5). Also, Software Project (8.2.1) and Knowledge and Skills (8.2.3.6) were defined.

- **Way of Working**: the related concepts are Method (8.2.2) and Practice (8.2.3). Also, the Method Enactment (9.3) was defined to describe in detail the practitioners' way of working.

6.5.1.3 Generic activities

The Practice (8.2.3) common concept can be used to define any type of practices. The Guide (8.2.3.3), composed of activities, does not restrict the inclusion of any kind of activities, so it is defined with a generic focus.

6.5.1.4 Kernel elements

a) The section Software Project Common Concepts Definition (8.2) includes a concise definition for each concept.

b) The UML class diagram (10.2) represents the relationships between common concepts.

c) Practice Instance Lifecycle (9.2) and Method Enactment (9.3) describe the different states that the elements may take over time. Including the criteria appropriate for each element.

d) The examples (8.5, 9.6 and Annex E) illustrate the application in practice, including how it may be instantiated, tailored or extended to support the work of a specific project team using specific practices.

e) Measures (8.2.3) consider appropriate metrics that can be used to assess progress, quality or performance of a practice. Also, the Method Enactment Board (9.5.1) and the Practice Instance Board (9.5.2) provide a control view of measures.

6.5.1.5 Scope and coverage

The common concepts, that compose this proposal, are sufficient to allow the definition of practices and methods supporting projects of all sizes and a broad range of lifecycle models and technologies used by significant segments of the software industry.

6.5.1.6 Extension

a) The common concepts allow project and organization specific extensions in terms of new elements and providing detail on existing ones (8.3.2) and (8.3.3).

b) The common concepts are adaptable to specific domains of application and to projects (9.4).

6.5.2 The Language

6.5.2.1 The Language Definition

6.5.2.1.1 MOF metamodel
The Language was developed as the KUALI-BEH Ontology (10) based on REFSENO. The requirement to use MOF is discussed in annex B.3 of this document.
6.5.2.1.2 Static and operational semantics
The Static View (8) and Operational View (9) lay as a basis for semantics.
6.5.2.1.3 Graphical syntax
The graphical concrete syntax that formally maps to the abstract syntax is provided for practitioners (8.4).
6.5.2.1.4 Textual syntax
The textual concrete syntax that formally maps to the abstract syntax is provided as an ontology (10).
6.5.2.1.5 SPEM 2.0 metamodel reuse
This requirement is discussed as an annex (B.2) of this document.
6.5.2.2 Language Features
6.5.2.2.1 Ease of use
The KUALI-BEH proposal was designed to be easy to use for practitioners at different competency levels. A workshop to prove this requirement has been developed and is discussed in annex C of this document.
6.5.2.2.2 Separation of views for practitioners and method engineers
The KUALI-BEH proposal provides features to express two different views of a method, to method engineers (8 and mainly 10) and practitioners (8 and 9).
6.5.2.2.3 Specification of kernel elements
a) Formal (8.2-4, 9.2-5 and 10.2.1) and informal (8.1 and 9.1) descriptions of the content and meaning of the elements. b) The relationship of the elements (8.2 and 10.2.2). c) States the practice and method elements may take over time and the events that cause transitions among those states (9.2-3). d) How the element is instantiated, including provisions for practice-specific adaptation (tailoring) of the element, and the basis for comparing different instantiations (9.2-4). e) Metrics defined to assess various attributes of the use of the element (8.2).
6.5.2.2.4 Specification of practices
a) Description of the particular cross-cutting concern addressed by the practice and the goal of the application of the practice (8.2.3). b) The elements relevant to the practice and how they are instantiated for use in the practice (8.2.3). c) Any work products required by and produced by the practice (8.2.3.1-2, .7, .9).

d) The expected progress of work under the practice, including progress states, the rules for transition between them and their relation to the states of relevant elements (9.2 and 9.3).

e) Verification that the goal of the practice has been achieved in it application (8.2.3).

6.5.2.2.5 Composition of practices

a) Identifying the overall set of concerns addressed by composing the practices (8.2.2 and 8.2.6).

b) Merging two elements from different practices that should be the same in the resulting practice, even if they have different contents defined in the practices being composed (9.4.3).

c) Separating two elements from different practices that should be different in the resulting practice (9.4.4).

d) Modifying an existing method by replacing a practice within that method by another practice addressing a similar cross-cutting concern (8.2.6 and 9.4.2).

6.5.2.2.6 Enactment of methods

a) Tailoring the methods to be used on a project (9.4 and 8.2.6).

b) Communicating and discussing practices and methods among the project team (9.3).

c) Managing and coordinating work during a project, including modifications to the methods over the course of the project by further tailoring the use of the practices in the method (9.3).

d) Monitoring the progress of the project (9.3 and 9.5).

6.5.3 Practices

6.5.3.1 Examples of Practices

The working examples show the use of the elements to describe practices (8.5, 9.6 and Annex E).

6.5.3.2 Existing Practices and Methods

The examples of how existing, ISO/IEC-style and Agile-style, practices and methods can be migrated to the new proposed specification are shown (8.5, 9.6 and Annex E).

Annex B: Issues to be Discussed
(Informative)

Why KUALI-BEH is an Agile Creation and Enactment of Software Engineering Methods:

- Practitioners can start defining individual useful practices and then combine them in methods (coherent, consistent and complete practice sets). The traditional approach is to begin with processes, not easy to integrate, and the "agile" approach is to collect several practices (advices or techniques) not necessary consistent and complete.
- Method improvement can be done "offline" through the modifications of the organizational Methods and Practices Infrastructure, or "online" applying the method adaptation during its enactment. We think that the online adaptation adds the real agility to the software project execution.
- Work team is empowered in our proposal, because the main decisions on what to do, how to do it, who will do it, effort estimations, and others, are in their hands. So we try to follow the first principle of the Agile Manifesto "individuals and interactions over processes and tools".

B.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 the Collaborative Workshop, see Annex C.

The following can be alternatives for our software project common concepts:
- Guide – guidance
- Knowledge and Skills – competences
- Method – process (in use today) – methodology (in use decades before)
- Methods and Practices Infrastructure – organizational 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

B.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 at "activity", "guide", "work product", "tool" or "role" level of abstraction for example, but more abstract concepts are not easy to understand and some differences between them can be identified. Table 19 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 19 — KUALI-BEH and SPEM 2.0 concepts

List of 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	Methods and Practices Infrastructure	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 sections 8.5, 9.6 and Annex E, using KUALI-BEH permitted to model perfectly 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.

B.3 MOF issue

Knowledge can be represented on different levels of abstraction. For the purpose of this proposal three knowledge levels are used, the epistemological level, the conceptual level, and the linguistic level.

According to [10] 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, MOF 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 20 presents the mapping between MOF layers and the KUALI-BEH ontology.

Table 20 — MOF layers and KUALI-BEH ontology

MOF and KUALI-BEH Ontology Mapping		
Layer	MOF	KUALI-BEH Ontology
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

Annex C: Proof of Concept Statement
(Informative)

The design phase of this specification had been completed and prototyped.

The prototype was developed as a collaborative workshop, attended by software industry and research community members. 16 participants (practitioners and method engineers) from 3 software industry organizations plus 3 master students attended the workshop in order to understand the KUALI-BEH proposal and apply it in their organizations.

The methodology of the workshop included on-site and virtual interactive sessions with practitioners and method engineers in order to get feedback and improvements to the proposal. Besides, the participants carried out activities and surveys in order to apply the proposal in real life situations and analyze its usefulness, merits and drawbacks.

The workshop activities were divided into 8 two-hour sessions that took place every two weeks. The content of each session was organized as follows:

- **Webinar Briefing:**
    - Purpose exposition of the academy – industry collaborative research workshop
    - KUALI-BEH presentation and invitation to join the workshop
- **Session 1:**
    - Static view presentation
        - Induction
        - Software Project, Method and Practice common concepts
        - Graphical Representation
        - Practice Template
    - Activity: Documenting a practice that you execute in your daily work using the practice template
    - Survey: Similarity between the proposal and the real life. Pertinence, appropriateness and proficiency of the common concepts
- **Session 2:**
    - Review and discussion of the session 1 Activity and Survey results
    - Static view presentation
        - Method Template
        - Method properties
        - Methods and Practices Infrastructure
    - Activity: Documenting a method and its respective practices that you execute in your daily work using the method template
    - Survey: Pertinence and appropriateness of the method properties
- **Session 3:**
    - Review and discussion of the session 2 Activity and Survey results
    - Operational View presentation
        - Induction
        - Practice Instance Lifecycle
        - Method Enactment
    - Activity: Discussing about the differences and similarities between the real life and the proposed method enactment

    - Survey: Similarity between the proposal and the real life. Pertinence and appropriateness of the practice instance lifecycle and method enactment
- **Session 4:**
    - Review and discussion of the session 3 Activity and Survey results
    - Operational View presentation
        - Method Adaptation
        - Practice notation and operations
    - Activity: Applying the operations in order to adapt the previously documented method
    - Survey: Pertinence and appropriateness of the method adaptation and proficiency of the operations
- **Session 5:**
    - Review and discussion of the session 4 Activity and Survey results
    - Operational View presentation
        - Practice Instance Board
        - Method Enactment Board
    - Activity: Adapting the practice instance and method enactment boards to your daily work
    - Survey: Pertinence and appropriateness of the practice instance and method enactment boards
- **Session 6:**
    - Review and discussion of the session 5 Activity and Survey results
    - Analysis and discussion of the suggestions and improvements to the proposal expressed by the workshop participants
    - Experiment Design presentation
    - Activity: Carrying out the experiment in your organization
- **Session 7:**
    - Review and discussion of the session 6 Activity results
    - Presentation of the workshop results

At the end of the workshop, multiple benefits were identified and obtained by both parties.

On one hand, the software industry participants identified as benefits:

- Better organization of their knowledge though practices and methods
- Easy to transmit and apply their knowledge into the organization
- Foster the training of new people in the organization
- Attractiveness of the new approach to document the actual way of working, so to say, they document what they actually do and not what they are supposed to do.

On the other hand, the proposal was improved taking into account 93 suggestions from the workshop participants. The suggestions were obtained from the surveys applied to the participants or were directly expressed by participant during the sessions.

After reviewing and analyzing the suggestions, the submission team applied fully 36 suggestions, while 28 were applied with some modifications and 29 were rejected. The suggestions were mainly directed to the Method Enactment section. In order to obtain more feedback, proofs and improvements, a deeper experiment is planned to be carried out during the third quarter of the year in one of the organizations that participated in the workshop.

The proposal was presented, last April, at the XV Ibero-American Conference on Software Engineering CIbSE'12 taking place in Buenos Aires, Argentina with a warm reception and considering KUALI-BEH to be of great worth.

Also, the proposal had been reviewed by experts in the field from different countries obtaining important feedback, support and offers to collaborate from individual researchers and important research groups, especially from Alarcos research group, University of Castilla – La Mancha, headed by PhD Mario Piattini Velthuis.

Moreover, at Science Faculty of the UNAM, the Software Engineering undergraduate course for the Computer Science major is being redesigned using the KUALI-BEH approach. The theoretical part of the course is presented as a set of practices and the experimental part is based on the practice instance execution and method enactment. M. Sc. María Guadalupe Ibargüengoitia González is in charge of this project. The students are learning and practicing a broad scope of real software engineering in the academic environment, which, hopefully, will prepare them for their inclusion in the industry in a better manner.

Finally, we can mention that the design and construction of software tools to support KUALI-BEH has been started. The prototype of this set of tools is focused on promoting and maximizing the interaction and collaboration of the work team, integrating such multimedia elements as virtual boards and desktops. The aim is to apply the technology that endorses participation, discussion, collaboration and cooperation. For this project the PhD Fernando Gamboa Rodríguez, expert in human-machine interaction and creator of the *Classroom of the future* from the Center of Applied Sciences and Technological Development (CCADET-UNAM), is supporting the prototype development team.

The tools prototypes are being developed in conjunction with four master students and three researchers from Graduate Science and Engineering Computing, UNAM. The first prototypes are planned to be delivered later this year.

Annex D: Definitions and Sources Considered
(Informative)

List of definitions and sources used to create the definitions in this proposal.

D.1 Software Project

Project [9] **–** A temporary endeavor undertaken to create a unique product, service, or result.
Project [6] **–** Endeavour with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements.

D.2 Stakeholder

Stakeholder [6] – 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.

D.3 Software Product

Software product [6] – Set of computer programs, procedures, and possibly associated documentation and data.

D.4 Stakeholder Needs

Need [7] **–**

1. Circumstances in which something is necessary; necessity.
2. A thing that is wanted or required.

Need [8] – Want, requirement, requisite, demand, exigency.

D.5 Project Conditions

Condition [7] –

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 [8] **–** Circumstances, state, status, action.

D.6 Work Team

None

D.7 Practitioner

Practitioner [7] –

1. A person actively engaged in an art, discipline, or profession, especially medicine.

Practitioner [8] **–** Professional, expert, specialist.

Judgment [7] –

1. The ability to make considered decisions or form sensible opinions.

Judgment [8] **–** Discernment, experience, perception.

D.8 Method

Method [7] –

1. A particular procedure for accomplishing or approaching something.
2. Orderliness of thought or behaviour.

Method [8] – Means, procedure.

Method [1] – 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.

Process [5] – Set of interrelated or interacting activities which transforms inputs into outputs.

D.9 Practice

Practice [7] –

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 [8] – Routine, usual procedure.

Practice [1] – 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.

D.10 Input

None

D.11 Result

None

D.12 Guide

Guide [7] –

1. A directing principle or standard.

Guide [8] – Paradigm, pattern, advice.

D.13 Activity

Activity [7] –

1. A condition in which things are happening or being done.
2. An action taken in pursuit of an objective.

Activity [8] – State of being active.

Activity [4] – 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 [6] – Set of cohesive tasks of a process.

Activity [1] – 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.

D.14 Task

Task [7] **–**

1. A piece of work.

Task [8] **–** Job or chore, often assigned.

Task [1] **–** Task is a required, recommended or permitted action.

Task [6] – Requirement, recommendation, or permissible action, intended to contribute to the achievement of one or more outcomes of a process.

D.15 Knowledge and Skills

Knowledge [7] –

1. Information and skills acquired through experience or education.
2. Awareness or familiarity gained by experience.

Knowledge [8] – Person's understanding; information, ability, attainments.

Skill [7] –

1. The ability to do something well; expertise or dexterity.
2. Train (a worker) to do a particular task.

Skill [8] – Ability, talent to do something, competence.

D.16 Work Product

Input Products [4] – Products required to perform the process and its corresponding source, which can be another process or an external entity to the project.
Output Products [4] – Products generated by the process and its corresponding destination, which can be another process or an external entity to the project.
Internal Products [4] – Products generated and consumed by the process.
Product [5] – Result of a process.

D.17 Condition

Condition [7] –

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 [8] **–** Circumstances, state, status, action.

D.18 Tool

Tool [7] –

1. A device or implement, typically hand-held, used to carry out a particular function.

Tool [8] **–** Device, apparatus, instrument.

D.19 Methods and Practices Infrastructure

Practice Infrastructure [1] – 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.

D.20 Pattern

Pattern [11] – 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 [12] – 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.

D.21 Coherent

Coherent [7] –

1. (of an argument or theory) logical and consistent.
2. Holding together to form a whole.

Coherent [8] – Understandable.

D.22 Consistent

Consistent [7] –

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 [8] **–** Constant, regular.

D.23 Similar

Similar [7] –

1. Of the same kind in appearance, character, or quantity, without being identical.

Similar [8] **–** Analogous, coincident, congruent, matching.

D.24 Complete

Complete [7] –

1. Having all the necessary or appropriate parts; entire.
2. Having run its full course; finished.

Complete [8] **–** Total, not lacking.

Annex E: Static and Operational Views Examples
(Informative)

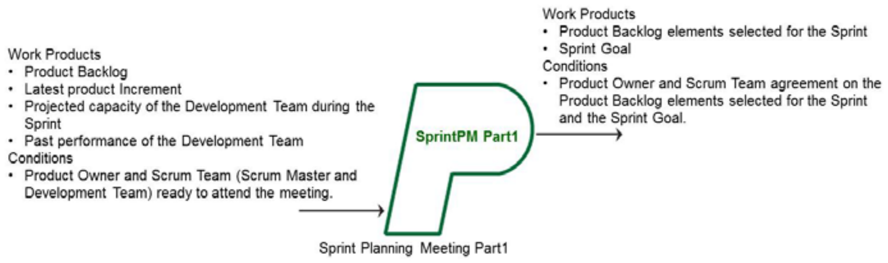
This annex contains two applications of the KUALI-BEH concepts. The first one is the use of KUALI-BEH practice templates to express the Scrum events in a structured way. The second one is the adaptation of the ISO/IEC 29110-5-1-2 Basic Profile Software Implementation process to the context of a fictional software development organization. An example of the method enactment during a specific project execution is provided. The aim is to illustrate the process and actions taken by the work team under particular circumstances of a project.

E.1 Scrum Practices Static View Example

This section explains how the KUALI-BEH practice templates can be used to express the Scrum events. The content of the practices is based on The Scrum Guide -The Definitive Guide to Scrum: The Rules of the Game [13], developed and sustained by Ken Schwaber and Jeff Sutherland.

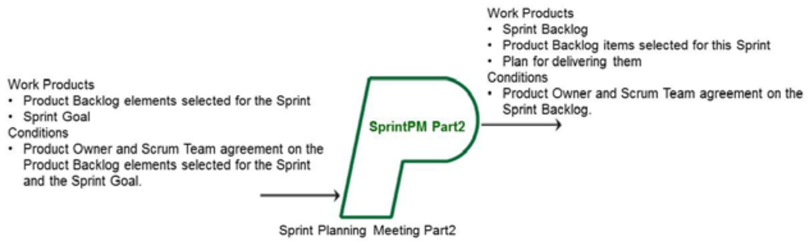
Tables 21-25 document the Sprint Planning Meeting (part 1 and 2), Daily Scrum Meeting, Sprint Review Meeting and Sprint Retrospective Meeting events. The structured presentation of Scrum events through the KUALI-BEH practice template format can be useful for educational and training purposes.

Table 21 — Sprint Planning Meeting Part 1 practice

SprintPM Part 1	Practice
Sprint Planning Meeting Part1	
Objective	
Forecast the functionality that will be developed during the Sprint and understand the work of the Sprint.	
 Work Products • Product Backlog • Latest product Increment • Projected capacity of the Development Team during the Sprint • Past performance of the Development Team Conditions • Product Owner and Scrum Team (Scrum Master and Development Team) ready to attend the meeting. SprintPM Part1 Sprint Planning Meeting Part1 Work Products • Product Backlog elements selected for the Sprint • Sprint Goal Conditions • Product Owner and Scrum Team agreement on the Product Backlog elements selected for the Sprint and the Sprint Goal.	
Input	Result
Work Products • *Product Backlog* • *Latest product Increment* • *Projected capacity of the Development Team during the Sprint* • *Past performance of the Development Team* *Conditions* • *Product Owner and Scrum Team (Scrum Master and Development Team) ready to attend the meeting.*	*Work Products* • *Product Backlog elements selected for the Sprint* • *Sprint Goal* *Conditions* • *Product Owner and Scrum Team agreement on the Product Backlog elements selected for the Sprint and the Sprint Goal.*
Guide	
Activities	

1. Product Owner presents ordered Product Backlog items to the Development Team *2. Entire Scrum Team collaborates on understanding the work of the Sprint* *3. The number of items is selected from the Product Backlog for the Sprint up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.* *4. Scrum Team crafts a Sprint Goal.*
Knowledge and Skills
Product Owner is the sole person responsible for managing the Product Backlog. *Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint*
Verification Criteria
Product Owner and Scrum Team agreed on the Product Backlog elements selected for the Sprint and the Sprint Goal.
Measures
Meeting duration [suggested time-box is four hours for a one-month Sprint]

Table 22 — Sprint Planning Meeting Part2 practice

SprintPM Part 2	Practice
Sprint Planning Meeting Part2	
 Work Products • Product Backlog elements selected for the Sprint • Sprint Goal Conditions • Product Owner and Scrum Team agreement on the Product Backlog elements selected for the Sprint and the Sprint Goal. SprintPM Part2 Sprint Planning Meeting Part2 Work Products • Sprint Backlog • Product Backlog items selected for this Sprint • Plan for delivering them Conditions • Product Owner and Scrum Team agreement on the Sprint Backlog.	
Objective	
The Development Team decides how the selected functionality will be built into a "Done" product Increment during the Sprint. Sprint Backlog composed of the Product Backlog items selected for this Sprint plus the plan for delivering them is defined.	
Input	Result
Work Products • *Product Backlog elements selected for the Sprint* • *Sprint Goal* *Conditions* • *Product Owner and Scrum Team agreement on the Product Backlog elements selected for the Sprint and the Sprint Goal.*	*Work products* • *Sprint Backlog* • *Product Backlog items selected for this Sprint* • *Plan for delivering them.* *Conditions* • *Product Owner and Scrum Team agreement on the Sprint Backlog.*
Guide	
Activities	
1. The Development Team starts by designing the system and the work needed to convert the Product Backlog into a working product increment. Work may be of varying size, or estimated effort. However, enough work is planned during the Sprint Planning meeting for the Development Team to forecast what it believes it can do in the upcoming Sprint. *2. Work planned for the first days of the Sprint by the Development Team is decomposed into units of one day or less by the end of this meeting. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during the Sprint Planning Meeting and as needed throughout the Sprint.* *3. The Product Owner may be present during the second part of the Sprint Planning Meeting to clarify the selected Product Backlog items and to help make trade-offs. If the Development Team determines it has too much or too little work, it may renegotiate the Sprint Backlog items with the Product Owner. The Development Team may also invite other people to attend in order to provide technical or domain advice.*	
Knowledge and Skills	
Product Owner is the sole person responsible for managing the Product Backlog *Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.*	
Verification Criteria	
By the end of the Sprint Planning meeting, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment.	

Measures
Meeting duration [suggested time-box is four hours for a one-month Sprint]

Table 23 — Daily Scrum Meeting practice

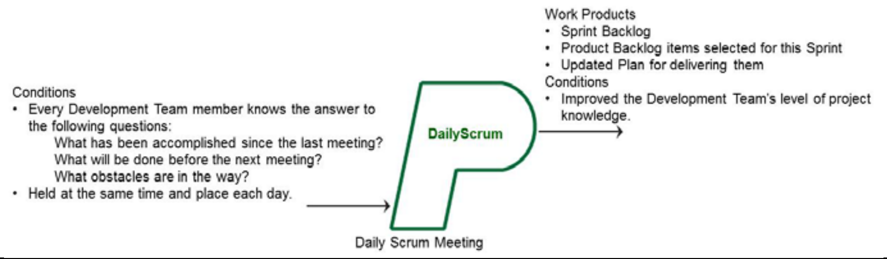
DailyScrum	Practice
Daily Scrum Meeting	
 Work Products • Sprint Backlog • Product Backlog items selected for this Sprint • Updated Plan for delivering them Conditions • Improved the Development Team's level of project knowledge. Conditions • Every Development Team member knows the answer to the following questions: What has been accomplished since the last meeting? What will be done before the next meeting? What obstacles are in the way? • Held at the same time and place each day. DailyScrum Daily Scrum Meeting	
Objective	
Development Team meeting to synchronize activities and create (adapt) a plan for the next 24 hours. To assess progress toward the Sprint Goal and to assess how progress is trending toward completing the work in the Sprint Backlog.	
Input	Result
Conditions • *Every Development Team member knows the answer to the following questions:* o *What has been accomplished since the last meeting?* o *What will be done before the next meeting?* o *What obstacles are in the way?* • *Held at the same time and place each day.*	*Work products* • *Sprint Backlog* • *Product Backlog items selected for this Sprint* • *Updated Plan for delivering them* *Conditions* • *Improved the Development Team's level of project knowledge.*
Guide	
Activities	
1. During the meeting, each Development Team member explains: o *What has been accomplished since the last meeting?* o *What will be done before the next meeting?* o *What obstacles are in the way?* *2. The Development Team often meets immediately after the Daily Scrum to re-plan the rest of the Sprint's work.*	
Knowledge and Skills	
Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.	
Verification Criteria	
Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work together as a self-organizing team to accomplish the goal and create the anticipated increment in the remainder of the Sprint.	
Measures	
Meeting duration [suggested time-box 15 minutes].	

Table 24 — Sprint Review Meeting practice

SprintReview	Practice
Sprint Review Meeting	

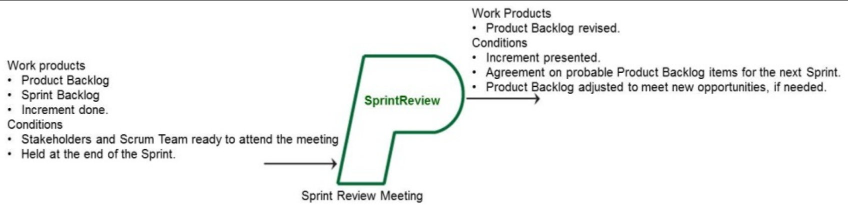
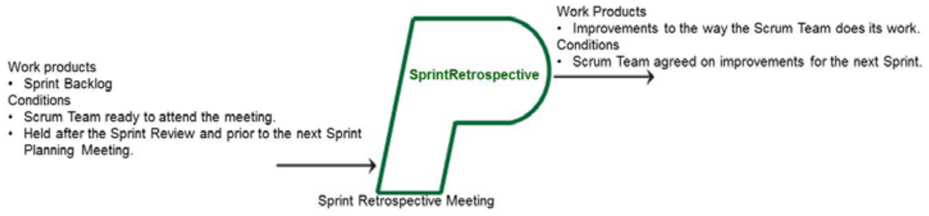
 Work products • Product Backlog • Sprint Backlog • Increment done. Conditions • Stakeholders and Scrum Team ready to attend the meeting • Held at the end of the Sprint. SprintReview Sprint Review Meeting Work Products • Product Backlog revised. Conditions • Increment presented. • Agreement on probable Product Backlog items for the next Sprint. • Product Backlog adjusted to meet new opportunities, if needed.	
Objective	
To inspect the Increment and adapt the Product Backlog, if needed. Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done.	
Input	Result
Work products • *Product Backlog* • *Sprint Backlog* • *Increment done.* *Conditions* • *Stakeholders and Scrum Team ready to attend the meeting* • *Held at the end of the Sprint.*	*Work products* • *Product Backlog revised.* *Conditions* • *Increment presented.* • *Agreement on probable Product Backlog items for the next Sprint.* • *Product Backlog adjusted to meet new opportunities, if needed.*
Guide	
Activities	
1. *The Product Owner identifies what has been "Done" and what has not been "Done";* 2. *The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;* 3. *The Development Team demonstrates the work that it has "Done" and answers questions about the Increment;* 4. *The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date. He or she may also adjust the overall Product Backlog to meet new opportunities ; and,* 5. *The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning Meetings.*	
Knowledge and Skills	
Product Owner is the sole person responsible for managing the Product Backlog *Stakeholders involved in the project.* *Scrum Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint.*	
Verification Criteria	
Product Backlog was revised and the probable Product Backlog items for the next Sprint were defined. *The Product Backlog may also be adjusted overall to meet new opportunities.*	
Measures	
Meeting duration [suggested time-box four hours for a one-month Sprint].	

Table 25 — Sprint Retrospective Meeting practice

SprintRetrospective	Practice
Sprint Retrospective Meeting	
 Work products • Sprint Backlog Conditions • Scrum Team ready to attend the meeting. • Held after the Sprint Review and prior to the next Sprint Planning Meeting. SprintRetrospective Sprint Retrospective Meeting Work Products • Improvements to the way the Scrum Team does its work. Conditions • Scrum Team agreed on improvements for the next Sprint.	
Objective	

The Sprint Retrospective is held by the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint. The goals of the meeting are: • *Inspect how the last Sprint went with regards to people, relationships, process, and tools;* • *Identify and order the major items that went well and potential improvements; and,* • *Create a plan for implementing improvements to the way the Scrum Team does its work.*	
Input	Result
Work products • *Sprint Backlog* *Conditions* • *Scrum Team ready to attend the meeting.* • *Held after the Sprint Review and prior to the next Sprint Planning Meeting.*	*Work products* • *Improvements to the way the Scrum Team does its work.* *Conditions* • *Scrum Team agreed on improvements for the next Sprint.*
Guide	
Activities	
1. *The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. The tasks to do are:* a. *Inspect how the last Sprint went with regards to people, relationships, process, and tools;* b. *Identify and order the major items that went well and potential Improvements; and,* c. *Create a plan for implementing improvements to the way the Scrum Team does its work.*	
Knowledge and Skills	
Scrum Master is responsible for ensuring Scrum is understood and enacted. *Development Team consists of professionals who do the work of delivering a potentially releasable Increment of “Done” product at the end of each Sprint.*	
Verification Criteria	
The Scrum Team should have identified Improvements that it will implement in the next Sprint and agreed on them.	
Measures	
Meeting duration [suggested time-box three-hour for a one-month Sprint].	

E.2 ISO/IEC 29110-5-1-2 Basic Profile Static and Operational Views Example

The aim of this section is to describe how the KUALI-BEH static and operational views can be used for the definition of practices and methods and their enactment in the context of a fictional software development organization. An example of the method enactment during a specific project execution is provided. The aim is to illustrate the process and actions taken by the work team under particular circumstances of a project.

E.2.1 ISO/IEC 29110 5-1-2 Basic Profile Static View

The aim of this section is to describe the practices that compose a method in the context of a fictional software development organization. The KUALI-BEH Software Project Common Concepts and templates are used for defining practices and methods.

The context of the organization, the origin of the method and practices and their templates are presented.

E.2.1.1 KUALI-BEHSoftware Organizational Context

KUALI-BEHSoftware is a small software development entity with 20 employees. The organization has started to execute projects following the Basic Profile of ISO/IEC 29110-5-1-2 standard [4]. This standard is applicable to Very Small Entities (VSEs). VSEs are enterprises, organizations, departments or projects involving up to 25 people.

E.2.1.2 KUALI-BEHSoftware Method and Practices

ISO/IEC 29110-5-1-2 Basic Profile standard includes two processes: Project Management and Software Implementation. These processes specify a set of roles, work products and activities broken down in tasks. The organization has been using the activities, work products and roles described in both processes; however the practitioners have customized them in accordance with their experience and knowledge and have originated their own practices.

KUALI-BEHSoftware decides to create a repository of their practices in order to organize, consult and improve them. This repository is called Methods and Practices Infrastructure (KUALI-BEHSoftware-MPI). All practitioners of the organization participate in the creation of KUALI-BEHSoftware-MPI contributing with their knowledge and experience. The intention is to take advantage of the past projects execution, centralize the expertise and organize all this knowledge for future benefit and training.

The first method to be included in KUALI-BEHSoftware-MPI is related to their core business to develop a new software product.

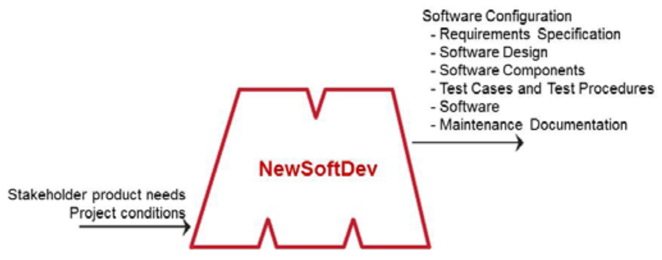
E.2.1.3 New Software Product Development Method Definition

In order to define the method for developing a new software product (NewSoftDev), the practitioners selected the following activities of ISO/IEC 29110-5-1-2 Basic Profile Software Implementation process as candidates for their practices:

- Software Requirements Analysis (SRA)
- Software Architectural and Detailed Design (SADD)
- Software Construction (SC)
- Software Integration and Tests (SIT)
- Product Delivery (PD)

The method and practice templates (see 8.3.2 and 8.3.3) and symbols (see 8.4) were used to document NewSoftDev and its practices. Table 26 shows the NewSoftDev method and Figure 18 presents the relationships among practice inputs and results.

Table 26 — NewSoftDev method

NewSoftDev	Method
Method for developing a new software product.	
 Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software - Maintenance Documentation NewSoftDev Stakeholder product needs Project conditions	
Purpose	
Systematically perform the analysis, design, construction, integration and tests activities for new software products according to the specified requirements.	
Input	Result
Stakeholders product needs:	*Software Configuration*

Statement of Work • *Product description: purpose of the product and general customer requirements.* • *Scope description of what is included and what is not* • *Project objectives* • *Deliverables list of products to be delivered to customer*	- *Requirements Specification* - *Software Design* - *Software Components* - *Software* - *Test Cases and Test Procedures* - *Test Report* - *Maintenance Documentation*
Project conditions:	
• *Project conditions established by the customer.* • *Schedule of the Project.* • *Identification of Project Risks.*	
Practices	
Software Requirements Analysis (SRA) *Software Architectural and Detailed Design (SADD)* *Software Construction (SC)* *Software Integration and Tests (SIT)* *Product Delivery (PD)*	

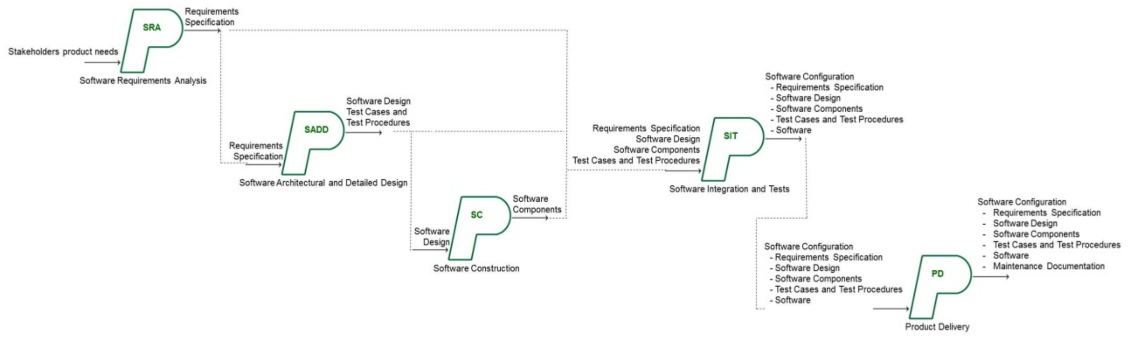
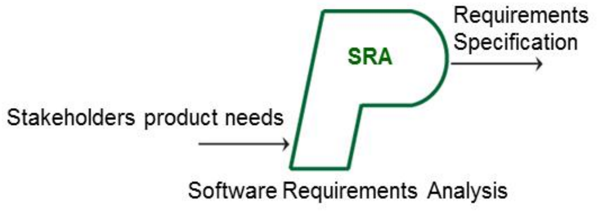

SRA
Requirements Specification
Stakeholders product needs
Software Requirements Analysis
SADD
Software Design
Test Cases and Test Procedures
Requirements Specification
Software Architectural and Detailed Design
SC
Software Components
Software Design
Software Construction
SIT
Requirements Specification
Software Design
Software Components
Test Cases and Test Procedures
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
Software Integration and Tests
PD
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
- Maintenance Documentation
Product Delivery

Figure 18 – NewSoftDev method practices inputs and results relationship

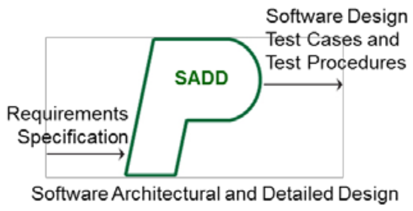
The practices are shown: Table 27 - SRA, Table 28 - SADD, Table 29 - SC, Table 30 - SIT and Table 31 - PD.

Table 27 — Software Requirements Analysis practice

SRA	Practice
Software Requirement Analysis	
 SRA Requirements Specification Stakeholders product needs Software Requirements Analysis	
Objective	
Define software requirements, analyze them for correctness and testability, get their approval by the customer, establish them as baseline and communicate them.	
Input	Result
Stakeholders product needs	*Requirements Specification*
Guide	
Activities	

1. Document or update the Requirements Specification. *Identify and consult information sources (customer, users, previous systems, documents, etc.) in order to get new requirements.* *Analyze the identified requirements to determinate the scope and feasibility.* *Generate or update the Requirements Specification.*
2. Verify and obtain approval of the Requirements Specification. *Verify the correctness and testability of the Requirements Specification and its consistency with the Stakeholders product needs. Additionally, review that requirements are complete, unambiguous and not contradictory.*
3. Validate and obtain approval of the Requirements Specification *Validate that Requirements Specification satisfies needs and agreed upon expectations, including the user interface usability.*
4. Incorporate the Requirements Specification to the Software Configuration in the baseline.
Knowledge and Skills
Knowledge and experience in eliciting, specifying and analyzing requirements.
Verification Criteria
Consistency between Requirements Specification and Stakeholders product needs.
Measures
Effort in hours to elaborate, document, verify and validate the Requirements Specification.

Table 28 — Software Architectural and Detailed Design practice

SADD	Practice
Software Architectural and Detailed Design	
 Software Design Test Cases and Test Procedures SADD Requirements Specification Software Architectural and Detailed Design	
Objective	
Develop the software architectural and detailed design, describing the Software Components and internal and external interfaces of them and establish the baseline of the software design. Prepare Test cases and test Procedure based on Requirements Specification	
Input	Result
Requirements Specification	*Software Design* *Test Cases and Test Procedures*
Guide	
Activities	
1. Understand Requirements Specification.	
2. Document or update the Software Design: *Analyze the Requirements Specification to generate the architectural design, its arrangement in subsystems and software components defining the internal and external interfaces. Describe in detail, the appearance and the behavior of the interface, based on the Requirements Specification in a way that resources for its implementation can be foreseen.* *Provide the detail of software components and their interfaces to allow the construction in an evident way.*	
3. Verify and obtain approval of the Software Design *Verify correctness of Software Design documentation, its feasibility and consistency with their Requirement Specification.*	
4. Establish or update Test Cases and Test Procedures for testing based on Requirements Specification. Customer provides testing data, if needed.	
5. Verify and obtain approval of the Test Cases and Test Procedures. *Verify consistency among Requirements Specification and Test Cases and Test Procedures.*	
6. Incorporate the Software Design and the Test cases and Test Procedures to the Software Configuration as part of the baseline. *Incorporate the Test Cases, and Test Procedures to the Project Repository.*	
Knowledge and Skills	

Knowledge and experience in the design of software architecture, planning and performing system tests.
Verification Criteria
Consistency between Software Design and Requirements Specification. *Consistency between Test Cases and Test Procedures and Requirements Specification.*
Measures
Effort in hours to elaborate, document and verify Software Design and Test Cases and Test Procedures.

Table 29 — Software Construction practice

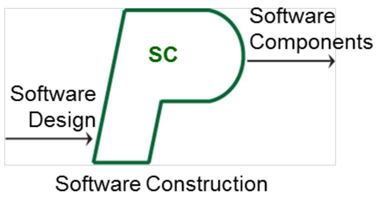
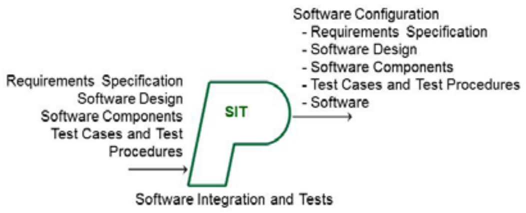
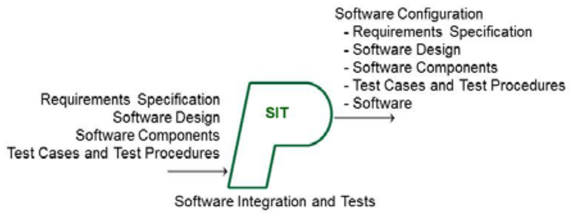
SC	Practice	
Software Construction		
 SC Software Components Software Design Software Construction		
Objective		
Produce Software Components defined by design. Define and perform unit test to verify the consistency with the design.		
Input		Result
Software Design		*Software Components*
Guide		
Activities		
1. Understand Software Design.		
2. Construct or update Software Components based on the detailed part of the Software Design.		
3. Design or update unit test cases and apply them to verify that the Software Components implements the detailed part of the Software Design.		
4. Correct the defects found until successful unit test (reaching exit criteria) is achieved.		
5. Incorporate Software Components to the Software Configuration as part of the baseline.		
Knowledge and Skills		
Knowledge and experience in programming and unit testing.		
Verification Criteria		
Consistency between Software Components and Software Design.		
Measures		
Effort in hours to understand the Software Design, to construct the Software Components, to design unit test cases, to apply them and to correct defects.		

Table 30 — Software Integration and Tests practice

SIT	Practice
Software Integration and Tests	
 Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software Requirements Specification Software Design Software Components Test Cases and Test Procedures SIT Software Integration and Tests	
Objective	

Produce software performing integration of the Software Components and verify using Test Cases and Test Procedures. Record results in the Test Report and correct defects.	
Input	Result
Requirements Specification *Software Design* *Software Components* *Test Cases and Test Procedures*	*Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components* - *Test Cases and Test Procedures* - *Software*
Guide	
Activities	
1. Understand Test Cases and Test Procedures. *Set or update the testing environment.*	
2. Integrate the Software using Software Components and update Test Cases and Test Procedures for integration testing, as needed. *3. Perform Software tests using Test Cases and Test Procedures and document results in Test Report.*	
4. Correct the defects found and perform regression test until exit criteria is achieved.	
5. Incorporate the Test Cases and Test Procedures, Test Report and Software to the Software Configuration as part of the baseline.	
Knowledge and Skills	
Knowledge and experience in programming, integration and system testing.	
Verification Criteria	
Consistency between Software and Test Cases and Tests Procedures.	
Measures	
Effort in hours to understand the Test Cases and Test Procedures, to perform the tests and integrate Software Components.	

Table 31 — Product Delivery practice

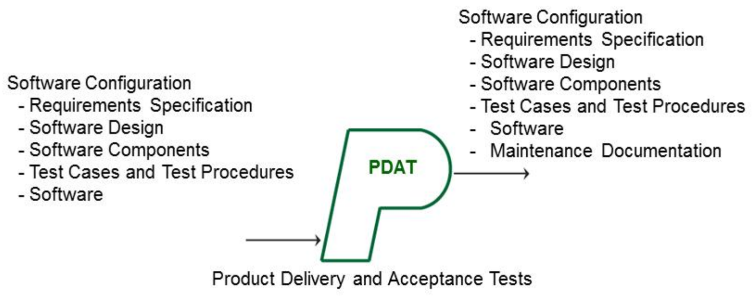
PD	Practice
Product Delivery	
 Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software Requirements Specification Software Design Software Components Test Cases and Test Procedures SIT Software Integration and Tests	
Objective	
Deliver the software product and applicable documentation to the customer.	
Input	Result
Software Configuration - *Requirements Specification* - *Software Design* - *Software Components* - *Test Cases and Test Procedures* - *Software*	*Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components* - *Test Cases and Test Procedures* - *Software* - *Maintenance Documentation*
Guide	
Activities	
1. Understand Software Configuration.	
2. Document the Maintenance Documentation or update the current one.	
3. Verify and obtain approval of the Maintenance Documentation. *Verify consistency of Maintenance Documentation with Software Configuration.*	

4. Incorporate the Maintenance Documentation as baseline for the Software Configuration.
5. Perform delivery according to delivery instructions agreed with the customer.
Knowledge and Skills
Knowledge and experience in software configuration and maintenance documentation elaboration.
Verification Criteria
Consistency between Maintenance Documentation and Software Configuration. *Fulfillment of the product delivery*
Measures
Effort in hours to deliver the software product, document and verify the Maintenance Documentation

E.2.1.4 Product Delivery and Acceptance Tests Practice Definition

Sometimes the KUALI-BEHSoftware practitioners are required by the customer to participate in acceptance tests. Therefore, they decide to define an extra practice that will be included as an individual practice in KUALI-BEHSoftware-MPI. This practice has to cover product delivery, acceptance test planning and performing. Table 32 shows the resulting practice.

Table 32 — Product Delivery and Acceptance Test practice

PDAT	Practice
Product Delivery and Acceptance Tests	
 Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software PDAT Software Configuration - Requirements Specification - Software Design - Software Components - Test Cases and Test Procedures - Software - Maintenance Documentation Product Delivery and Acceptance Tests	
Objective	
Perform the acceptance test and deliver the software product and applicable documentation to the customer.	
Input	Result
Software Configuration - *Requirements Specification* - *Software Design* - *Software Components* - *Test Cases and Test Procedures* - *Software*	*Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components* - *Test Cases and Test Procedures* - *Software* - *Maintenance Documentation*
Guide	
Activities	
1. Define an acceptance test strategy with the customer. *Participants, sessions, tasks to be performed and the management of issues and defects.*	
2. Elaborate or update the Test Procedures considering Requirements Specification and test strategy.	
3. Perform the acceptance tests and elaborate a Test Report.	
4. Correct defects agreed with the customer.	
5. Understand Software Configuration.	
6. Document the Maintenance Documentation or update the current one.	
7. Verify and obtain approval of the Maintenance Documentation. *Verify consistency of Maintenance Documentation with Software Configuration.*	

8. Incorporate the Maintenance Documentation, Test Procedures and Software as baseline for the Software Configuration.
9. Perform delivery according to delivery instructions agreed with the customer.
Knowledge and Skills
Knowledge and experience in acceptance testing, software configuration and maintenance documentation elaboration.
Verification Criteria
Consistency between Maintenance Documentation and Software Configuration. *Fulfillment of the product delivery.*
Measures
Effort in hours to planning and perform the acceptance test. *Effort in hours to deliver the software product, to document and to verify the Maintenance Documentation.*

E.2.2 ISO/IEC29110 5-1-2 Basic Profile Operational View

This section provides an example of the method enactment during a specific project execution. The aim is to illustrate the process and actions taken by the work team under particular circumstances of a project.

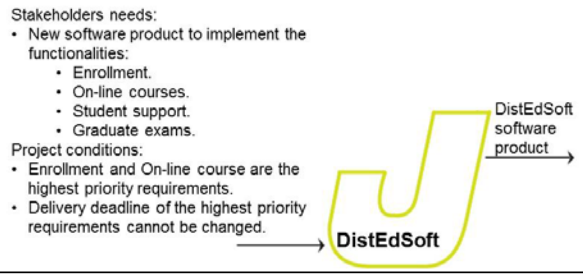
The example is based on the characteristics of KUALI-BEHSoftware organization. First, the context of the project is explained, and then the main steps of the method enactment are described. All the names mentioned in the example (organization, client, project and work team members) are fictional.

E.2.2.1 DistEdSoft Project Context

The School of Distance Education (DistEd) is a customer of KUALI-BEHSoftware organization. DistEd needs new software that will support on-line teaching operations. Both organizations agreed to start the DistEdSoft project to develop a new software product.

KUALI-BEHSoftware assigned seven practitioners to the project work team (WT). Before the beginning of the DistEdSoft project the DistEd school representatives and WT agreed on stakeholder needs and project conditions. Table 33 presents the details of the initial project template.

Table 33 — DistEdSoft project

DistEdSoft	**Software Project**
Project to develop the School of Distance Education software.	
 Stakeholders needs: • New software product to implement the functionalities: • Enrollment. • On-line courses. • Student support. • Graduate exams. Project conditions: • Enrollment and On-line course are the highest priority requirements. • Delivery deadline of the highest priority requirements cannot be changed. DistEdSoft DistEdSoft software product	
Stakeholder	
DistEd	
Start date	Finish date
02/07/2011	*08/15/2011*
Input	Result
Stakeholders needs: • *New software product to implement the functionalities:* o *Enrollment.*	*DistEdSoft software product*

o *On-line courses.* o *Student support.* o *Graduate exams.* *Project conditions:* • *Enrollment and On-line courses are the highest priority requirements.* • *Delivery deadline of the highest priority requirements cannot be changed.*	
Method	
Undefined.	
Work Team	
Olivia *Laura* *Jaime* *Nicolás* *Martín* *Ana* *Susana*	

E.2.2.2 DistEdSoft Project Method Enactment

Getting to know the new project conditions and stakeholder needs, the WT is ready to start DistEdSsoft project execution.

This section shows the examples of main steps performed by the WT during the DistEdSoft project method enactment.

WT selects a method

The WT consults the KUALI-BEHSoftware-MPI and selects the method for developing a new software product NewSoftDev (see Table 26).

WT adapts the NewSoftDev method to the project context

The WT analyzes the method and the project information in order to establish work to be done. The practitioners identify the following inputs:

- Stakeholder needs. The statement of work contains the general customer requirements:
  R1. Enrollment.
  R2. On-line courses.
  R3. Student support.
  R4. Graduate exams.

- Project conditions. The project particular conditions are:
  C1. R1 and R2 are the highest priority requirements established by the customer.
  C2. Delivery deadline of the highest (R1, R2) requirements cannot be changed.

The WT decides to divide the work into two increments, the software system covering R1 and R2 will be developed as the first increment and R3 and R4 as the second one. The decision is based on the requirements priority established by the customer.

The C2 condition is an important issue. To mitigate the risk of missing the first delivery deadline, the WT decides to prepare the test cases and procedures while the design is being developed.

The WT decides to adapt the method splitting the Software Architectural and Detailed Design (SADD) practice in two practices: Architectural and Detailed Design (ADD) and Tests Cases and Test Procedures Elaboration (TCTPE).

In summary, the WT makes two decisions: to repeat (iterate) method practices for two increments and to split the practice SADD in two practices for the first increment.

The adaptation of the selected method is carried out as follows:

> **Step 1**. SADD practice splitting. Figure 19 shows the result of this operation.
>
> **Step 2**. The resulting practices are instantiated as work units planned to be executed into DistEdSoft project. Each practice instance lifecycle is activated. Figures 20 and 21 show the adapted NewSoftDev method practice instances.

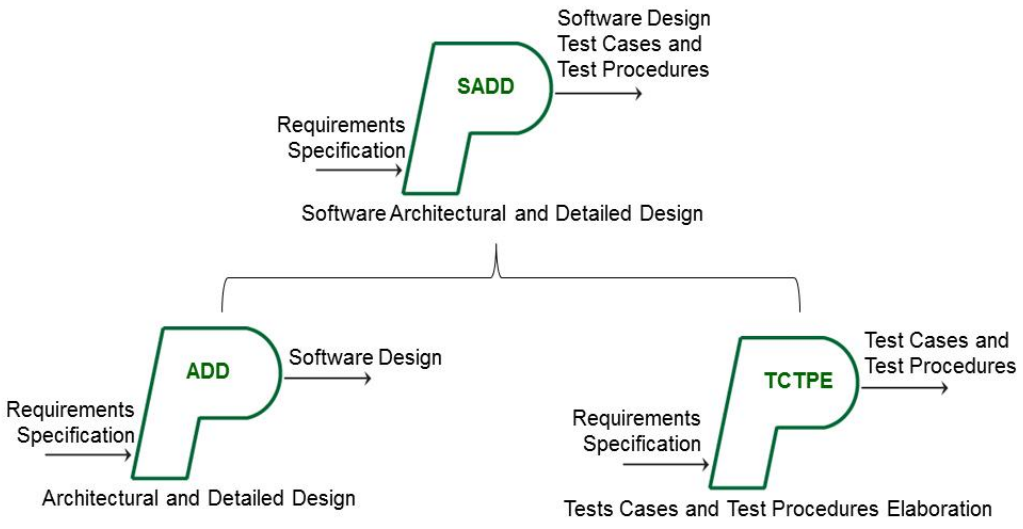


Figure 19 – Splitting operation of NewSoftDev method adaptation Step 1

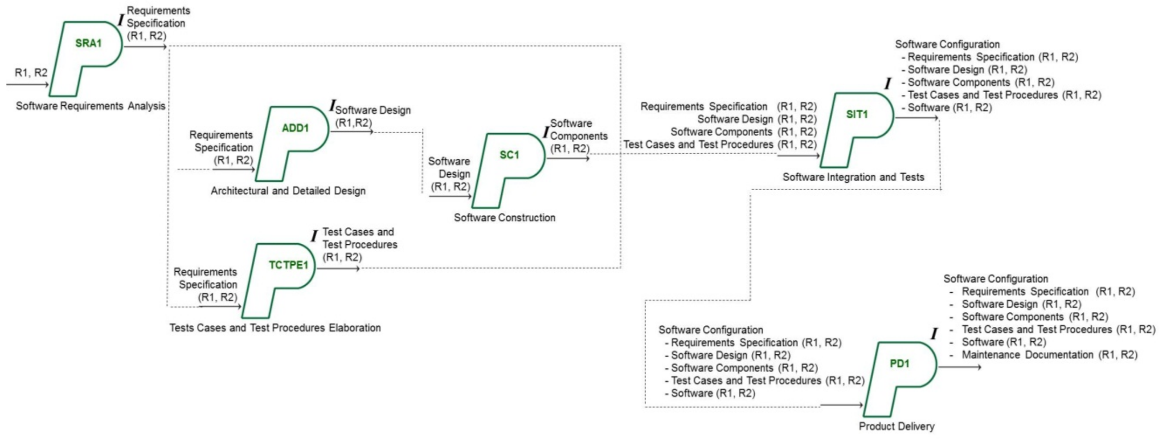

Requirements
Specification
(R1, R2)
SRA1
R1, R2
Software Requirements Analysis
ADD1
Software Design
(R1,R2)
Requirements
Specification
(R1, R2)
Architectural and Detailed Design
SC1
Software
Components
(R1, R2)
Software
Design
(R1, R2)
Software Construction
TCTPE1
Test Cases and
Test Procedures
(R1, R2)
Requirements
Specification
(R1, R2)
Tests Cases and Test Procedures Elaboration
Requirements Specification (R1, R2)
Software Design (R1, R2)
Software Components (R1, R2)
Test Cases and Test Procedures (R1, R2)
SIT1
Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
Software Integration and Tests
Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
PD1
Software Configuration
- Requirements Specification (R1, R2)
- Software Design (R1, R2)
- Software Components (R1, R2)
- Test Cases and Test Procedures (R1, R2)
- Software (R1, R2)
- Maintenance Documentation (R1, R2)
Product Delivery

Figure 20 – 1st increment practices instances of NewSoftDev method adaptation Step 2

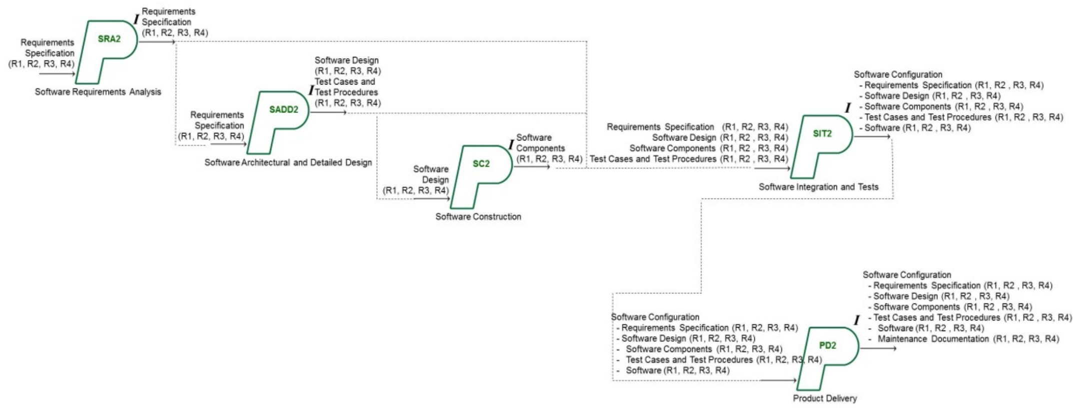

Requirements Specification (R1, R2, R3, R4)
SRA2
Requirements Specification (R1, R2, R3, R4)
Software Requirements Analysis
Requirements Specification (R1, R2, R3, R4)
SADD2
Software Design (R1, R2, R3, R4)
Test Cases and Test Procedures (R1, R2, R3, R4)
Software Architectural and Detailed Design
Software Design (R1, R2, R3, R4)
SC2
Software Components (R1, R2, R3, R4)
Software Construction
Requirements Specification (R1, R2, R3, R4)
Software Design (R1, R2 , R3, R4)
Software Components (R1, R2 , R3, R4)
Test Cases and Test Procedures (R1, R2 , R3, R4)
SIT2
Software Configuration
- Requirements Specification (R1, R2 , R3, R4)
- Software Design (R1, R2 , R3, R4)
- Software Components (R1, R2 , R3, R4)
- Test Cases and Test Procedures (R1, R2 , R3, R4)
- Software (R1, R2 , R3, R4)
Software Integration and Tests
Software Configuration
- Requirements Specification (R1, R2, R3, R4)
- Software Design (R1, R2, R3, R4)
- Software Components (R1, R2, R3, R4)
- Test Cases and Test Procedures (R1, R2, R3, R4)
- Software (R1, R2, R3, R4)
PD2
Software Configuration
- Requirements Specification (R1, R2 , R3, R4)
- Software Design (R1, R2 , R3, R4)
- Software Components (R1, R2 , R3, R4)
- Test Cases and Test Procedures (R1, R2 , R3, R4)
- Software (R1, R2 , R3, R4)
- Maintenance Documentation (R1, R2, R3, R4)
Product Delivery
I

Figure 21 – 2nd increment practices instances of NewSoftDev method adaptation Step 2

When the WT completed the method adaptation, the practitioners visualized the work to be done during the project on the NewSoftDev method enactment board (see Table 34). All the practice instances are in Instantiated state and the method enactment state is Adapted.

Table 34 — NewSoftDev method enactment board with practice instances at Instantiated column

DistEdSoft-NewSoftDev	Method Enactment Board	*02/07/2011*	*08/15/2011*
Input	Result	129 days left	
Stakeholders product needs:	*DistEdSoft Software Configuration*		
Statement of Work *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*	*- Requirements Specification* *- Software Design* *- Software Components Software* *- Test Cases and Test Procedures* *- Test Report* *- Maintenance Documentation*		
Project conditions:			
Project conditions established by the customer *C1. Enrollment and On-line courses are the highest priority requirements.* *C2. Delivery deadline of the highest priority requirements cannot be changed.*			

Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
				1st. increment				
1	*SRA1*							20
2	*ADD1*							20
3	*TCTPE1*							20
4	*SC1*							20
5	*SIT1*							20
6	*PD1*							20

2nd. Increment								
7	*SRA2*							20
8	*SADD2*							20
9	*SC2*							20
10	*SIT2*							20
11	*PD2*							20
							Total	220/1100
	Work Product / Conditions							
	Statement of Work (R1, R2, R3 and R4) –Agreed							

WT assigns inputs to practice instances

The WT has available inputs to assign them to some practice instances:

- The Stakeholders product needs R1 and R2 can be assigned as inputs to SRA1 instance (1st. increment). Table presents SRA1 instance board.

Table 35 — SRA1 instance board in Can-Start state

DistEdSoft -NewSoftDev-SRA1	Practice Instance Board		
Input		Result	
R1, R2		*Requirements Specification (R1, R2)*	
Work Team Practitioners		Measures	
Undefined		Estimated	Actual
		Undefined	*Undefined*
Activity Progress			
Activities	Progress	Responsible	Comments
1. Document or update the Requirements Specification.			
2. Verify and obtain approval of the Requirements Specification.			
3. Validate and obtain approval of the Requirements Specification.			
4. Incorporate the Requirements Specification to the Software Configuration in the baseline.			

Practice Instance States						
Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%
	X					

When the WT assigns the inputs to SRA1, this instance is included in Can-Start column of the NewSoftDev method enactment board. Can-Start is a practice instance state and Ready-to-Begin is a state associated to the method enactment.

WT chooses practice instances, estimates and agrees work distribution

The practice instances in the Can-Start state can be chosen by the WT. In this case, the practitioners estimate the SRA1 practice instance measures and agree who will execute them. SRA1 changes to the In-Execution state. Table 36 shows the current state, the responsible practitioners and the estimation associated to measures of the SRA1 instance.

Table 36 — SRA1 instance board in In-Execution state

DistEdSoft-NewSoftDev-SRA1	Practice Instance Board					
Input			Result			
R1, R2			*Requirements Specification (R1, R2)*			
Work Team Practitioners			Measures			
Olivia Laura Jaime Nicolás Martín Ana Susana Jaime			Estimated: *Effort: 46 man-hours Start date: 02/09/2011 Finish date:02/19/2011*		Actual: *Undefined*	
Activity Progress						
Activities	Progress		Responsible		Comments	
1. Document or update the Requirements Specification.			*Olivia Laura Jaime Nicolás Martín Ana Susana*			
2. Verify and obtain approval of the Requirements Specification.			*Olivia Laura Martín*			
3. Validate and obtain approval of the Requirements Specification.			*Susana*			
4. Incorporate the Requirements Specification to the Software Configuration in the baseline.			*Jaime*			
Practice Instance States						
Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%
		X				

When WT chooses SRA1, this instance is included at In-Execution column of the NewSoftDev method enactment board. In-Execution is a practice instance state and In-Progress is a state associated to method enactment.

The practitioners perform the activities and tasks included in the guide of practice SRA. The WT registers the progress of each activity in Activity Progress columns of the SRA1 practice instance board. When the result Requirements Specification (R1, R2) is produced, the responsible practitioners have to verify it according to guide activities. In this moment the SRA1 instance changes to In-Verification state.

WT produces verified results and collects data measures

When the instance of SRA1 practice produced its verified result and collects measures data, the SRA1 changed to Finished state (see Table 37).This change originates a Progress Snapshot of the NewSoftDev method enactment and the SRA1 instance moves to Finished column in the NewSoftDev method enactment board (see Table 38).

Table 37 — SRA1 instance board in Finished state

SDESoft-NewSoftDev-SRA1	Practice Instance Board					
Input			Result			
R1, R2			*Requirements Specification (R1, R2)*			
Work Team Practitioners			Measures			
Olivia Laura			Estimated		Actual	
Nicolás Martín Ana Susana Jaime			*Effort: 46 man-hours Start date: 02/09/2011 Finish date:02/19/2011*		*Effort: 77 man-hours Start date: 02/09/2011 Finish date:02/23/2011*	
Activity Progress						
Activities	Progress		Responsible		Comments	
1. Document or update the Requirements Specification.	*100%*		*Olivia Laura Jaime Nicolás Martín Ana Susana*			
2. Verify and obtain approval of the Requirements Specification.	*100%*		*Olivia Laura Martín*			
3. Validate and obtain approval of the Requirements Specification.	*100%*		*Susana*			
4. Incorporate the Requirements Specification to the Software Configuration in the baseline.	*100%*		*Jaime*			
Practice Instance States						
Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%
						X

Table 38 — NewSoftDev method enactment board with practice instances at Finished column

DistEdSoft-NewSoftDev	Method Enactment Board	*02/07/2011*	*08/15/2011*
Input	Result	119 days left	
Stakeholders product needs:	*DistEdSoft Software Configuration*		
Statement of Work *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*	*- Requirements Specification* *- Software Design* *- Software Components Software* *- Test Cases and Test Procedures* *- Test Report* *- Maintenance Documentation*		
Project conditions:			
Project conditions established by the customer *C1. Enrollment and On-line courses are the highest priority requirements.*			

C2. Delivery deadline of the highest priority requirements cannot be changed.								
Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
1st. increment								
1							*SRA1*	100
2	*ADD1*							20
3	*TCTPE1*							20
4	*SC1*							20
5	*SIT1*							20
6	*PD1*							20
2nd. increment								
7	*SRA2*							20
8	*SADD2*							20
9	*SC2*							20
10	*SIT2*							20
11	*PD2*							20
							Total	300/1100
	Work Product / Conditions							
	Statement of Work (R1, R2, R3 and R4) –Agreed *Requirements Specification (R1, R2) -Validated*							

WT assigns results as inputs

The result produced by SRA1 instance is the input to both instances ADD1 and TCTPE1. Therefore, the SRA1 instance result is assigned to ADD1 and TCTPE1 instances, so these instances change to Can-Start state. The WT decides to work on the execution of both instances simultaneously. So, five practitioners choose ADD1 and two choose TCTPE1. All practitioners make the needed estimations and ADD1 and TCTPE1 instances change to In-Execution state. Tables 39 and 40 show ADD1 and TCTPE1 practice instance boards respectively.

Table 39 — ADD1 instance board in In-Execution state

DistEdSoft -NewSoftDev-ADD1	Practice Instance Board		
Input		Result	
Requirements Specification (R1, R2)		*Software Design (R1, R2)*	
Work Team Practitioners		Measures	
Laura *Nicolás* *Ana* *Susana* *Olivia*		Estimated	Actual
		Effort: 104 man-hours *Start date: 02/26/2011* *Finish date:03/19/2011*	*Effort: 140 man-hours* *Start date: 02/26/2011* *Finish date:03/23/2011*
Activity Progress			
Activities	Progress	Responsible	Comments

1. Understand Requirements Specification.		*Laura Nicolás* *Ana Susana* *Olivia*	
2. Document or update the Software Design		*Laura Nicolás* *Ana Susana* *Olivia*	
3. Verify and obtain approval of the Software Design.		*Nicolás* *Laura*	
6. Incorporate the Software Design to the Software Configuration as part of the baseline.		*Susana*	

Practice Instance States						
Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%
		X				

Table 40 — TCTPE1 instance board in In-Execution state

DistEdSoft -NewSoftDev-TCTPE1	Practice Instance Board		
Input		Result	
Requirements Specification (R1, R2)		*Test Cases and Test Procedures (R1, R2)*	
Work Team Practitioners		Measures	
Martín *Jaime*		Estimated	Actual
		Effort: 39 man-hours *Start date: 02/26/2011* *Finish date:02/05/2011*	*Effort: 54 man-hours* *Start date: 02/26/2011* *Finish date:03/07/2011*
Activity Progress			
Activities	Progress	Responsible	Comments
4. Establish or update Test Cases and Test Procedures for testing based on Requirements Specification. Customer provides testing data, if needed.		*Martín* *Jaime*	
5. Verify and obtain approval of the Test Cases and Test Procedures.		*Martín* *Jaime*	
6. Incorporate the Test cases and Test Procedures to the Software Configuration as part of the baseline		*Martín* *Jaime*	

Practice Instance States						
Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%
		X				

The ADD1 and TCTPE1 instances state changes are applied in the NewSoftDev method enactment board, moving these instances to the In-Execution column of the In-Progress state. Table 41 shows this movement in the method enactment board.

Table 41 — NewSoftDev method enactment board with ADD1 and TC instance at In Execution column

DistEdSoft-NewSoftDev	Method Enactment Board	*02/07/2011*	*08/15/2011*
Input	Result	118 days left.	
Stakeholders product needs: *Statement of Work* *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*	*DistEdSoft Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components Software* - *Test Cases and Test Procedures* - *Test Report* - *Maintenance Documentation*		
Project conditions: *Project conditions established by the customer* *C1. Enrollment and On-line courses are the highest priority requirements.* *C2. Delivery deadline of the highest priority requirements cannot be changed.*			

Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
			1st. increment					
1							*SRA1*	100
2			*ADD1*					60
3			*TCTPE1*					60
4	*SC1*							20
5	*SIT1*							20
6	*PD1*							20
			2nd. increment					
7	*SRA2*							20
8	*SADD2*							20
9	*SC2*							20
10	*SIT2*							20
11	*PD2*							20
							Total	380/1100
	Work Product / Conditions							
	Statement of Work (R1, R2, R3 and R4) –Agreed *Requirements Specification (R1, R2) -Validated*							

The ADD1, TCTPE1, SC1, SIT1 and PD1 instances continue their lifecycle to reach the Finished state during the 1st increment.

WT adapts the NewSoftDev method for the second time

When the SRA1, ADD1, TCTPE1, SC1, SIT1 and PD1 practice instances of the 1st increment are finished, the Progress Snapshot of the method enactment is analyzed by the WT.

At this moment, the customer has requested that the practitioners of WT participate in the acceptance tests. Therefore, the WT decides to adapt the planned practice instances performing the substitution operation. The Product Delivery (PD) practice is to be substituted by Product Delivery and Acceptance Tests (PDAT) practice. The product delivery, planning and performing of acceptance testing will be performed through PDAT practice (see Table 32, section E.2.1.4).

The adaptation of the NewSoftDev method is carried out as follows:

Step 1. The PD practice is substituted by PDAT. Figure 22 shows this operation.

Step 2. The 2nd increment practices are instantiated according to the substitution operation of Step 1. Every practice instance lifecycle is activated. Figure 23 shows the NewSoftDev method practice instances.

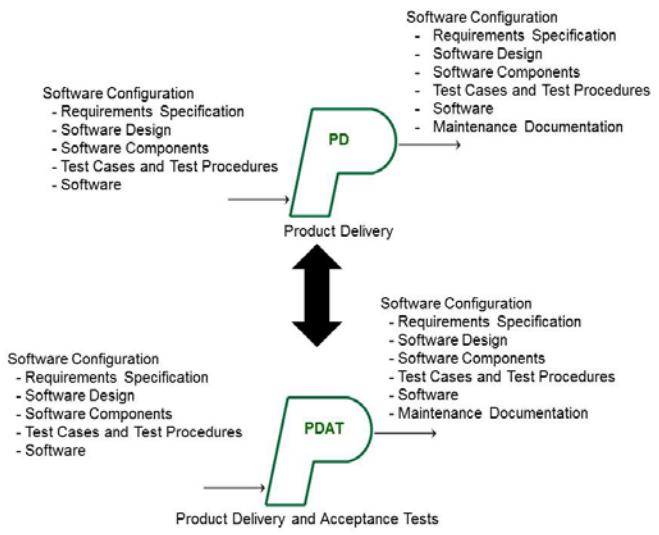

Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
PD
Product Delivery
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
- Maintenance Documentation
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
PDAT
Product Delivery and Acceptance Tests
Software Configuration
- Requirements Specification
- Software Design
- Software Components
- Test Cases and Test Procedures
- Software
- Maintenance Documentation

Figure 22 – Substitution operation of NewSoftDev method adaptation

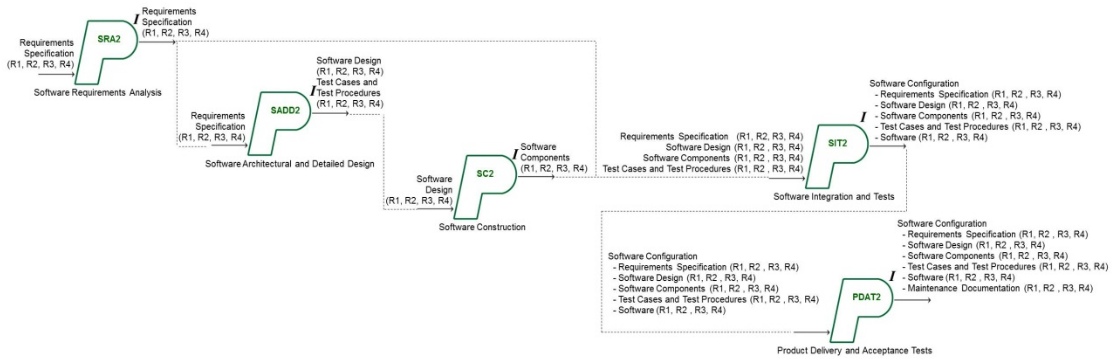

Requirements Specification (R1, R2, R3, R4)
SRA2
Software Requirements Analysis
Requirements Specification (R1, R2, R3, R4)
Requirements Specification (R1, R2, R3, R4)
SADD2
Software Architectural and Detailed Design
Software Design (R1, R2, R3, R4)
Test Cases and Test Procedures (R1, R2, R3, R4)
Software Design (R1, R2, R3, R4)
SC2
Software Construction
Software Components (R1, R2, R3, R4)
Requirements Specification (R1, R2, R3, R4)
Software Design (R1, R2 , R3, R4)
Software Components (R1, R2 , R3, R4)
Test Cases and Test Procedures (R1, R2 , R3, R4)
SIT2
Software Integration and Tests
Software Configuration
- Requirements Specification (R1, R2 , R3, R4)
- Software Design (R1, R2 , R3, R4)
- Software Components (R1, R2 , R3, R4)
- Test Cases and Test Procedures (R1, R2 , R3, R4)
- Software (R1, R2 , R3, R4)
Software Configuration
- Requirements Specification (R1, R2 , R3, R4)
- Software Design (R1, R2 , R3, R4)
- Software Components (R1, R2 , R3, R4)
- Test Cases and Test Procedures (R1, R2 , R3, R4)
- Software (R1, R2 , R3, R4)
PDAT2
Product Delivery and Acceptance Tests
Software Configuration
- Requirements Specification (R1, R2 , R3, R4)
- Software Design (R1, R2 , R3, R4)
- Software Components (R1, R2 , R3, R4)
- Test Cases and Test Procedures (R1, R2 , R3, R4)
- Software (R1, R2 , R3, R4)
- Maintenance Documentation (R1, R2 , R3, R4)

Figure 23 – 2nd increment practices instances of NewSoftDev method adaptation Step

The column Adapted of NewSoftDev method enactment board has to be changed to include the resulting practice instances of Step 2 (see Table 42).

Table 42 — NewSoftDev method enactment board with practice instances of 2nd increment

DistEdSoft-NewSoftDev	Method Enactment Board	*02/07/2011*	*08/15/2011*

Input	Result	
Stakeholders product needs:	*DistEdSoft Software Configuration* *- Requirements Specification* *- Software Design* *- Software Components Software* *- Test Cases and Test Procedures* *- Test Report* *- Maintenance Documentation*	58 days left
Statement of Work *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*		
Project conditions:		
Project conditions established by the customer *C1. Enrollment and On-line courses are the highest priority requirements.* *C2. Delivery deadline of the highest priority requirements cannot be changed.*		

Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
				1st. increment				
1							*SRA1*	100
2							*ADD1*	100
3							*TCTPE1*	100
4							*SC1*	100
5							*SIT1*	100
6							*PD1*	100
				2nd. increment				
7	*SRA2*							20
8	*SADD2*							20
9	*SC2*							20
10	*SIT2*							20
11	*PDAT2*							20
							Total	700/1100

	Work Product / Conditions
	Statement of Work (R1, R2, R3 and R4) –Agreed *Software Configuration* *- Requirements Specification (R1, R2) –Validated* *- Software Design (R1, R2) –Validated* *- Software Components (R1, R2) - Corrected* *- Test Cases and Test Procedures (R1, R2)- Verified* - Software *(R1, R2) -Corrected* - Maintenance Documentation *(R1, R2) –Verified*

WT produces the software product

Once all practice instances have completed their lifecycles, the method enactment is finished and the DistEdSoft software product is delivered to the customer. Table 43 shows the NewSoftDev method enactment board with all practices instances in the Finished state

Table 43 — Final NewSoftDev method enactment board

DistEdSoft-NewSoftDev	Method Enactment Board	*02/07/2011*	*08/15/2011*
Input	Result	0 days left	
Stakeholders product needs:	*DistEdSoft Software Configuration* - *Requirements Specification* - *Software Design* - *Software Components Software* - *Test Cases and Test Procedures* - *Test Report* - *Maintenance Documentation*		
Statement of Work *General customer requirements:* *R1. Enrollment.* *R2. On-line courses.* *R3. Student support.* *R4. Graduate exams.*			
Project conditions:			
Project conditions established by the customer *C1. Enrollment and On-line courses are the highest priority requirements.* *C2. Delivery deadline of the highest priority requirements cannot be changed.*			

Enactment States								
	Adapted	Ready to Begin	In Progress			Progress Snapshot		Global Progress
	Instantiated 20%	Can Start 40%	In Execution 60%	In Verification 80%	Stand By N/A	Cancelled N/A	Finished 100%	
				1st. increment				
1							*SRA1*	100
2							*ADD1*	100
3							*TCTPE1*	100
4							*SC1*	100
5							*SIT1*	100
6							*PD1*	100
				2nd. increment				
7							*SRA2*	100
8							*SADD2*	100
9							*SC2*	100
10							*SIT2*	100
11							*PDAT2*	100
							Total	1100/1100

	Work Product / Conditions
	Statement of Work (R1, R2, R3 and R4) –Agreed *Software Configuration* *- Requirements Specification (R1, R2, R3, R4) –Validated* *- Software Design (R1, R2, R3, R4) –Validated* *- Software Components (R1, R2, R3, R4) - Corrected* *- Test Cases and Test Procedures (R1, R2, R3, R4)- Verified*

	- Software *(R1, R2, R3, R4) -Corrected* - Maintenance Documentation *(R1, R2, R3, R4) –Verified*

References

1. A Foundation for the Agile Creation and Enactment of Software Engineering Methods, http://www.omg.org/cgi-bin/doc?ad/2011-6-26 16/05/11
2. Oktaba, H., García, F., Piattini, M., Pino, F., Ruíz, F., Alquicira, C.: Software Process Improvement: The COMPETISOFT Project. IEEE Computer, Vol 40, No. 10 (2008)
3. Mexican National Standard NMX-I-059-NYCE-2005 Modelo de Procesos para la Industria del Software (MoProSoft) (2005)
4. Standard ISO/IEC 29110-5-1-2 Software engineering -- Lifecycle profiles for Very Small Entities (VSEs) – Management and Engineering Guide: Generic profile group: Basic Profile, http://standards.iso.org/ittf/PubliclyAvailableStandards/c051153_ISO_IEC_29110-5-1-2_2011.zip 07/08/12 (2011)
5. Standard ISO 9000:2005 Quality management systems -- Fundamentals and vocabulary (2005)
6. Standard ISO/IEC 12207:2008 Systems and software engineering -- Software life cycle processes (2008)
7. Oxford Concise Oxford English Dictionary © 2008 Oxford University Press (2008)
8. Thesaurus Roget's 21st Century Thesaurus © 2011, Third Edition Philip Lief Group (2011)
9. Project Management Institute. A Guide to the Project Management Body of Knowledge: PMBOK Guide. Third Edition, Newtown Square, Pennsylvania, Project Management Institute (2004)
10. Tautz, C. and Wangenheim, C.G., REFSENO: A Representation Formalism for Software Engineering Ontologies, in Technical IESE-Report 015.98/E, Fraunhofer Institute for Experimental Software Engineering (1998)
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional (1995)
12. Alexander, C.: The Timeless Way of Building. Oxford University Press (1979)
13. The Scrum Guide -The Definitive Guide to Scrum: The Rules of the Game, developed and sustained by Ken Schwaber and Jeff Sutherland, http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011.pdf 07/08/12 (2011)