UNIVERSIDAD NACIONAL DE COLOMBIA

SEDE MEDELLÍN

FACULTAD DE MINAS

SOFTWARE ENGINEERING: METHODS, MODELING AND TEACHING

VOL. 3

SOFTWARE ENGINEERING:

METHODS, MODELING, AND TEACHING, VOL. 3

Carlos Zapata, Luis Fernando Castro (Ed.)

SOFTWARE ENGINEERING:

METHODS, MODELING, AND TEACHING, VOL. 3

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS

Medellín, Colombia, 2014

005.1
S63 Software engineering : methods, modeling and teaching / editores Carlos Mario Zapata y Luis Fernando Castro. -- Medellín : Universidad Nacional de Colombia. Facultad de Minas, 2014.
Volumen 3 (96 páginas) : ilustraciones.

ISBN : 978-958-775-080-5

1. INGENIERÍA DE SOFTWARE. 2. DESARROLLO DE PROGRAMAS PARA COMPUTADOR. 3. MÉTODOS DE ENSEÑANZA. I. Zapata, Carlos Mario, editor. II. Castro, Luis Fernando, editor.

Catalogación en la publicación Universidad Nacional de Colombia

Software Engineering: Methods, Modeling, and Teaching, Vol.3

Primera edición: Medellín, agosto de 2014
ISBN : 978-958-775-080-5
Caratula: Corporación Cidenet

Coordinación editorial
Centro Editorial de la Facultad de Minas
Universidad Nacional de Colombia, Sede Medellín
Carrera 80 No. 65-223, Bloque M9-103
Teléfono: (57-4) 425 53 43
Correo-e: ceditorial_med@unal.edu.co

CONTENIDO

Preface

C.M. Zapata
Universidad Nacional de Colombia

L. Castro
Universidad Del Quindío

S. Huang
Florida Atlantic University

P.-W. Ng
Ivar Jacobson International

SEMAT (Software Engineering Method and Theory) has reached a new stage. By the date of this book, SEMAT officially became a new OMG standard. These are good news for the software engineering as a discipline. In this book, we are joining the SEMAT celebration by doing what we do best: promoting the usage of the SEMAT kernel in as many ways as we can. Since we Latin American researchers are also happy with our new OMG standard, we devote our effort to promote the work on the SEMAT kernel. This book is the direct result of this effort. In twelve Chapters, we work on the foundations of the software engineering theory, the creation of SEMAT-kernel-based models, and the promotion of teaching by using the ideas behind the SEMAT initiative.

This third volume of *Software Engineering—Methods, Modeling, and Teaching* was written by authors coming from five countries: USA, China, Mexico, Colombia, and Brazil. Several points of view related to the SEMAT initiative are promoted and a lot of different theories and examples are prepared for the reader. Whether you belong to the Academic world or the Industry Practitioners, this book is very helpful for you as the seed of the usage of this new OMG standard.

This book is divided into three main parts: (i) theoretical development, (ii) method and practice representation, and (iii) teaching. If you belong to the Academy, probably you will be interested in Parts (i) and (iii) more than Part (ii). If you are a practitioner, maybe the Part (ii) will be more helpful for you. Regardless of your point of view, we strongly suggest the reading of the fundamental topics about the SEMAT kernel included in this Preface, since these ideas are cross-cutting to the entire book. Hopefully, we think you will find what you looking for inside this book: different ways to understand, practice, and use the main SEMAT ideas.

OVERVIEW AND KEY CONCEPTS OF ESSENCE

In this Section we give brief overview of Essence with focuses on describing its key concepts (Jacobson *et al.*, 2012; 2013), namely alphas and alpha states and their applications.

The Essence constitutes the kernel along with the language supporting the kernel. The Essence kernel includes a stripped-down, light-weight set of elements that are universal to all software engineering endeavors. Through states defined for its elements, the Essence kernel provides a novel and effective instrument for reasoning about the progress and health of the software development endeavors in a method independent way. It helps practitioners to understand where they are, point out what they should do next and, suggest what they should improve and where. The kernel provides a common ground for understanding and describing the commonalities and diversities of software engineering practices, and this common ground is realized as a universal set of elements called alphas. Presenting the essence of software engineering in this way enables us to build our knowledge on top of what we have known and learnt, and to apply and reuse gained knowledge across different application domains and software systems of differing complexity.

The benefits of using these kernel elements are that they are harvested from existing common practices in industry, so they already exist and prevalent in software endeavor. They are what we always have (*e.g.*, teams and work), what we always do (*e.g.*, specify and implement), and what we always produce (*e.g.*, software systems) when we develop software. Even without a well-defined method, the Essence kernel can be used to monitor the progress and health of specific software endeavor, and to analyze the strengths and weaknesses of a team way of working.

Alphas

Alphas (Abstract-Level Progress Health Attribute) are one of the core concepts of Essence. Essence uses an object-oriented approach to identify typical dimensions of software engineering challenges. These objects are called alphas. Each alpha represents a key dimension of endeavors. Alphas are separated into three different areas of concerns, which are Customer, Solution, and Endeavor (see Figure P1).

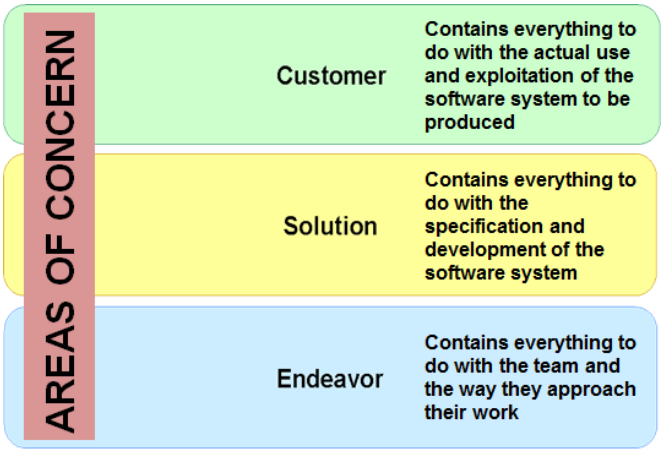


Figure P1. Three Areas of Concerns

The Essence kernel has identified seven method-independent alphas that are common to software development, namely Opportunity, Stakeholders, Requirements, Software System, Work, Team, and Way-of-Working. From Essence OMG specification (OMG, 2014), they are defined as follows:

- Opportunity is the set of circumstances that makes it appropriate to develop or change a software system
- Stakeholders: The people, groups, or organizations who affect or are affected by a software system
- Requirements: What the software system must do to address the opportunity and satisfy the stakeholders
- Software System: A system made up of software, hardware, and data that provides its primary value by the execution of the software
- Work: Activity involving mental or physical effort done in order to achieve a result
- Team: A group of people actively engaged in the development, maintenance, delivery, or support of a specific software system
- Way-of-Working: The tailored set of practices and tools used by a team to guide and support their work

Alphas are agnostic to practitioner's chosen practices and methods. For example, the team performs and plans work does not imply any specific order in that they perform and plan the work. These seven alphas and their relationships are shown in Figure P2.

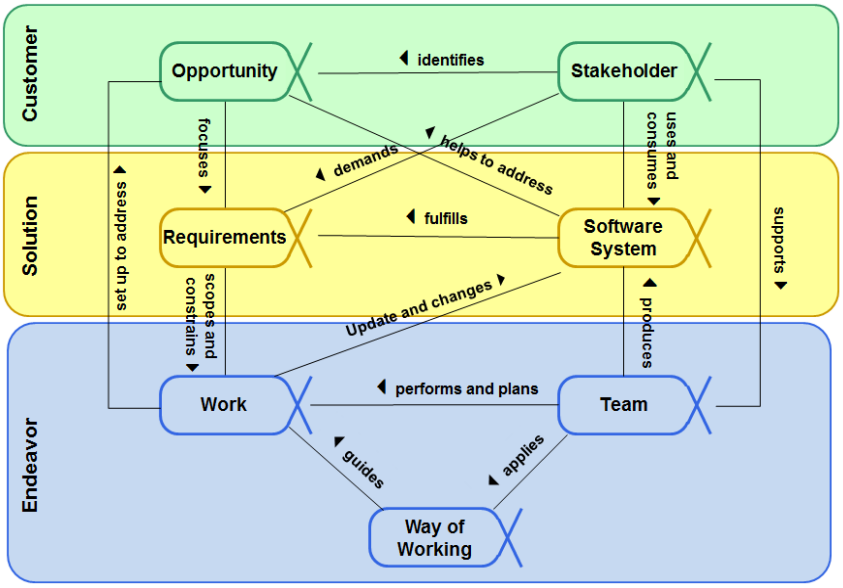


Figure P2. Alphas and their relationships

Alpha States

Each alpha has states that provide guidance for development teams to achieve progress along these dimensions and to detect risks and problems early.

The alpha states are the following:

- Opportunity: *Identified, Solution Needed, Value Established, Viable, Addressed*, and *Benefit Accrued*.
- Stakeholders: *Recognized, Represented, Involved, In Agreement, Satisfied for Deployment and Satisfied in Use*.
- Requirements: *Conceived, Bounded, Coherent, Acceptable, Addressed and Fulfilled*.
- Software System: *Architecture Selected, Demonstrable, Usable, Ready, Operational and Retired.*
- Team: *Seeded, Formed, Collaborating, Performing, and Adjourned*.
- Work: *Initiated, Prepared, Started, Under Control, Concluded and Closed*.

- Way of Working: *Principles Established, Foundation Established, In Use, In Place, Working Well and Retired*.

Essence kernel provides a detailed checklist for each alpha and their states (see Figure P3). An example of the alpha *opportunity* is shown in Figure P4.

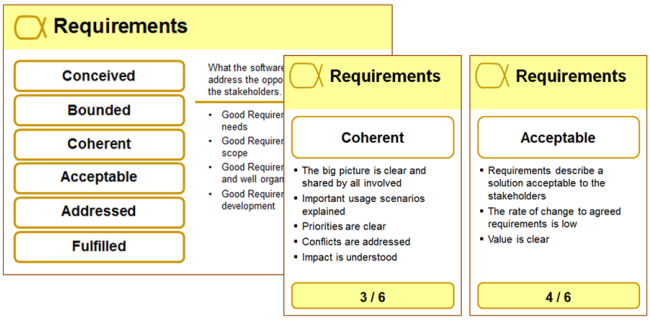


Figure P3: Requirement Alpha and Its Selected States

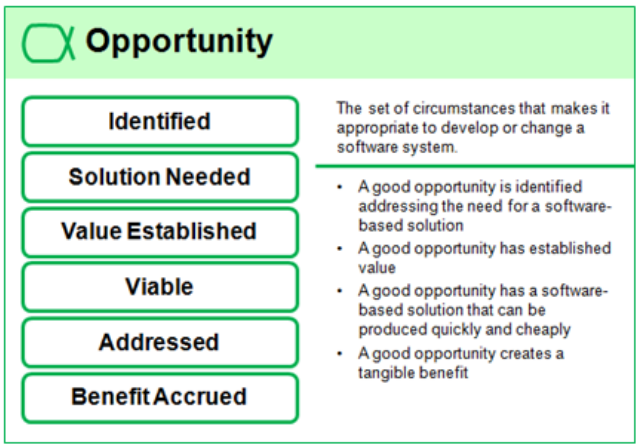


Figure P4. The alpha *opportunity* (Jacobson *et al.*, 2013)

As illustrated in Figure P5, both the kernel alphas and their states can be represented as a deck of cards. These cards may be used for monitoring the progress and health of software development endeavors. The monitoring may also be supported by spider graphs (SEMAT accelerator, 2014) in Figure 5 that illustrates the common progress of all alphas.

Figure P5. Cards and progress of the alphas

Alpha states can be also arranged in a roadmap combining the alphas and the project milestones, as shown in Figure P6.

Besides the alphas ("things we always work with"), the Essence kernel defines activity spaces ("things we always do"), also classified into the three areas of concern, as shown in Figure P7.

The other element Essence kernel defines with a close set of values is *competency*, encompassing abilities, capabilities, attainments, knowledge, and skills necessary to do a certain kind of work. The set of competencies defined by the Essence kernel is depicted in Figure P8.

The Essence kernel does not include any instances of other elements—like the ones defined in Table P1—since such instances are acquired by the elements in the context of a specific practice (Jacobson *et al.*, 2013). The most important associations among elements are depicted in Figure P9.

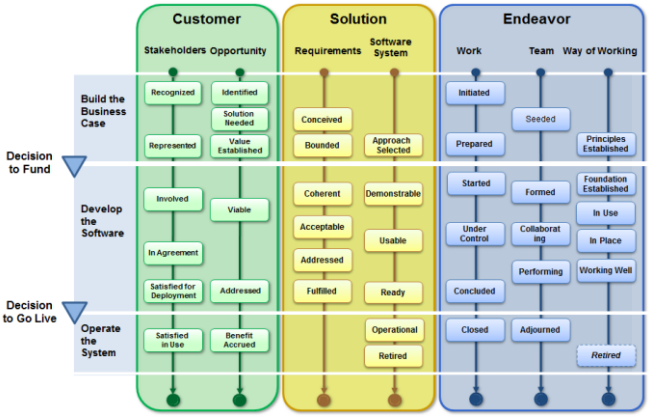


Figure P6. Alpha states arranged in a roadmap.

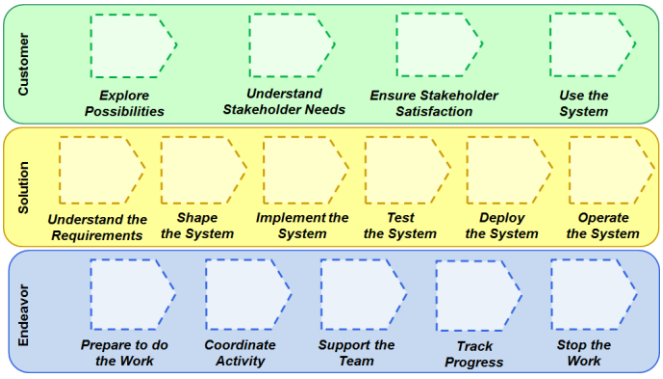


Figure P7. Activity spaces.

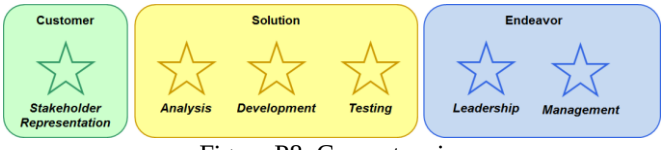
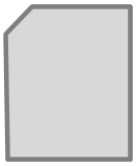
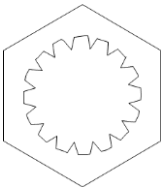
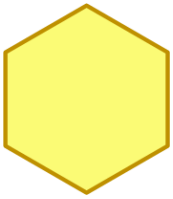


Figure P8. Competencies

Table P1. Essence kernel elements

Element	Symbol	Additional Information
Alpha		Provide descriptions of the kind of things that a team will manage, produce, and use in the process of developing, maintaining, and supporting good software. SEMAT has identified seven alphas: Opportunity, Stakeholders, Requirements, Software System, Work, and Way of working and Team.
Alpha state		Is the progress and health of an alpha. Represented in a checklist. For example the Alpha *Opportunity* has the following states: Identified, Solution needed, Value established, Viable, Addressed and Benefit accrued.
Activity space		Represent the essential things which have to be done to develop good software. For example, the Activity space called System use, which allows use of system in a live environment to benefit the stakeholders.
Activity		Define one or more kinds of works items and gives guidance how to perform these.
Competency		Encapsulate the ability to perform an activity involving the performance of work within the software engineering process. For example the competency *testing* encapsulates the ability to test a system verifying it is usable and it meets the requirements.
Work Product		Is an artifact of value and relevance for a software engineering endeavor. A work product may be a document, a piece of software, a creation of a test environment, or the delivery of a training course.
Kernel		A set of elements used to form a common ground for describing a software engineering endeavor.
Practice		Is considered as an element group which names all Essence elements necessary to express the desired work guidance with a specific objective.

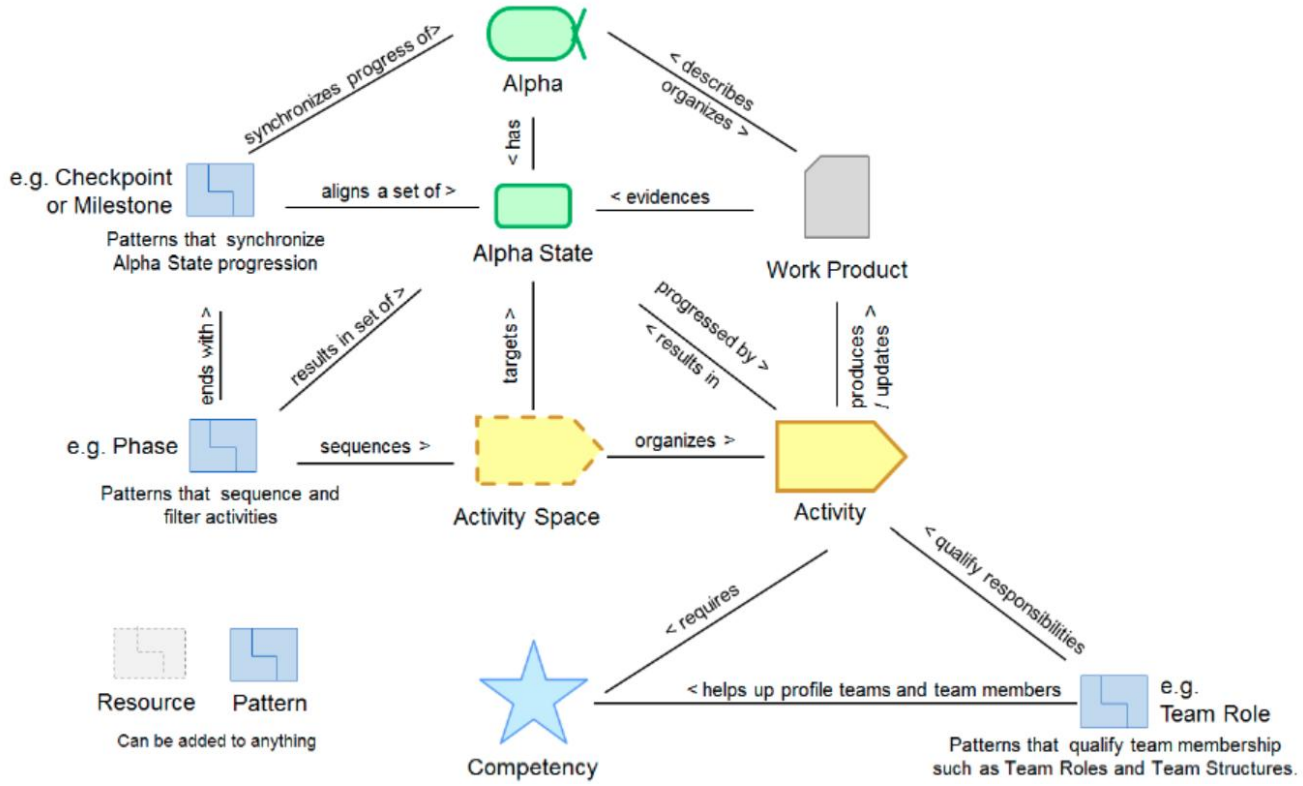


Figure P8. Competencies

REFERENCES

OMG. 2014. *Essence Submission*. Available http://www.omg.org/spec/Essence/

Jacobson, I., Ng, P.-W., McMahon, P., Spence, I. & Lidman, S. 2012b. The Essence of Software Engineering: the SEMAT kernel. *Communications of the ACM* (10): 42-49.

Jacobson, I., Ng, P.-W., McMahon, P., Spence, I., & Lidman, S. 2013. *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley.

SEMAT accelerator. 2014. Available https://sematacc-meteor-com-ddp.meteor.com/

This page intentionally left blank

Part I: Theoretical development

We must not forget that the wheel is reinvented so often because it is a very good idea; I've learned to worry more about the soundness of ideas that were invented only once.

— David L. Parnas (Why Software Jewels are Rare, IEEE Computer, 2/96).

This page intentionally left blank

Chapter #1

An executable pre-conceptual schema for a software engineering general theory

C.M. Zapata
Universidad Nacional de Colombia

1 INTRODUCTION

According to Sjøberg *et al.* (2008), we can build theories as a way to gain and gather general knowledge. General theories are proposed in several knowledge fields. General theories are so ancient than the Science itself. For example, Keynes (1937) proposed a general theory of employment, trying to assess the economical behavior of employment under some constraints; such a theory was then reviewed by Rueff (1947). In the field of physics, Green and Laws (1433) proposed a general theory of rods based on thermodynamics principles. Bartels (1968) proposed a general theory of marketing, trying to gather together some previous approaches and base theories about the subject. Köhler (1979) tried to explain armaments by using a model resembling the general theory about this subject. Recently, Frickel and Gross (2005) promoted the debate regarding the general theory about scientific/intellectual movements.

Software engineering has also been enhanced with several attempts of general theories. Trachtenberg (1990) proposed a general theory of software reliability by using a mathematical model, Zerangue (1993) made the first attempt to discuss the need for a general theory of software engineering, and Mitchell *et al.* (1997) establish some guidelines for working with stakeholders by defining the constructs of a theory about the subject. Some other attempts are authored by Coleman and O'Connor (2007), related to the software process improvement, Adolph *et al.* (2012), for software process management, and Medeiros and Horta (2013) for relief-based perspectives. Be advised that most of the previously described attempts—with the exception of the Zerangue's work—are related to specific software engineering subjects. In addition, only some model-based general theories are formally stated, leading to a weak—or even absent—performance on proof theory.

As the previous review reflect, the novelty of software engineering in Sciences show its infancy in building general theories, compared to other ancient Sciences like management or physics.

Jacobson *et al.* (2013) have been promoting the SEMAT (Software Engineering Methods and Theory) initiative, for establishing the need of refunding the software engineering. Some of the SEMAT ideas are used by Ekstedt (2013) and Johnson *et al.* (2013) in order to promote the guidelines of the creation of a general software engineering theory. By following the ideas behind the SEMAT initiative, in this Chapter we use the so-called executable pre-conceptual schemas (2011) for representing the main constructs of a general theory and instantiating a general theory about software engineering. We also use the ideas promoted by Sjøberg *et al.* (2008) and Ekstedt (2013) for defining the concepts related to the general theory. The instances related to software engineering are based on the SEMAT initiative and the related kernel (Jacobson *et al.*, 2013).

The remainder of this Chapter has the following structure: in Section 2 we propose the material and methods; in Section 3 we propose the general theory about software engineering and we exemplify it; in Section 4 we discuss the results; finally, in Section 5 we conclude and state the future work.

2 MATERIAL AND METHODS

2.1 *Methodology for building the general theory*

Ekstedt (2013) establishes a 3-step methodology for building the general theory, and he provides two examples of the methodology usage. Steps are the following:

- Scope the theory: In our case, the scope will be the software engineering itself. We will start by explaining the phenomena surrounding the software engineering and we will use the SEMAT kernel elements as constructs for our theory.

- Find casualty: The casualty will be related to the several propositions we will have in the theory. Properties and dependencies should be provided as an explanation for each proposition. In this step we decide on using pre-conceptual schemas in order to formalize the entire theory.

- Iterate, integrate, and keep consistent: Some iteration will be considered, since we will define some constructs and their relationships. The terms will be always the same, since we will use the SEMAT kernel elements, so terminology will help to keep consistency.

2.2 *Executable pre-conceptual schemas*

According to Zapata *et al.* (2006), pre-conceptual schemas are diagrams for representing knowledge about any domain. The main advantages of using pre-conceptual schemas are related to their proximity to both the natural language and the formal logic. Also, the possibility of defining operations among concepts by using the so-called executable pre-conceptual schemas (Zapata *et al.*, 2011) is one of the reasons for selecting this graphical formalism.

The main symbols we will use from the pre-conceptual schemas are depicted in Figure 1, and they are explained as follows: (i) concepts are nouns and noun phrases of the domain; (ii) connections are one-direction links between concepts and relationships and vice versa; (iii) structural relationships are the verbs to be and to have; (iv) dotted connections are used for linking concepts and notes; (v) notes are used for including possible values of concepts.

An example of an executable pre-conceptual schema is depicted in figure 2. Be advised that the leaf concepts (the ones which receive a "has" relationship and do not start another relationship) have values associated to them. Another feature of the schema is the addition of tables for including examples of certain concepts.

3 A PRE-CONCEPTUAL-SCHEMA-BASED GENERAL THEORY ABOUT SOFTWARE ENGINEERING

Initially, pre-conceptual schemas are selected for representing general theories. In Figure 3 we represent the first approach to the general theory about software engineering, based on the information included in the SEMAT kernel, which can be consulted in the Preface of this book. Particularly, we are using the alphas and their relationships and the card of the alpha *opportunity*. The main ideas about general theories are taken from Sjøberg *et al.* (2008), and they can be summarized as follows (see the blue elements in Figure 5):

Theories have one of some types (ANALYSIS; EXPLANATION; PREDICTION; EXPLANATION AND PREDICTION; DESIGN AND ACTION), propositions, explanations, and scope. Propositions are relationships among constructs. Constructs have an archetype class (ACTOR; TECHNOLOGY; ACTIVITY; SOFTWARE SYSTEM), a name, and a value.

Some other information (see the pink elements in Figure 5) is gathered from Ekstedt (2013):

Explanation has property and dependency.

The final information (see the green element in Figure 3) is related to the SEMAT kernel (Jacobson *et al.*, 2013):

The archetype class can be generalized in the shape of the so-called concern areas (CUSTOMER, SOLUTION, ENDEAVOR).

Once we created the formalism, we can use it with the purpose of explaining the initial set of the SEMAT kernel elements. In order to achieve this goal, we can assign values to the concepts based on the information the SEMAT kernel has provided. Table 1 summarizes four propositions related to the general theory for explaining the concepts of the SEMAT kernel. Also, Figure 4 includes the values of the fourth proposition.

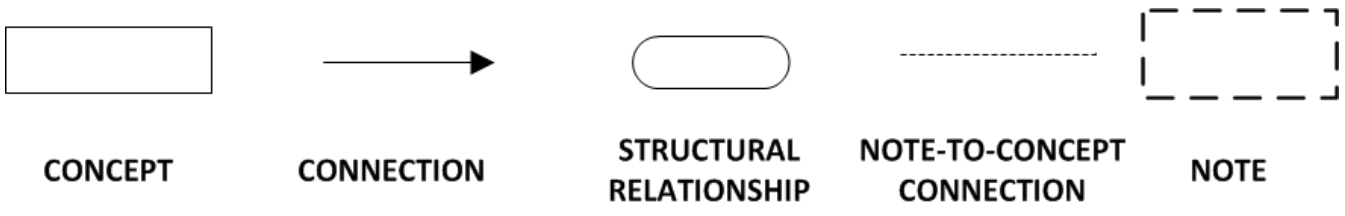
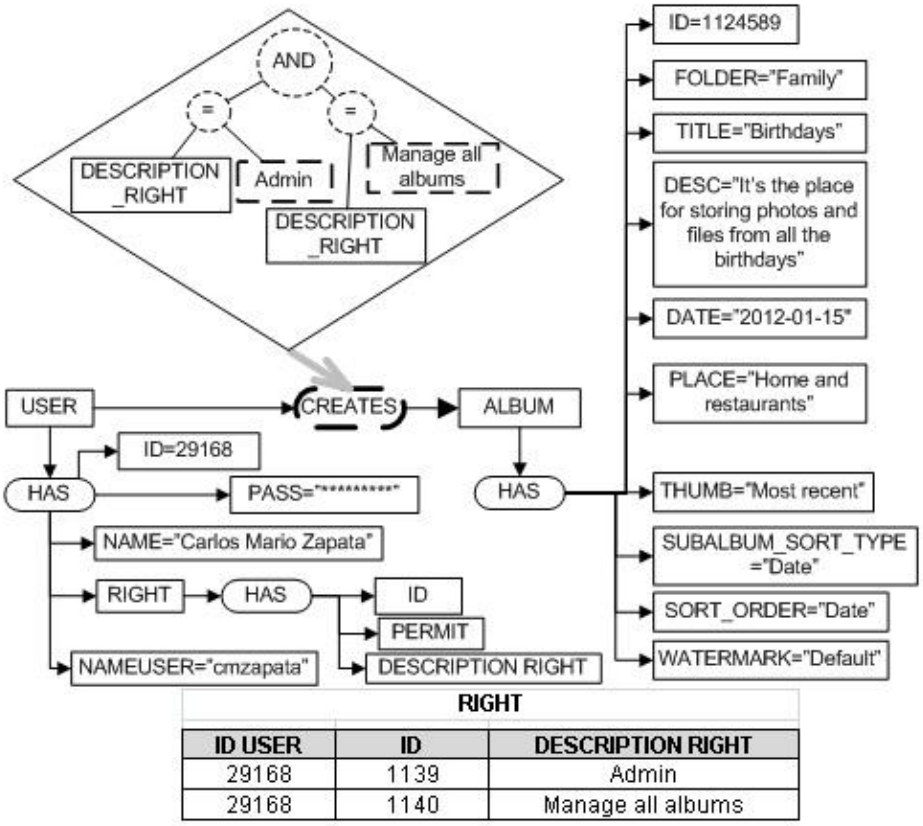


Figure 1. Main symbols of the pre-conceptual schemas. Adapted from Zapata *et al.* (2006)



RIGHT

ID USER	ID	DESCRIPTION RIGHT
29168	1139	Admin
29168	1140	Manage all albums

Figure 2. An executable pre-conceptual schema (Zapata *et al.*, 2011)

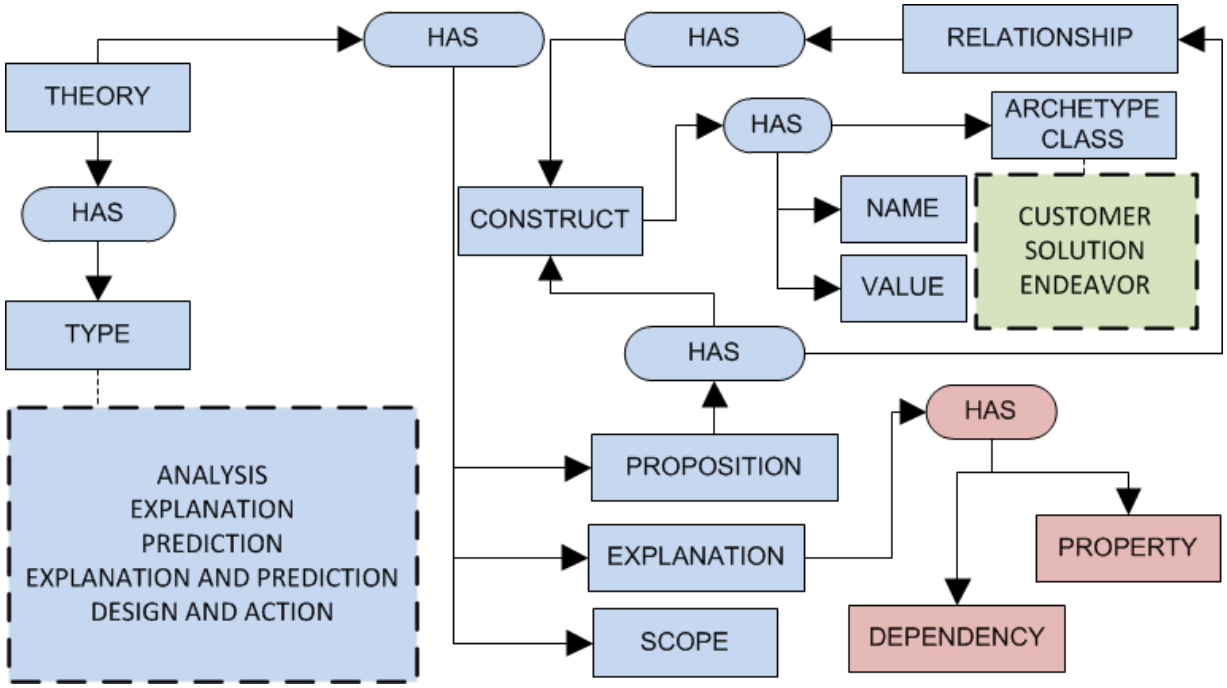


Figure 3. First pre-conceptual schema for the general theory about software engineering.

THEORY										
PROPOSITION							EXPLANATION		SCOPE	TYPE
CONSTRUCT			RELATIONSHIP	CONSTRUCT			PROPERTY	DEPENDENCY		
NAME	VALUE	ARCHE-TYPE CLASS	NAME	NAME	VALUE	ARCHE-TYPE CLASS				
Alpha	Stakehol-der	Customer	Identifies	Alpha	Oppor-tunity	Customer			Explain-ing the main concepts of the SEMAT kernel	Explanation
Alpha	Stakehol-der	Customer	Supports	Alpha	Team	Endeavor				
Alpha	Opportu-nity	Customer	Has	State	Identi-fied	Customer	A good oppor-tunity is iden-tified address-ing the need for a software-based solution	Before the state <solution need-ed>		
Alpha	Opportu-nity	Customer	Has	State	Value establis-hed	Customer	A good oppor-tunity has es-tablished val-ue	After the state <solution need-ed> and before the state <viable>		

Table 1. Values of four propositions related to the general theory about software engineering.

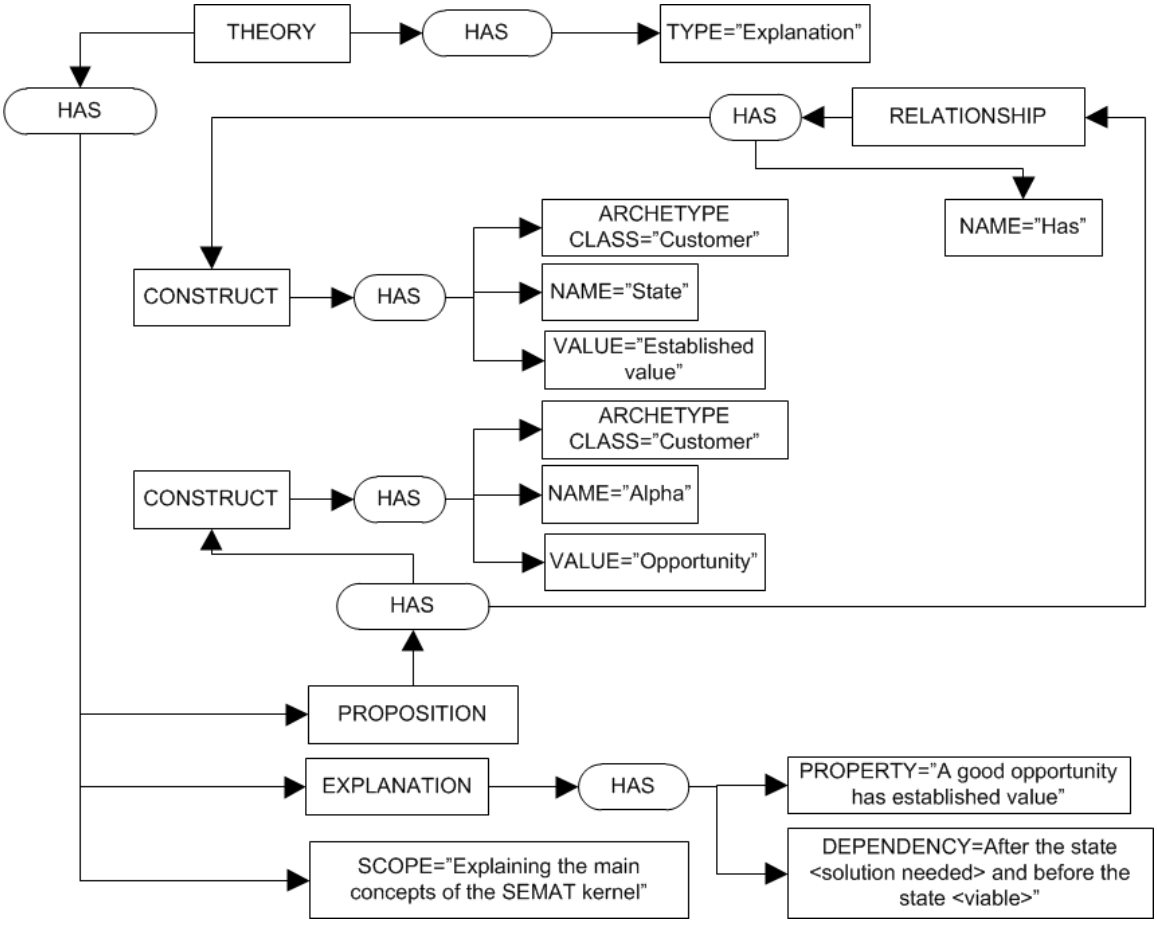


Figure 4. Executable pre-conceptual schema for the values of the proposition "Opportunity has the state <established value>"

4 DISCUSSION AND RESULTS

A general theory about the software engineering is useful for several purposes: the explanation of several assertions about the domain itself, the analysis of several propositions, the prediction of expected values, the design of situations about the domain, etc. Since we have the initial information about the SEMAT kernel, we can provide some explanation of such information with the help of a general theory. The selected formalism in this case is related to the so-called executable pre-conceptual schemas, due to the fact that they can express the information in a standardized way and they can keep controlled the usage of terminology.

As we could demonstrate by creating both the Table 1 and the Figure 4, the concepts related to the general theory are adequate for including the initial information of the SEMAT kernel, explaining several propositions and the properties behind them in a suitable way. Some other elements of the SEMAT kernel can be considered in the same way, for example activity spaces, competencies, and patterns.

5 CONCLUSIONS AND FUTURE WORK

General theories have been developed for several knowledge fields in order to explain, analyze, design, and study the phenomena related to such fields. Software engineering also have some general theories, but they are focused on specialized issues instead of general ones, and they sometimes lack the minimum formalism needed for generating a proof theory. For the aforementioned reasons, in this paper we proposed a pre-conceptual-schema-based general theory about software engineering. We used the methodology proposed by Ekstedt (2013), and the information provided by Sjøberg *et al.* (2008) and Jacobson *et al.* (2013) for completing and exemplifying the intended general theory.

The results are promising, since we could establish some propositions included in the SEMAT kernel, but we need to enhance and prove other capabilities of the general theory for generating additional tasks like analysis, prediction, and design. Probably, new elements will be needed, but the initial ones also can prove to be useful for the other purposes. As the elements of the SEMAT kernel evolves, we need to involve as many elements as we need, but keeping control of the terminology defined by our proposed general theory about software engineering.

6 REFERENCES

Adolph, S., Kruchten, Ph., & Hall, W. 2012. Reconciling perspectives: a grounded theory of how people manage the process of software development. *The journal of systems and software* 85: 1269–1286.

Bartels, R. 1968. The general theory of marketing. *Journal of marketing* 32(1): 29–33.

Coleman, G. & O'Connor, R. 2007. Using grounded theory to understand software process improvement: a study of Irish software product companies. *Information and software technology* 49: 654–667.

Ekstedt, M. 2013. An empirical approach to a general theory of software (engineering). In *proceedings of the GTSE*, San Francisco, 23–26.

Frickel, S. & Gross, N. 2005. A general theory of scientific/intellectual movements. *American Sociological Review* 70(2): 204–232.

Green, A. E. & Laws, N. 1966. A general theory of rods, *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences* 293(1433), 1966, 145–155.

Jacobson, I., Pan-Wei, N., McMahon, P., Spence, I., & Lidman, S. 2013. *The Essence of Software Engineering—Applying the SEMAT Kernel*. London: Addison-Wesley.

Johnson, P., Jacobson, I., Goedicke, M., & Kajko-Mattson, M. 2013. Second SEMAT workshop on a General Theory of Software Engineering (GTSE 2013). In *proceedings of ICSE*, San Francisco, 1525–1526.

Keynes, J.M. 1937. The general theory of employment. *The quarterly journal of economics* 51(2): 209–223.

Köhler, G. 1979. Toward a general theory of armaments. *Journal of Peace Research* 16(2): 117–135.

Medeiros, P. & Horta, G. 2013. On the representation and aggregation of evidence in software engineering: a theory and belief-based perspective. *Electronic notes in theoretical computer science* 292: 95–118.

Mitchell, R., Agle, B., & Wood, D. 1997. Toward a theory of stakeholder identification and salience: defining the principle of who and what really counts. *The academy of management review* 22(4): 853–886.

Rueff, J. 1947. The fallacies of Lord Keynes general theory. *The quarterly journal of economics* 61(3): 343–367.

Sjøberg, D., Dybå, T., Anda, B., & Hannay, J. 2008. Building theories in software engineering. In F. Shull *et al.* (eds), *Guide to advanced empirical software engineering*: 312–336. London: Springer-Verlag.

Trachtenberg, M. 1990. A general theory of software-reliability modeling. *IEEE transactions on reliability* 39(1): 92–96.

Zapata, C. M. 2012. *UNC-Method revisited: elements of the new approach*. Saarbrüken: Lambert Academic Publishing.

Zapata, C. M., Gelbukh, A., & Arango, F. 2006. Preconceptual Schema: A Conceptual-Graph-Like Knowledge Representation for Requirements Elicitation. *Lecture Notes in Computer Science* 4293: 17–27.

Zapata, C. M., Giraldo, G., & Londoño, S. Esquemas preconceptuales ejecutables. *Avances en Sistemas e Informática* 8(1): 15–23.

Zerangue, K. 1993. On developing a general theory of software engineering. In *Proceedings of the 17th annual international computer software and applications conference*, Phoenix, 334–335.

This page intentionally left blank

Chapter #2

Essence as a Framework for Conducting Empirical Studies

S. Huang
Florida Atlantic University

P.-W. Ng
Ivar Jacobson International

1 INTRODUCTION

Science and technology fields, such as physics and computing, have continuously sought underlying theories, rules, and patterns that we can use to precisely predict outcomes given certain inputs, and to explain certain phenomena of the universe. Theories help us explain how the universe works. Good theories and models should be able not only to explain what has been observed, but also predict phenomena that have yet to be observed (Tichy, 2011). Similar to other mature disciplines, *e.g.*, physics, medicine, psychology etc., software engineering as a discipline is on its way to become more mature evidenced by the many theories and models that describe our field. Some theories (models)—related to empirical software—have established their broader impact. Examples of such theories (models) include Human Information Process (HIP) model (Hedge, 2014; Miller, 2014; Miller, 1956) that explains human ability to process and respond to the information they received through their senses; Technology Acceptance Model (TAM) that models that how users are come to accept and use a technology (TAM, 2014); and the Theory of Planed Behavior (TPB) (Ajzen, 1991) that has been used successfully to predict and explain a wide range of health behaviors and intentions.

From new methods and technologies adoption perspective, many results and tools from software engineering research often failed to move from research prototype to be widely adopted and deployed in industry; instead, they remained as research orphans. Glass *et al.* (2002) noted that for those that did make to industry, it could take some 15 years on average from initial discovery to practical usage. The academic and research community had started to address this problem. Adoption Centric Software Engineering (ACSE; Tilley *et al.*, 2002; Balzer *et al.*, 2004) was an example of such effort. ACSE aimed to advance the adoption of software engineering tools and techniques by bringing together researchers and practitioners to investigate novel approaches for fostering the transition from limited-use research prototypes to broadly applicable and practical solutions.

The industry is also looking at bridging the gap between industry and research. Jacobson *et al.* (2009) noted that software engineering today is gravely hampered by immature practices. They later started the Software Engineering Method and Theory (SEMAT, 2014) initiative with the goal to bridge this gap (Jacobson *et al.*, 2012). As a first step, SEMAT working groups have established a language and a kernel that define commonly used elements in software engineering, known as Essence (OMG, 2014). Jacobson *et al.* (2012b; 2013) further demonstrated how to use Essence in software development in both traditional development contexts and modern agile development contexts with both small and large teams. Although Essence is relatively new, its earlier version has been applied to industrial settings (Jacobson *et al.*, 2012b; 2013).

To effectively foster adoption of new methods and research results, a mechanism is needed for providing a common basis of comparison and evaluation, and give objective assessment of effectiveness and efficacy of the new approach. In this Chapter we propose the use of Essence—the newly recommended OMG standard—as such a desired foundation, and we demonstrate how to use Essence to systematically assess an existing empirical study, report its findings, and identify the validation threats to the selected case study.

This Chapter is organized as follows[1]: in Section 2 we review some existing empirical software engineering research and identify one of the emerging common challenges of empirical software engineering—the need for a comparison framework. In Section 3 we further discuss the benefit of providing common comparison framework for empirical research, and we layout the quality attributes belonging to Essence as a good candidate for such a framework. In Section 4 we show how to augment Essence with properties of interest

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

that make it suitable to serve a good evaluation framework. In Section 5 we describe in detail how to use Essence to provide analytical guideline for conducting case studies. In Section 6 we illustrate an example of using Essence as an evaluation framework for conducting case studies. Finally, in Section 7 we conclude this research and we point out future work.

2 EMPIRICAL SOFTWARE ENGINEERING RESEARCH AND THE NEED FOR A COMMON COMPARISON FRAMEWORK

Empirical software research has been growing recognition (Dybå *et al.*, 2005; Budgen *et al.*, 2002; Tichy, 2011) due to its critical roles in assessing and comparing different software development methods and provides evidence-based software engineering (Kitchenham, 2004). As we can see from the existing research below, our community is in dire need of a common comparison framework that can be used to assess and evaluate different software engineering methods, which is exactly what Essence provides.

Dybå and Dingsøyr (2008) surveyed research for empirical evidence of agile software development and found that different reporting content hinders analysis. They researched 36 empirical studies out 1996 studies related to agile development. They grouped their studies into four themes: introduction and adoption, human and social factors, perceptions on agile methods, and comparative studies. Their goal is to investigate what is currently known about the benefits and limitations of, and the strength of evidence for, agile methods. Their findings pointed out that the main implication for research is a need for more and better empirical studies of agile software development within a common research agenda.

Jedlitschka *et al.* (2008) also pointed out that a major problem for integrating software engineering research results into a common body of knowledge is the diversity of reporting styles. It is difficult to locate relevant information; and important information is often missing. They suggested a schema for describing (dependent, independent, or moderating) variables.

Petersen and Wohlin (2009) found that studies investigating a similar object do not agree on which context facets are important to mention and provided a checklist that aims to help researchers make informed decisions on what to include and not to include. In their survey, they used context facets such as market, organization, product, processes, practices (including tools and techniques), and people. They also noted that there is still work needed to reach a consensus within the software engineering research community.

Feldt and Magazinius (2010) reviewed the papers in ESEM 2009 to compare the numbers of validity threats versus mitigation strategies and found that an alarming number of these do not address validity threats and a large number mitigate the threats as future work. They also pointed out that these papers "had no unified framework with which to classify the threats or strategies, so our analysis at this stage is only indicative."

Murphy-Hill and Williams (2012) also discussed the capability to generalize the research findings. The question is whether it is valid to extrapolate the environment in which the software engineering research took place to somewhere closer to the reader environment. They identified several similarity types: people similarity, tool similarity, activity similarity, artifact similarity, and temporal similarity. They highlighted that the behavior between software engineers and knowledge workers in other domains are similar. Such similarities would expand the scope of validity of research findings. Nevertheless, they recognized that the study of generality is still at its infancy in software engineering research.

Runeson and Höst (2009) present a nice collection of case study methodology and provide guidelines for researchers conducting case studies and readers studying reports of such studies. The reports are the main source of information for judging the quality of the study. Guidelines (*e.g.*, in detailed tabular format and checklist) of reporting case study findings have been proposed by Jedlitschka and Pfahl (2005) and evaluated by Kitchenham *et al.* (2008) with the goal to define a standardized reporting of experiments that enables cross-study comparisons through systematic review. As Runeson and Höst (2009) pointed out, the high-level structures are mostly based on qualitative data, and the low-level detail is less standardized and more depending on the individual case.

In an interview with Sjøberg conducted by Tichy (2011), when asked about whether UML (Unified Modeling Language) is helpful (in software development), Sjøberg indicated that "the lack of a framework for formulating specific research questions leads to a lack of sharing of independent, dependent, and context variables among studies, as well as questionable quality of many of the studies." Sjøberg reaffirmed our belief about the need for a common comparison framework.

From the brief survey above, it is clear that reporting empirical study findings lacks a systematic framework. The objective of this Chapter is to show that Essence can be a foundation for such a framework.

3 CHARACTERISTICS OF ESSENCE AS AN EMPIRICAL RESEARCH FRAMEWORK

Surely, too much effort is still needed in order to bring research and industry together. Huang and Tilley (2004) noted that fostering adoption of research findings required more empirical studies and reported the challenges in doing so. Our brief review showed that there is indeed a growing interest in

empirical software engineering research. However, we also found that a systematic framework for reporting case studies and findings is lacking. As a consequence, it is difficult to search for relevant case studies, compare findings, and generalize results.

Before proceeding further, we would like to clarify what a framework means in the context of this Chapter. We recognize that there have been generally accepted approaches on conducting research and reporting case studies, such as those by Shaw (2002) and Runeson and Höst (2009). These focus more on research methods. On the other hand we want to focus on research data, which we call properties of interest. Examples of properties of interest include team size, team distribution, system complexity, and stakeholder diversity.

The framework we are looking for should provide guidelines as to what properties should be collected and reported as well as how to organize them for easy understanding. Specifically, properties of interest should have a good coverage of software engineering endeavors, be organized hierarchically, and at each level of the hierarchy be orthogonal or non-overlapping as far as possible. In addition, this framework should also provide analytical guidelines to evaluate properties for their realism and the generality of the findings.

3.1 *Essence as a foundation for an empirical research framework*

We acknowledge that Essence is still at its infancy. Furthermore, we also recognize that the primary audience of Essence is practitioners, as opposed to researchers. Thus, additional work is needed to make Essence also useful and usable for researchers. Specifically, Essence explicitly does not identify or define a common set of observable, controllable properties of interest which researchers can select and use in empirical research.

Nevertheless, we still believe that Essence is an attractive candidate as a foundation for an empirical research framework because of several reasons:

a) Comprehensiveness—Essence identifies the dimensions of challenges a typical software development endeavor faces. These dimensions are intuitively orthogonal and provide a good basis for spanning and organizing properties of interest.
b) Model based—Essence expresses the relationships between these different dimensions, and hence provides a foundation to model the relationships between various properties of interest.
c) Extensible—Essence has extension mechanisms to cover more challenges by adding practices as needed. This overcomes the trap of having too much unnecessary information too early. It gives researchers and practitioners the ability to zoom in to the details as appropriate. This additional information, *i.e.* properties of interest, is added by what Essence calls *practices*.
d) Configurable—Essence provides several mechanisms to describe the diverse range of software engineering approaches. As an example, Figure 1 describes the differences between modern software development and traditional and more conservative development, which has different risk emphasis.

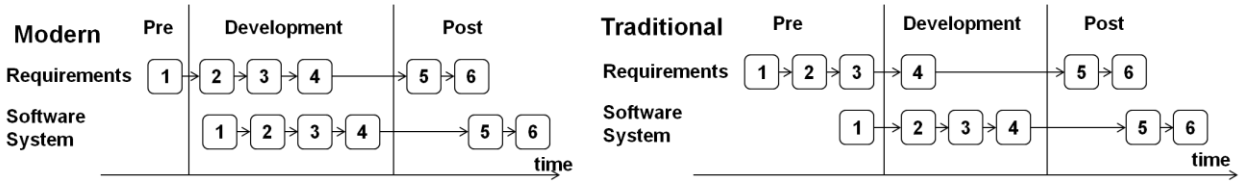


Figure 1. Difference between Traditional and Modern Development Approaches (Ng & Huang, 2013)

e) Tangible – Each alpha states and their checklist are presented as a deck of cards (business card size) as shown in the Preface of this book. We usually get project teams to discuss what they need to do by laying out the cards on the table and moving them around. The state definitions can be used to design task boards and Kanbans (Kniberg, 2010) that are common in agile methods and tooling today.

The above characteristics of Essence make it a good candidate as a foundation for a systematic framework to report case studies. We achieved this goal after augmenting Essence with different kinds of properties for describing case study research and analytical guidelines for evaluating research data and findings.

3.2 *Evaluating the Framework*

To evaluate our framework, we took an existing case study and compared it with how we would have described and analyzed results, albeit using this framework. The existing case study we chose was conducted by Koskela and Abrahamsson (2004). This case study investigated if the role of a customer representative was too demanding in an extreme programming (XP) environment. We took this case study and evaluat-

ed how its properties of interest could be better organized. We analyzed if it had included relevant properties of interest.

To our surprise, by using our framework, we detected a strong threat to the validity of this case study. Understandably, this validity threat could not be detected by any watchful eye without using any specific framework. Nevertheless, the fact that such a validity threat had not been pointed out by the original paper or subsequent secondary studies only serve to highlight the importance of having systematic reporting and concrete evaluation guidelines. This is what a framework is supposed to do.

More importantly, through this case study we show that the value Essence can bring to software engineering research, which we believe is a small but important step towards building a body of knowledge surrounding Essence and SEMAT. Ultimately, we would like to encourage the software engineering research community to use Essence in their research. The experience and feedback gained from the research community will not only serve to validate Essence empirically, but also provides inputs to its further improvement.

4 AUGMENTING ESSENCE WITH PROPERTIES OF INTEREST

As mentioned before, the primary audience of Essence is practitioners. The alphas represent different dimensions of risk and challenges, and achieving progress is about moving from one state to the next along each alpha.

The Essence specification (OMG, 2014) does not, at the time of writing of this Chapter, explicitly identify a common list of properties for describing alphas. While properties such as size and complexity of a Software System, size and distribution of a Team, community size and diversity of a Stakeholder are mentioned in the Essence specification, there is no explicit attempt to identify a common set. However, such properties are important for software engineering research because they can be either independent or dependent variables in software engineering research.

However, Essence does provide extensibility mechanisms to overlay properties on top of the kernel. How to use these mechanisms to introduce properties is out of the scope of this Chapter.

Instead, we want to identify the kinds of properties that are of interest to readers of the software engineering research results. Structural properties are those about structures and relationships in a software engineering endeavor. Flow properties are those about information and process flows in a software engineering endeavor. We will give some examples but leave the outlining a comprehensive list as future work.

4.1 *Structural Properties*

Structural properties are those that describe and characterize the structure and complexity of entities. They are useful for characterizing software engineering research contexts. A good number of commonly used structural properties can be expressed as a function of alpha instances and their relationships, or pegged to them. For example, size of the team is the number of people in a team instance.

Briand *et al.* (1996) provide a general mathematical framework for several important measurement concepts (size, length, complexity, cohesion, and coupling) for Software System artifacts, which we believe can be extended to other alpha types by using Essence.

Characterizations of relationships are also important. For example, Bird *et al.* (2009; 2011) by making empirical findings showed that developer-module relationships have strong influence on quality predictions. Minor contributors of software modules have a higher likelihood to introduce defects. Such relationships can be expressed in Essence language as relationships between Team (member) instances and Software System (component) instances, where the objects in the parenthesis are introduced by Team and Software System practices, respectively. These relationships can be expressed succinctly through some team organization patterns. Similarly, the relationships between Stakeholder instances and Requirements, as well as stakeholder expectations and competencies have strong impact on the success of a software development.

Alphas provide a way to organize properties of interest in a model. Figure 2 shows a partial model comprising Team and Software System, augmented with some structural properties.

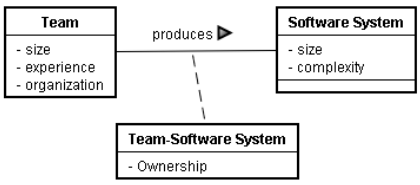


Figure 2. Augmenting Alphas with Structural Properties

4.2 *Flow Properties*

Flow properties are those that measure progress flows. For example, different instances of the Requirements alpha can flow through the Requirement alpha states at different places. Some instances move slower, while others move faster depending on how complex they are and how quickly the team works. We can augment properties such as the time spent on a particular alpha state, the waiting time, and value added effort versus non-value added effort to each state (see Figure 3).

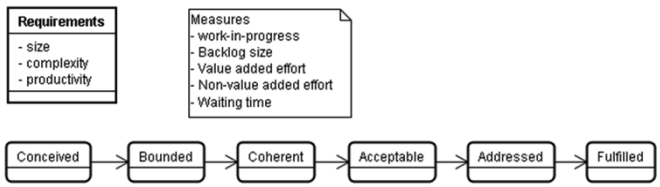


Figure 3. Augmenting flow Measures to Alpha States

Such properties are used when looking for productivity gains and are heavily used in lean process improvements (Mujtaba *et al.*, 2010; Petersen & Wohlin, 2010). We want to point out that the productivity properties have to be anchored to the items being worked on. In the case of Figure 3, they are about Requirements, as opposed to architecture work, which should be pegged to the Software System alpha.

5 PROVIDING ANALYTICAL GUIDELINES

As highlighted earlier, the software engineering research community lacks a common framework to report studies (Petersen & Wohlin, 2010; Jedlitschka *et al.*, 2008; Dybå and Dingsøyr, 2008; Feldt & Magazinius, 2010; Murphy-Hill & Williams, 2012). Essence provides a way to address this problem by organizing properties of interest in a model of alphas and their relationships as defined by Essence. Now we take a step further to provide guidelines for analyzing case studies. In particular, we identify the need for two kinds of measures:

- Comprehensiveness measures are about whether the study has adequate and relevant information to help the reader understand it.
- Distance measures allow for comparisons as to how far a study is to reality or to a reader situation.

These measures help the researcher evaluate if there is sufficient data being reported in the study, and to consider the realism and generality of his study.

To be more precise in what we mean by comprehensiveness and distance, we will first use a simple mathematical set-theoretic formulation. We will then use this formulation in the next Section when we demonstrate an example of using Essence to conduct research.

5.1 *Comprehensiveness Measures*

A study describes a software engineering endeavor or a set of endeavors whose setup and execution are observed. We first define the description P_E of a software engineering endeavor E as a list of properties:

$$P_E = \{p_{E1}, p_{E2}, p_{E3}, p_{E4}, \ldots\}$$

Each p_{Ei} is a property description in the form of:

alpha-property:value

For example, consider the development endeavor E_{Web} of a simple web-based system performed by three students working together. Its description is a list of properties:

$$P_{E_{Web}} = \{\text{Team-size: 3, Team-experience: student, Team-distribution: co-located, Software System-technology: web }\}$$

This is a very brief description and far from complete. Moreover, the property Team-experience is vague as it does not indicate what the experience is about, *i.e.*, about the Requirements, Software System, or Way of Working. It only indicates that the experience is "students". Thus, $P_{E_{Web}}$ is not comprehensive. Although it has some information about Team and Software System, it does not have information about other Essence alphas, such as Opportunity, Stakeholders, Requirements, Work, and Way of Working. Thus, when talking about comprehensiveness, it is important to also talk about the dimensions of comprehensiveness. These dimensions should be as orthogonal as possible because we do not want to have duplicate information. Incidentally, the Essence alphas are chosen to be as orthogonal as possible, albeit only intuitively.

We can identify a comprehensiveness of a property description P as $C_{Dimension}(P)$ where the dimension can be a particular alpha, or all the alphas in the Essence. Thus, $C_{Stakeholders}(P_{E_{Web}})$ is zero since there is no information about stakeholders in $P_{E_{Web}}$, *i.e.*, $P_{E_{Web}}$ is an empty set. $C_{Essence}(P_{E_{Web}})$ is the comprehensiveness of the description $P_{E_{Web}}$ over all the different dimensions spanned by the alphas and relationships defined in the Essence kernel. We will use $C(P_{E_{Web}})$ as a short form for $C_{Essence}(P_{E_{Web}})$.

In this case, we have four pieces of information (*i.e.*, four descriptions of four properties), which is inadequate to give a thorough description and we say that the comprehensiveness of $P_{E_{Web}}$ is *weak*. However, there is more information in the Team dimension, and we can say that the comprehensiveness about the team dimension is *strong*. Thus, we encounter the problem of scales, *i.e.*, what *weak* is, what *strong* is, how many property description constitute *weak* or *strong*. Note that comprehensiveness is a function of the proper number of properties in the list. The more items in a description list, the stronger the confidence, provided that the properties are orthogonal. A detailed discussion of the scales for a comprehensiveness measure is a complex one and we will consider the discussion to be outside the scope of this Chapter. For the purpose of this Chapter, we will use terms like *weak* and *strong* intuitively. However, by now we have already

achieved an important step, which is to distinguish different dimensions of comprehensiveness according to alphas.

5.2 *Distance Measures*

The reader of a case study is interested whether the case study applies to his/her situation, or whether a practice is suitable for a particular software endeavor. The measure of applicability and suitability can be identified to be a distance measure.

Before proceeding further, we want to introduce a shorthand E to represent the *complete* description of a software engineering endeavor. Strictly speaking, an endeavor and its description are never the same things, and we can never describe an endeavor completely. We can compare the characteristics of two software engineering endeavors E_1 and E_2 by using some distance measure as $D(E_1, E_2)$, which is the *actual* distance as opposed to the *measured* distance $D(P_{E_1}, P_{E_1})$, because we can measure only what we are given.

Similarly, as in the previous subsection, we can quantify the dimension of the distance. For example, $D_{Stakeholders}(E_1, E_2)$ is the distance (how different) between two engineering endeavor E_1 and E_2 over the Stakeholders dimension.

The same notation can also be used for describing a practice. For example, a practice A is suitable for co-located teams building a small web application.

P_A = {Team-size: small, Team-distribution: co-located,
Software System-technology: web}

It is now possible to describe the applicability this practice A for a software engineering endeavor, such as the web development endeavor E_{Web} by using the distance measure:

$$D(P_{E_{Web}}, P_A)$$

If we can describe both completely, then the distance becomes $D(E_{Web}, A)$.

When considering distances, one need also consider scales. Note that since distance is a function of the differences in each property description, scales have to be individually defined for each property. For example, Runeson (2003) found that there are small differences between graduate students and industry people on one hand, while there are significant differences between graduate students and freshmen on the other hand. We will consider the discussion of scales to be outside the scope of this Chapter. In this Chapter, we will use intuitive terms like *weak* or *strong* for comprehensiveness and *near* or *far* for distance.

6 AN EXAMPLE OF USING ESSENCE TO CONDUCT RESEARCH

In Section 2, we have pointed out that a systematic framework for comparing and reporting research findings is lacking and we advocated Essence as a foundation, albeit after augmenting with property descriptions and analytical guidelines.

In this Section, we demonstrate how to use this augmented Essence as a framework for conducting research. As an example, we compare how an existing software engineering research reported its findings versus how our framework would report the same findings. Our candidate software engineering research case study is conducted by Koskela and Abrahamsson (2004). This case study was not arbitrarily chosen. Dybå and Dingsøyr (2008) searched for empirical studies up to 2005, inclusive. They identified 1996 studies from the literature, of which 33 were found to be research studies of acceptable rigor, credibility, relevance, and were primary studies. Koskela and Abrahamsson (2004) was one of these 33 studies. Using the guidelines from Runeson and Höst (2009), we organize our analysis in the following subsections:

A. Formulating the Research
B. Describing the Research Context
C. Describing the Research Execution
D. Analyzing and Concluding the Research

In each subsection, we have different paragraphs representing the original research paper and our suggested approach, respectively:

- **Study Description**—this contains text that largely comes from the original study itself. Our changes are primarily editorial.
- **Essence Description**—we use the model-based approach discussed in the Preface of this book to describe the case study.
- **Essence Analysis**—we use the guidelines in Section 5 to analyze the comprehensiveness of the case study and its validity and highlight possible issues concerning the study.

6.1 *Formulating the Research*

Study Description—the goal of the study was to assess whether the role of the customer representative is too demanding in an extreme programming (XP) environment. This study was conducted in a university setting with students and staff.

Essence Analysis—There are several entities to consider for such a study, namely:

- The practice *P*, which is extreme programming here.

- The experimental environment, which is the software engineering endeavor *E*, conducted in a university setting.
- The real world environment *R*, which readers of the study are interested in.
- The hypothesis *H*, which is whether XP places too much demand on the customer representative.

To have realistic and conclusive results from the study, both the distance $D(P,E)$ and the distance $D(R,E)$ should be near. The first distance ensures that the experiment applies XP faithfully, and the second distance ensures that results are useful for the real world.

In addition, this study hypothesis is about customer representatives and teamwork. Thus, we should place emphasis on the comprehensiveness over both Stakeholders and Team dimensions, *i.e.*, $C_{Stakeholders}$ and C_{Team}.

Essence Description—to formulate this study, we would summarize the above analysis, and highlight the strategies to keep the distance measures near.

6.2 *Describing Research Context*

Study Description—the research setting was a team of 4 developers (5th or 6th year students) who had one to four years of experience. The team members were well versed in Java programming language and object-oriented analysis and design approaches. They were beginners to XP with just a two-day training. They used Eclipse/JUnit/CVS. The application was written in Java and JSP (JavaServer Pages) and it used a MySQL relational database to store link data, in addition to an Apache Tomcat 4 Servlet/JSP container. The team worked in a co-located development environment. The customer, *i.e.*, the first author of the case study, shared the same office space with the development team. The office space and workstations were organized according to the suggestions made in XP literature (Jeffries *et al.*, 2001; Beck & Andres, 2004) to support efficient teamwork. Qualitative data included development diaries maintained by the developers, a customer diary, post-mortem analysis session recordings and developer interviews. The developers and the customer were updating their diaries continuously during the project, tracking time and filling in observations.

Essence Description—the study's context can be modeled concisely using Essence through structural measures associated with alphas as depicted in Figure 5. It is a visual representation of properties of a software engineering endeavor. Such a visual approach facilitates discussions and helps identify issues in the experimental context and description. At the very least, it gives a visual feel regarding which dimensions are more comprehensively described over those that are weakly described.

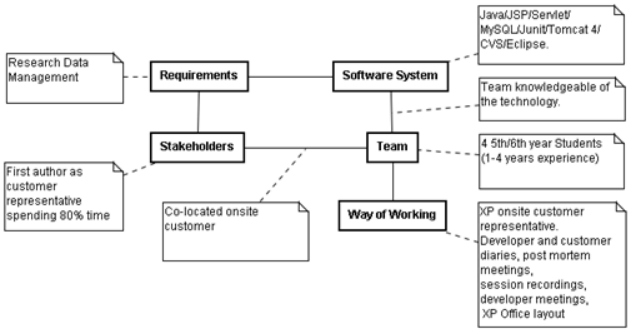


Figure 4. Using Essence structural properties to describe study context

Essence Analysis—from the study description and the visual representation in Figure 4, it is clear that comprehensive information is available for both Team and Way of Working dimensions, *i.e.*, both $C_{Team}(P_E)$ and $C_{WayOfWorking}(P_E)$ are strong.

It is also clear from Figure 4 that the experiment design has little detail on the *Opportunity and Requirements*, *i.e.*, both $C_{Opportunity}(P_E)$ and $C_{Requirements}(P_E)$ are weak.

More importantly, the *Stakeholders* environment, involving only one person, is simplistic. As a software engineering coach, the first author of this paper often encounters environment where the customer representative has to interface with many other stakeholders. This study did not mention the additional work the customer representative was responsible for. In short, $C_{Stakeholders}(P_E)$ is moderate and $D_{Stakeholders}(R,E)$ is far.

6.3 *Describing Research Execution*

Study Description—system development was carried out in six iterations, of which the first three took two weeks of calendar time each, next two took one week each, while the sixth iteration was a two-day correction release. The collected data for each iteration included total work effort, number of user stories implemented, and tasks defined. They also tracked the time duration when the customer representative was present.

Essence Description—the properties being observed can be modeled visually and concisely using Essence as depicted in Figure 5.

Essence Analysis—since the goal of this study was about the workload on the customer representative, it is important to explore factors affecting the workload. Essence expresses a relationship from *Stakeholders* to *Requirements*, so it is natural to ask about the requirements churn and complexity, which would have an impact on the customer representative workload. The study does not provide such details. This suggests that $C_{Requirements}(P_E)$ is weak. Since requirements

change is a typical challenge in the real world environment, we might add that $D_{Requirements}(R, E)$ is far.

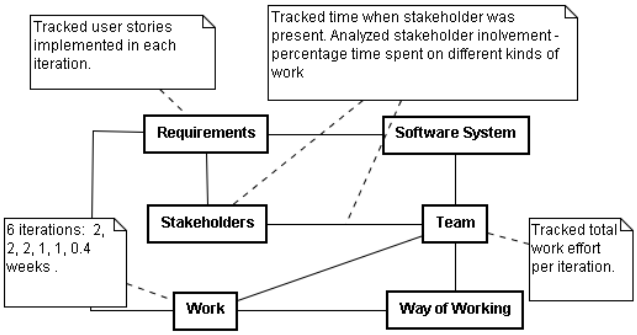


Figure 5. Using Essence to describe flow properties of observed behaviors

6.4 *Describing Research Execution*

Study Description—Koskela and Abrahamsson (2004) noted that many authors claim that on-site customer involvement is often difficult to realize or even unrealistic due to the required customer work effort. A contrasting result offered by this study was that while the customer was present with the team at an average of 83% of his work effort, only 21% was required for assisting the development team in the actual development work.

This study also reported that the developed solution had not been used since as actively as intended. The reason for this can be attributed to the relatively poor usability of the system. Yet, all the related stakeholders were happy with the solution when it was under development. The on-site customer also had a lot to say on how the system should function. Consequently, while the experiences were mostly positive, the data also revealed that the on-site customer practice was in danger of creating a false sense of confidence towards the system under development. The study suggested that the customer needed for example to invest in user-centered design (UCD) to address this issue.

Essence Analysis—From our analysis, it is difficult to support the claims made in this study because:

- In the experimental setup, the stakeholder environment is simplistic, *i.e.*, $D_{Stakeholders}(R, E)$ is far.
- In the experimental execution, there is little information on the complexity of the requirements and requirements churn, which may have impact on the customer's involvement, *i.e.*, $D_{Requirements}(R, E)$ might be far. In the real world, the customer representatives would also be spending much time, *e.g.*, collecting user inputs and mediating requirements between different stakeholders. Apparently, this was not something that happened in the study. In short, $D_{Stakehdolers}(R, E)$ is also far.

Thus, our conclusion is that the original case study had a strong threat to validity.

Essence Description—based on the above analysis, we have two choices to report the study findings. The first is to report the same observed data, but to recognize the threat to validity. For example, in addition to stating that the customer representative spent 21% of his time, it could be added that the figure would be higher after factoring the additional time that would have been needed for learning usability design, and collecting user opinions that is potentially different.

The second choice is to investigate what would be more realistic stakeholder conditions and, if feasible, include that in the experiment design. This would mean repeating the entire study itself. However, this would be more realistic.

Conclusion—in this particular example, we found that we would have a conclusion different from that of the original case study. Now, we did not begin with the intention to refute any existing studies. The detection of this deficiency is only coincidental. Nevertheless, this coincidence is an indication of the importance of having a systematic framework, and in particular the value of Essence as a foundation for such a framework.

7 CONCLUSIONS AND FUTURE WORK

The goal of SEMAT is to bridge the gap between software engineering research and industry. As we analyze existing software engineering research literature to ascertain the value which Essence can bring to software engineering research, we find that there is growing recognition in empirical research. Unfortunately, we also begin to realize that there has no widely accepted framework for reporting empirical results, nor a systematic way to evaluate these results. Consequently, it is difficult to compare the findings from different studies; and it is also difficult to aggregate the results.

In this Chapter, we demonstrated the use of Essence as a foundation for such systematic framework. This framework provides a simple means to model and visualize the properties of interest that are captured in a case study. The framework also has a set of measures for evaluating case studies. Comprehensiveness measures help evaluate a case study that includes relevant information and identifies the missing information. Distance measures help evaluate the realism of the case study and the applicability of a practice to different contexts.

We took an existing case study and used Essence to describe it. Essence was not only able to provide a model-based and visual representation, but also in this particular case, was also able to identify a strong threat to validity. It is important to note that being able to detect such validity threats could be

achieved with a watchful eye without a framework. However, the role of software engineering research is to provide practitioners with mechanisms to detect risks and address them as early as possible in their development lifecycle, rather than leaving to chance. In this particular case, a tool to systematically detect the presence of validity threats was missing in the original case study. In this paper, we have shown that how our proposed framework could be such a tool to detect threats early.

Our experience with the case study highlights the importance of having a systematic framework to conduct and report empirical software engineering research. It serves as evidence to highlight the value of Essence to software engineering research. This experience had been very encouraging to us. Nevertheless, we recognize that this is still preliminary research and much work lies ahead. We only evaluated our framework and approach against a single case study. We did not discuss how Essence would be useful for systematic reviews, *i.e.*, when attempting to summarize multiple and diverse studies.

For future work, our first and foremost task is to use Essence to describe case studies beyond the one we investigated. This approach will yield several benefits. First, it builds experience and provides feedback for improvement. Secondly, it may help to identify a set of common properties of interest in software engineering endeavors for case study reporting.

It is also important to research further on comprehensiveness measures and distance measures. The results would increase our understanding of the relationships among properties and make the proposed framework stronger. More importantly, the results would make comparisons between studies and practices more accurate.

Note:

This Chapter is an extension to the paper "On the Value of Essence to Software Engineering Research: A Preliminary Study" published in *Proceedings of 2nd SEMAT Workshop on a General Theory of Software Engineering* (GTSE 2013), co-located with ICSE 2013. May 26, 2013, San Francisco, CA, USA. pp. 51–58.

8 REFERENCES

Ajzen, I. 1991. The theory of planned behavior. *Organizational Behavior and Human Decision Processes* 50: 179-211.

Balzer, B., Litoiu, M., Müller, H., Smith, D., Storey, M., Tilley, S., & Wong, K. 2004. 4th International Workshop on Adoption-Centric Software Engineering. In *Proceedings of 26th International Conference Software Engineering (ICSE 2004)*, 748–749.

Beck, K. & Andres, C. 2004. *Extreme Programming Explained: Embrace Change*, Addison Wesley Professional.

Bird, Ch., Nagappan, N., Gall, H., Murphy, B., & Devanbu, P. 2009. Putting it all together: Using socio-technical networks to predict failures. *In Proceedings of the 20th International Symposium on Software Reliability Engineering, 2009 (ISSRE'09)*, 109-119.

Bird, Ch., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. 2011. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th Symposium on the Foundations of Software Engineering and the 13rd European Software Engineering Conference*, 4-14.

Briand, L., Emam, K., & Morasca, S. 1996. On the application of measurement theory in software engineering. Empirical Software Engineering 1(1): 61-88.

Briand, L. Morasca, S., & Basili, V. 1996. Property-based software engineering measurement. IEEE Transactions on Software Engineering 22(1): 68-86.

Budgen, D., Hoffnagle, G., Müller, M., Robert, F., Sellami, A., & Tilley, S. 2002. Empirical software engineering: a roadmap report from a workshop held at STEP 2002, Montreal, October 2002. In *Proceedings of the 10th International Workshop on Software Technology and Engineering Practice, 2002. STEP 2002*, 180-184.

Dybå, T. & Dingsøyr, T. 2008. Empirical studies of agile software development: A systematic review. *Information and software technology* 50(9): 833-859.

Dybå, T., Kitchenham, B., & Jørgensen, M. 2005. Evidence-based software engineering for practitioners. *IEEE Software* 22(1): 58-65.

Feldt, R. & Magazinius, A. 2010. Validity threats in empirical software engineering research—An initial survey. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering, SEKE*.

Glass, R., Vessey, I., & Ramesh, V. 2002. Research in Software Engineering: an analysis of the literature. *Information and Software technology* 44(8): 491-506.

Hedge, A. 2013. *Human Information Processing*, Available: http://ergo.human.cornell.edu/studentdownloads/dea3250pdfs/hip.pdf

Huang, S. & Tilley, S. 2004. On the Challenges in Fostering Adoption via Empirical Studies. In *Proceedings of the 4th IEEE International Workshop on Adoption-Centric Software Engineering (ACSE 2004: May 25, 2004, Edinburgh, Scotland, UK)*..

Jacobson, I., Huang, S., Kajko-Mattsson, M., McMahon, P., & Seymour, E. 2012. Semat—Three Year Vision. *Programming and computer software* 38(1): 1-12.

Jacobson, I., Meyer, B., & Soley, R. 2009. The SEMAT Initiative: A Call for Action. *Dr Dobb's Journal*, December 09.

Jacobson, I., Ng, P.-W., McMahon, P., Spence, I. & Lidman, S. 2012b. The Essence of Software Engineering: the SEMAT kernel. *Communications of the ACM* (10): 42-49.

Jacobson, I., Ng, P.-W., McMahon, P., Spence, I., & Lidman, S. 2013. *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley.

Jedlitschka, A., Ciolkowski, M., & Pfahl, D. 2008. Reporting experiments in software engineering. *Guide to advanced empirical software engineering*: 201-228.

Jeffries, R., Anderson, A., & Hendrickson, Ch. 2001. *Extreme Programming Installed*, Addison-Wesley.

Kitchenham, B., Al-Khilidar, H., Ali Babar, M., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., & Zhu, L. 2008. Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering* 13(1): 97–121.

Kitchenham, B., Dybå, T., & Jorgensen, M. 2004. Evidence-based software engineering" In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, 273–281.

Kniberg, H. 2010. *Kanban and Scrum-making the most of both*. Lulu.com.

Koskela, J. & Abrahamsson, P. 2004. On-site customer in an XP project: empirical results from a case study. *Lecture Notes in Computer Science* 3281: 1–11.

Miller, G. 1956. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review* 63: 81-97.

Miller, G. 2014. *Information Processing Theory*. Available http://www.instructionaldesign.org/theories/information-processing.html

Mujtaba, S., Feldt, R., & Petersen, K. 2010. Waste and lead time reduction in a software product customization process with value stream maps. In *Proceedings of the 21st Australian Software Engineering Conference (ASWEC)*, 139-148.

Murphy-Hill, E. & Williams, L. 2012. How can research about software developers generalize? In *Proceedings of the 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 105-109.

Ng, P.-W. & Shihong Huang. 2013. Essence: A Framework to Help Bridge the Gap between Software Engineering Education and Industry Needs. In *Proceedings of the IEEE 26th Conference on Software Engineering Education and Training (CSEE&T), Co-located with ICSE 2013*, 304–308.

Ng, P.-W., Huang, S., & Wu, Y. 2013. On the Value of Essence to Software Engineering Research: A Preliminary Study. In *Proceedings of 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE 2013), co-located with ICSE 2013*, 51–58.

OMG. 2014. *Essence Submission*. Available http://www.omg.org/spec/Essence/

Petersen, K. & Wohlin, C. 2009. Context in industrial software engineering research. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 401-404.

Petersen, K. & Wohlin, C. 2010. Software process improvement through the Lean Measurement (SPI-LEAM) method. *Journal of Systems and Software* 83(7): 1275-1287.

Runeson, P. 2003. Using students as experiment subjects–an analysis on graduate and freshmen student data. In *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering, Keele University, UK*, 95-102..

Runeson, P. & Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14(2): 131-164.

SEMAT. 2014. *Software Engineering Method and Theory*. Available: http://www.semat.org

Shaw, M. 2002. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer (STTT)* 4(1): 1-7.

TAM. 2014. Technology Acceptance Model. Available http://en.wikipedia.org/wiki/Technology_acceptance_model

Tichy, W. 2011. Empirical Software Research: An Interview with Dag Sjøberg, Univeristy of Olso, Norway. *Ubiquity*.

Tilley, S., Müller, H., O'Brien, L., & Wong, K. 2002. Report from the Second International Workshop on Adoption-Centric Software Engineering (ACSE 2002). In *Proceedings of the10th International Workshop Software Technology and Engineering Practice (STEP 2002)*, 74-78.

Part II: Method and practice representation

Simplicity is prerequisite for reliability.

— Edsger W. Dijkstra (How do we tell truths that might hurt?, EWD498, 1975)

This page intentionally left blank

Chapter #3

GBRAM from a SEMAT Perspective

C.M. Zapata
Universidad Nacional de Colombia

L. Castro
Universidad Del Quindío

F.A. Vargas
Tecnológico de Antioquia

1 INTRODUCTION

Software engineering has been evolving towards the standardization of processes and generating a common core of elements. Such a core could provide analysts and stakeholders with the tools— *e.g.*, methods, practices, etc.—for improving several aspects of the software development process. Standardized processes are useful for recognizing and establishing conditions that guarantee the relevance, quality, safety, efficiency, performance, and maintenance of a software application, regardless the software platform or the environment used (Johnson *et al.*, 2012).

The SEMAT kernel supports the modern practice representation of methods like Scrum, XP, RUP, and CDM (Jacobson *et al.*, 2013). A set of elements is defined in order to collect information from the software engineering process and control the activities performed during the software development process—e.g. stakeholder management, requirements elicitation, and software system development, among others.

Several methods focused on goal-oriented requirements specifications like KAOS (Knowledge Acquisition autOmated Specification; Dardenne *et al.*, 1993), I* (Yu, 1995), TROPOS, and GBRAM are suitable to be represented by using the SEMAT kernel. Particularly, GBRAM offers special features to be modeled in the SEMAT kernel. GRAM exhibits a more detailed level than other similar approaches (Tabatabaie *et al.*, 2010) and provides a set of practical guidance and heuristics which are useful for identifying and analyzing organizational goals. Also, GBRAM provides a top-down processing for refining goals. Thus, goals are defined in two phases: goal analysis and goal refinement. In goal analysis, the analyst explores several information sources as a way to identify possible goals and classify them according to goal dependencies. In goal refinement, the goal set is pruned if necessary; goals are analyzed for identifying obstacles. Finally, goals are translated into operational requirements (Antón, 1997).

GBRAM is a goal-based approach for identifying, elaborating, refining and organizing goals for requirements specification (Antón, 1997). Even though GBRAM includes good practices, integration to other methodologies is limited. Some frameworks have been proposed for representing GBRAM. Fabian *et al.* (2010) present a conceptual framework for representing GBRAM, but they only use concepts and notions related to the security requirements engineering. Therefore, relevant concepts belonging to GBRAM are avoided by such a framework. Another work attempting to describe GBRAM is developed by Kavakli (2002). In this framework, GBRAM is described as a collection of method fragments. Then, each fragment prescribes a way of progressing from an initial knowledge modeling state to a target knowledge modeling state. Given that this study is restricted to modeling the GBRAM way-of-working, the full set of elements forming GBRAM is not considered.

In this Chapter we propose a SEMAT-kernel-based representation of the requirements engineering phases belonging to GBRAM. So, we use the SEMAT kernel as a way to settle the foundations of GBRAM and prepare them to be combined with other similar methodologies. As Jacobson *et al.* (2013b) establish, "the kernel provides the mechanisms to migrate legacy methods from monolithic waterfall approaches to more modern agile ones and beyond, in an evolutionary way. It allows you to change your legacy methods practice-by-practice, while maintaining and improving the team ability to deliver."

The Chapter is organized as follows: In Section 2 we describe the main elements of the goal-oriented requirements specification, especially the GBRAM method[1]. In Section 3 we describe the processes carried out by representing the GBRAM method into the SEMAT kernel. Finally, in Section 4 we present some conclusions and future work.

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

2 THEORETICAL FRAMEWORK

2.1 *GORE (Goal-Oriented Requirements Engineering)*

Goals are promoted in this approach as the basis for the software requirements. Hence, the purpose of the system—represented by goals—is included as an intentional point of view of the system. The introduction of an intentional point of view allows for the stakeholders to express their needs in a more natural manner, focusing on what they want—their goals—instead of the way to achieve them (conventional requirements). From the goals, requirements can be derived as ways to achieve such goals (González-Baixauli, Laguna & Prado Leite, 2004).

2.1.1 *I**

Yu (1995) proposed this goal-oriented language which includes nodes representing actors, goals, tasks, and resources. The relationships among nodes are also represented. I* includes the idea of softgoals. The main feature of the business modeling in other fields of requirement engineering is the importance of the agents. An agent is defined as an organizational entity, which has goals and can either perform tasks or use resources for achieving those goals. Also, an agent can help other agents to achieve their goals.

2.1.2 *KAOS*

Dardenne *et al.* (1993) proposed this the tree-based representation of goals which is focused on performing the process of formal analysis of requirements. The process for the mapping of KAOS goals diagram requires the secondary goals subrogating the general goals and the subsequent subrogated goals—considered elementary or atomic goals. Expectations, requirements, and domain properties can be considered as leaf elements in this representation.

2.1.3 *TROPOS*

Castro *et al.* (2002) propose this methodology for the organization modeling, widely used in the early processes of software requirements elicitation. This methodology allows for capture the "what", the "how", and the "why" of software development in the organization. This methodology comprises a detailed description of the system dependencies. Also, the adequate specification of functional and non-functional requirements can be completed by using this methodology.

2.1.4 *GBRAM (Goal-Based Requirements Analysis Method)*

GBRAM was designed for identifying, elaborating, refining, and organizing goals for requirements specification (Antón, 1997). GBRAM includes the initial identification and abstraction of goals for all available information sources. Lastly, the goals are translated into operational requirements by generating a specification requirement document (SRD). Table 1 exhibits the basic concepts included in GBRAM.

Element	Additional information
Goal	Representation of the high-level objectives of the business, organization or system.
Requirement	Specification of the way a goal should be accomplished by a proposed system.
Operationalization	Process of defining a goal with enough detail so that its subgoals have an operational definition.
Achievement goal	A goal satisfied when its target condition is attained.
Maintenance goal	A goal satisfied while its target condition remains true.
Agent	Representation of either the entities or processes aiming to achieve goals related to an organization or system.
Constraint	Requirements should be met for goal completion. A constraint places a condition on the achievement of a goal.
Goal decomposition	The process of subdividing a set of goals into a logical subgroup so that system requirements can be more easily understood, defined, and specified.
Guidelines and heuristics	Elements useful for exploring, identifying, and organizing goals.
Scenario	A behavioral description of a system and its environment arising from restricted situations.
Goal obstacle	Behaviors or goals which either prevent or block the achievement of a given goal.
Inputs	Artifacts which can vary in accordance with the documentation initially available to analysts.
Outputs	Artifacts resulting from activities. The final output of GBRAM is a software requirements document (SRD).

Table 1 Basic concepts included in GBRAM.

The GBRAM method includes two types of activities (Antón, 1997): goal analysis and goal refinement. Goal analysis encompasses: the exploration of documentation for goal identification and the organization and classification of goals. Goal refinement involves the evolution of goals starting from the moment they are identified to the moment they are translated into operational requirements for the system specification. Figure 1 shows the activities (ovals) and artifacts (inclined rectangles) involved in GBRAM.

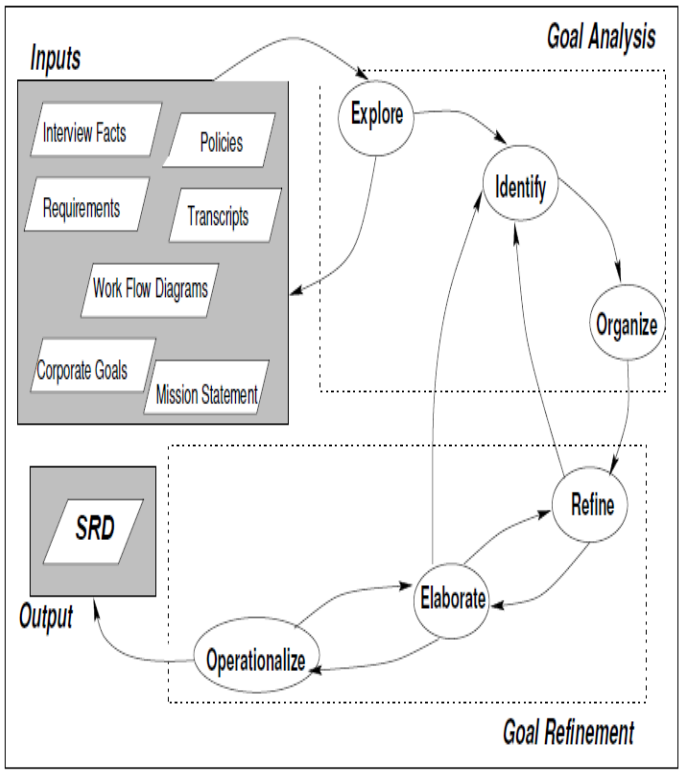


Figure 2. GBRAM activities from (Antón, 1997)

The goal analysis activities can be summarized as follows:

- Explore activities involve the examination of existing documentation for the initial identification of goals.

- Identify activities include the extraction of stakeholders, goals and their responsible agents from the available documentation.
  This type of activities comprises the following sub activities:

  - Identifying goals
  - Identifying stakeholders
  - Identifying agents and agent responsibilities

- Organize activities involve the classification of goals and organization of those goals according to goal dependency relations.
  This type of activities comprises the following sub activities:

  - Eliminating redundancies and reconciling synonymous goals
  - Differentiating goals according to their target conditions
  - Specifying goal dependencies
  - Constructing a goal hierarchy

The goal refinement activities can be summarized as follows:

- Refine activities entail the elimination of redundant goals and reconciliation of synonymous goals.

- Elaborate refers to the process of analyzing the goal set by considering possible goal obstacles and constructing scenarios to uncover hidden goals and requirements.
  This type of activities comprises the following sub activities:

  - Specifying goal obstacles
  - Constructing scenarios in order to consider alternative possible operationalization of goals for the identification of the most plausible solutions
  - Identifying constraints for goal completion. Constraints provide information regarding possible circumstances or conditions a given goal should meet in order to be completed

- Operationalize refers to translating goals into operational requirements for the final requirements specification.

3 GBRAM REPRESENTATION INTO THE SEMAT KERNEL

Elvesæter *et al.* (2013) present a comparison of the Essence 1.0 and draft and the Software & Systems Process Engineering Metamodel (SPEM) 2.0 specifications for software engineering methods from the Object Management Group (OMG). The comparison is based on results from the REMICS research project, in charge of defining an agile methodology for model-driven modernization of legacy applications to service clouds. The REMICS project has participated in the Software Engineering Method and Theory (SEMAT) initiative. SEMAT proposed a new specification named "Essence—Kernel and Language for Software Engineering Methods" (Jacobson *et al.*, 2013) as a response to the Request for Proposal (RFP) "A Foundation for the Agile Creation and Enactment of Software Engineering Methods" issued by the Object Management Group (OMG).

Elvesæter *et al.* (2012) present and discuss how the SEMAT kernel language supports an agile creation and enactment of software engineering methods. The SEMAT approach is illustrated by modelling parts of the Scrum project management practice.

Jacobson *et al.* (2013) combine agile methods with SEMAT with the purpose of taking advantage of good practices in a methodology for the benefit of the other and supporting the development of higher quality software.

SEMAT aims to achieve standardization of software engineering by means of a kernel which is intended to include a set of common software engineering elements and serve as a reference to relate and apply different software development methods in the industry.

In this Chapter we propose the representation of good practices of a goal-oriented requirements specification (GBRAM) by using the SEMAT kernel. We define a set of constructs—mainly work products and activities—of the GBRAM methodology and we relate them to the alphas defined in the SEMAT kernel. Once modeled in the SEMAT kernel, we can explore the integration of the GBRAM method with other similar methods.

GBRAM defines a *practice* as a description of the way to handle a specific aspect of a software engineering endeavor. Besides, a *practice* is considered as set of elements necessary to express the desired work guidance with a specific objective. Therefore, GBRAM method can be represented by the *practice* construct in the SEMAT kernel. Also, GBRAM includes specific work products and activities which can be related to the alphas and activity spaces defined by SEMAT. The subset of alphas and activity spaces needed for describing the GBRAM constructs are shown in Figure 2. Such a figure includes a set of universal alpha elements such as *Opportunity*, *Stakeholders*, and *Requirements*, and their relationships, as well as activity spaces such as *Explore Possibilities*, *Understand Stakeholder needs*, and *Understand the Requirements*. These elements belong to the *Customer* and *Solution* areas of concern.

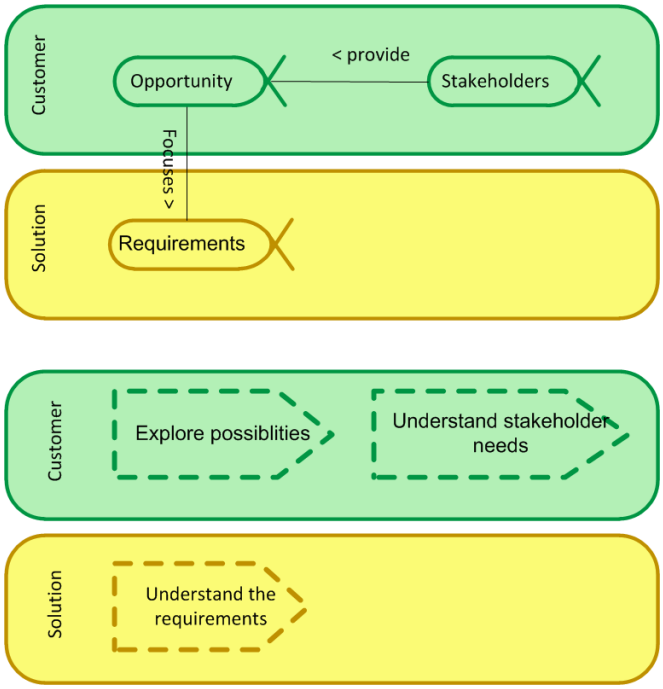


Figure 2. Subset of Alphas and Space activities

Figure 3 and Figure 4 show the representation of the GBRAM method by using the SEMAT kernel. The GBRAM approach is mapped to the alphas of the kernel (*i.e.*, placing *Policies* into Opportunity, *Agents* into Stakeholders, *Goal schemas* into *Requirements*, etc.) and to the activity spaces of the kernel (*i.e.*, placing *Explore* into Explore possibilities, *Identify* into Understand stakeholder needs, *Operationalize* into Understand the requirements, etc.).

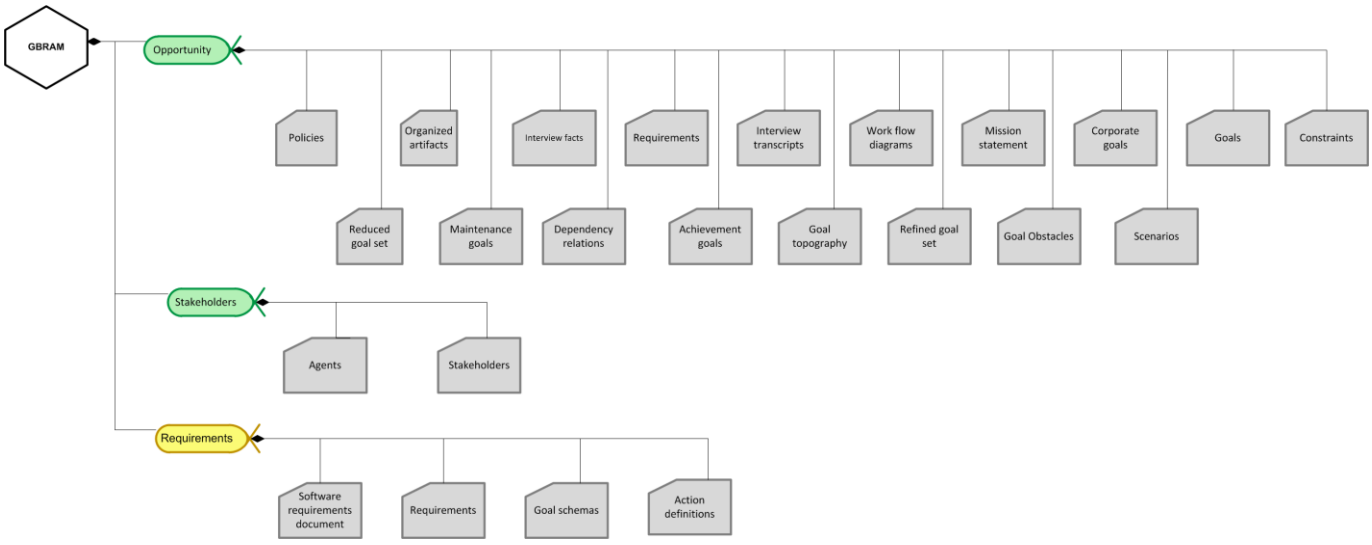


Figure 3. Alphas and Work Products representation

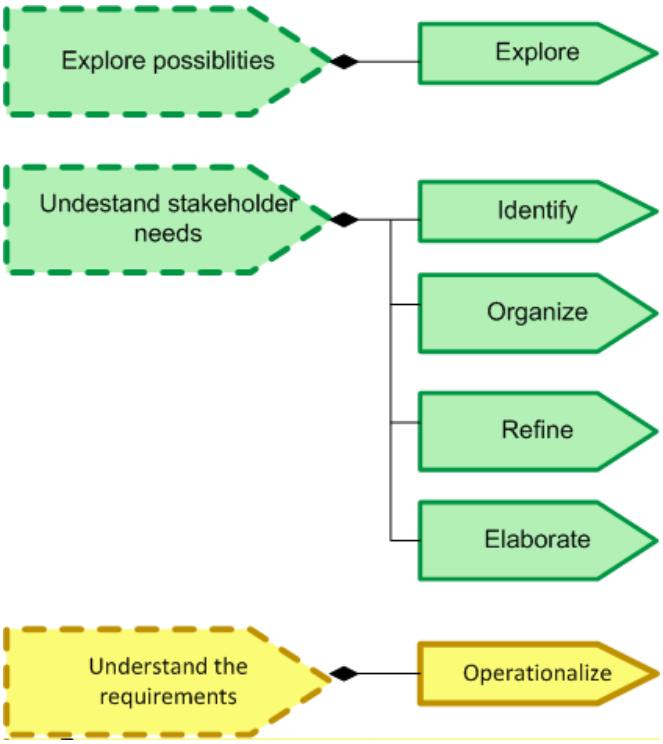


Figure 4. Space activities and Activities representation

4 CONCLUSIONS AND FUTURE WORK

In this Chapter we showed how the SEMAT kernel can support the achievement of the requirements engineering standardization. Also, the SEMAT kernel is directly related to the different software engineering methods of requirements engineering, commonly the representation of the GBRAM method. We can summarize the results of the representation as follows:

- GBRAM activities can be directly traced to the SEMAT kernel activities. The activity spaces containing such activities are related to two areas of concern: customer and solution. Most of the activities are related to the customer, since GBRAM is intended to be an early requirements method.

- SEMAT kernel must permit to establish a direct relationship between different software engineering methodologies, generating that the good practices of any methodology can be used by another.

- Goal-based best practices have a strong interaction with the concern area *customer*. In this way, a solution-based method—like Scrum (Schwaber, 2004), XP (Beck & Andres, 2004; Paulk, 2001), RUP (Kruchten, 2004), and CDM (Oracle Corporation, 2000)—can be suitable for combining with GBRAM.

Some future work can be extracted from the representation we propose in this Chapter:

- Exploring the differences among the several goal-based requirements engineering methods for determining the minimum set of work products to be included in an improved version of such methods.

- Analyzing the usage of other SEMAT kernel elements for representing the information about this kind of methods: for example, actors and phases can be represented by using patterns.

- Using the GBRAM representation obtained for teaching this kind of methods, especially in the industrial environment. Most of the software companies develop software by using the well-known methods (namely RUP, CDM, and, more recently, SCRUM), but they ignore the existence of other methods like the goal-oriented requirements engineering methods.

5 REFERENCES

Anton, A. I. 1997. *Goal identification and refinement in the specification of software-based information systems*.

Beck, K. & Andres, C. 2004. *Extreme programming explained: embrace change*. Addison-Wesley Professional.

Castro, J., Kolp, M., & Mylopoulos, J. 2002. Towards requirements-driven information systems engineering: the Tropos project. *Information systems* 27(6): 365-389.

Dardenne, A., Van Lamsweerde, A., & Fickas, S. 1993. Goal-directed requirements acquisition. *Science of computer programming*, 20(1), 3-50.

Elvesæter, B., Benguria, G., & Ilieva, S. 2013. A comparison of the Essence 1.0 and SPEM 2.0 specifications for software engineering methods. In *Proceedings of the Third Workshop on Process-Based Approaches for Model-Driven Engineering* (p. 2). ACM.

Elvesæter, B., Striewe, M., McNeile, A., & Berre, A. J. 2012. Towards an Agile Foundation for the Creation and Enactment of Software Engineering Methods: The SEMAT Approach. In *Second Workshop on Process-based approaches for Model-Driven Engineering* (PMDE 2012).

Fabian, B., Gürses, S., Heisel, M., Santen, T., & Schmidt, H. 2010. A comparison of security requirements engineering methods. *Requirements engineering* 15(1): 7-40.

González-Baixauli, B., Laguna, M. A., & do Prado Leite, J. C. S. 2004. Análisis de Variabilidad con Modelos de Objetivos. In WER (pp. 77-87).

Jacobson, I., Spence, I., & Ng, P. W. 2013. Agile and SEMAT: perfect partners. *Communications of the ACM*: 56(11), 53-59.

Jacobson, I., Ng, P. W., McMahon, P. E., Spence, I., & Lidman, S. (2013b). *The essence of software Engineering: applying the SEMAT kernel*. Addison-Wesley.

Johnson, P., Ekstedt, M., & Jacobson, I. 2012. Where's the Theory for Software Engineering? *IEEE software*, 29(5), 96.

Kajko-Mattsson, M., Jacobson, I., Spence, I., McMahon, P., Elvesater, B., Berre, A. J., & Seymour, E. 2012. Refounding software engineering: The Semat initiative (Invited presentation). In *International Conference on Software Engineering (ICSE)*, 2012 34th (pp. 1649-1650). IEEE.

Kavakli, E. 2002. Goal-oriented requirements engineering: A unifying framework. *Requirements Engineering* 6(4): 237-251.

Kruchten, P. 2004. *The rational unified process: an introduction*. Addison-Wesley Professional.

Malihe Tabatabaie, Fiona A.C. Polack, and Richard F. Paige. 2010. Evaluating Goal-Oriented Analysis in the Domain of Enterprise Information Systems. Enterprise Information Systems. *Communications in Computer and Information Science* Vol, 109: pp 62-70

Marcal, A. S. C., de Freitas, B. C. C., Furtado Soares, F. S., & Belchior, A. D. 2007. 31st IEEE Mapping CMMI project management process areas to SCRUM practices. In *Software Engineering Workshop* SEW 2007. (pp. 13-22). IEEE.

ORACLE Corporation. 2000. *Oracle method CDM quick tour*.

Paulk, M. C. 2001. Extreme programming from a CMM perspective. *IEEE Software* 18(6): 19-26.

Schwaber, K. 2004. *Agile project management with Scrum*. O'Reilly Media, Inc.

Thórisson, K. R., Benko, H., Abramov, D., Arnold, A., Maskey, S., & Vaseekaran, A. 2004. Constructionist design methodology for interactive intelligences. AI Magazine 25(4): 77.

Yu, E. 1995. *Modelling Strategic Relationships for Process Reengineering*. Ph.D. Thesis, University of Toronto, Toronto.

Chapter #4

Representing Software Specifications in the SEMAT kernel

C. M. Zapata
Universidad Nacional de Colombia. Medellín. Antioquia. Colombia.

P. A. Tamayo
Institución Universitaria de Envigado. Envigado. Antioquia. Colombia.

R. Manjarrés
Politécnico Colombiano Jorge Isaza Cadavid. Medellín. Antioquia. Colombia.

1. INTRODUCTION

Requirements specification is the first step performed in a software development process in order to understand and analyze the stakeholder needs. Several methods have been defined for driving the requirements specification process. Some of them are based on scenarios, business processes, and goals, amongst others. Such methods can be formally represented by using meta-models, pre-conceptual schemas, ontologies, etc., allowing for transformations and establishing constraints among the elements of the method. However, no comparison is possible between the methods, because their elements are not widely accepted and therefore not suitable for specific uses.

The SEMAT (Software Engineering Method and Theory) initiative has been proposed for re-founding the software engineering, by creating a kernel of widely accepted elements and promoting the standardized representation of methods and practices with it. UML (Unified Modeling Language) diagrams and formal languages are selected as the basis for this representation. SEMAT-kernel-based representations are intended to be mixable and comparable, since they are based on a common ground.

As a way to promote the usage of the SEMAT kernel, in this Chapter we develop the representation of the requirements specification phase by using some of the essential elements of the SEMAT kernel, like alphas, activity spaces, work products, and the card-based usage of the kernel. The resulting representation is suitable for analyzing, comparing, and complementing several existing methods.

This Chapter is organized as follows: Section 2 is devoted to the conceptual framework basis of this project; in Section 3 we present the background; the proposed representation is included in Section 4; conclusions and future work are established in Section 5.

2. CONCEPTUAL FRAMEWORK[1]

Several software development methods have been used to formalize the stakeholder requirements and convert them into software specifications, comprising a set of activities to carry out the conversion (Jacobson et al., 2001). Such software development methods commonly use the Unified Modeling Language (UML) for representing the conceptual schema of the software system to be built (Tamayo, 2007).

A software specification includes a set of elements for supporting all the involved actors in the process of analyzing and understanding the stakeholder needs. Indeed, the end user of the product describes what he really wants to get in this specification. Consequently, the software specification is useful for checking the correctness of the source code, since it describes interfaces in detail, such as user, software, hardware, and communications, as well as customer requirements and system attributes among others (Cobo and Morales, 2013).

The requirements specification is commonly made by using either natural language descriptions or knowledge representation languages (KRL), such as the UML diagrams. Natural language specifications lead to ambiguity problems, inaccuracy, and inconsistency. (Falgade, 2011).

Most of the software development methods use a graphical modeling language called UML. In 1997, the Object Management Group (OMG) accepted UML to be the standard modeling language for developing object-oriented systems. UML offers a wide variety of diagrams to visualize the system from several perspectives. 13 diagrams are included the more recent UML specification.

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

The UML taxonomy is presented in Figure 1 (OMG, 2014). The contents of the diagrams are the following:

- Class Diagram: describes the static structure of a system. The class diagram is used during the analysis and design phases of the software development process. The main goal of this diagram is to define the conceptual design of the information the system will handle and the components responsible for the operation.

- Composite Structure Diagram: shows the internal structure of a classifier, including its interaction points with other parts of the system.

- Component Diagram: specifies a set of constructs which can be used to define software systems of any size and complexity. The diagram specifies a component as a modular unit with well-defined interfaces that can be changed in its environment.

- Deployment Diagram: specifies a set of constructs which can be used to define the execution architecture of systems. The constructs are represented by nodes.

- Object diagram: uses a subset of class diagram elements for emphasizing the relationship between instances of the classes at some instance in time.

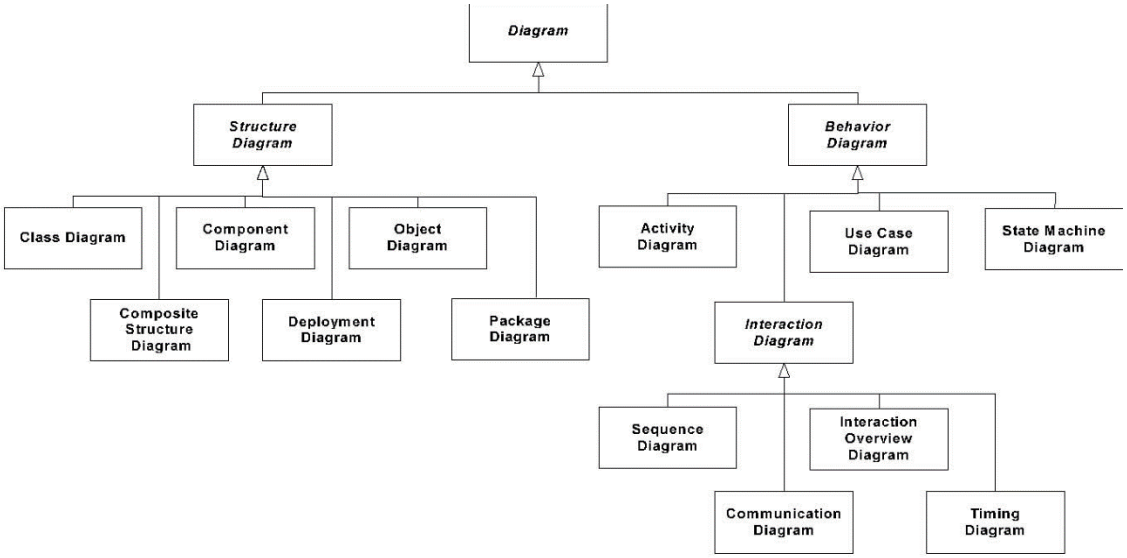


Figure 1. UML taxonomy. Taken from OMG (2014)

- Package diagram: describes how a system is divided into logical groups and shows the dependencies among these groups.

- Activity diagram: describes the functionality of the software in a high level of abstraction, emphasizing the sequence and conditions for coordinating lower-level behaviors.

- Use Case Diagram: describes the functional requirements of the system in terms of sequences of actions, including variants the system or other entity can perform by interacting with actors of the system.

- State Machine Diagram: models the behavior of a discrete system by using finite state transitions. Furthermore, state machine diagram can be used to represent the usage protocol for a part of the system.

- Sequence Diagram: describes an interaction by focusing on the sequence of messages exchanged, along with their corresponding specifications of events in the lifelines.

- Communication Diagram: shows how objects cooperate to execute a transaction, *i.e.*, how specific instances of the classes work together to achieve a common goal.
- Interaction Overview Diagram: promotes overview of the control flow and implements the class diagram associations by passing messages from one object to another.

- Timing Diagram: describes the behavior of individual classifiers and their interactions, focusing on the occurrence time of events causing changes in the modeling conditions of lifelines.

3. BACKGROUND

Software systems are documented with fully-structured descriptions, like functional specifications, prototypes, designs, and source code. Some other descriptions are

used, *e.g.*, poorly structured narrative requirements, user manuals, test plans, and maintenance guides. For such descriptions, Scacchi (2001) designs and implements a hyper-textual structure allowing for the identification and tracking of relationships between several semi-structured descriptions of the same system, in order to configure, validate, and maintain consistency of interrelated software descriptions as they evolve. However, this proposal does not allow for comparing the obtained specifications with those ones belonging to other projects, given that they have no fixed elements.

Wongthongtham *et al.* (2008) and Ji (2010) define an ontology for software engineering consisting of 362 concepts and 303 relations. The goal of this ontology is to facilitate communication among team members, and provide coherent understanding of the domain knowledge and project data. The proposed representation can be used by multiple teams. However, no comparison is possible among the results obtained by each, since each team can work with different elements from the ontology.

Prakash and Rizwan (2013) model software specifications written in natural language by using a simple graph, obtained with the application of techniques of natural language processing. Obtaining such a graph requires knowledge about natural language processing, which is not so common among the actors involved in the software development process.

On the other hand, other approaches are used to represent some elements of the software specification in different formalisms. However, they don't cover the whole process. For example, Estrada *et al.* (2002) generate a software requirements specification from a business model. The specification consists of a use cases model and their associated scenarios, which is the starting point to generate a description of the system behavior and prototypes for user interfaces.

The Software Process Engineering Meta-model (SPEM) is standard the defined by the OMG (Object Management Group) for representing software development processes. SPEM allows for modeling, documenting, presenting, managing, and exchanging software development processes and their components by providing syntax and a common structure for each aspect of the process (SEPM, 2008). Moreover, Szyrko and Rubio (2010) validate the implementation of the practices defined at the organizational level related to the reference model. Hence, they analyze the impact of any changes of both the reference model and the organizational process. However, this method is not used for all software development processes and therefore the evaluation and comparison of results is difficult to carry out and use in the future.

Finally, Zapata (2007) defines a pre-conceptual schema for automatically obtaining different diagrams used in the software specification. However, when using the defined schema, a comparison among the different methodologies used for the specification of software is not possible, because no fixed starting and ending points are provided.

4. SOFTWARE SPECIFICATIONS REPRESENTED IN THE SEMAT KERNEL

As a way to solve the problems found in the previous section, a possible solution is representing the software specification by using the elements found in the SEMAT kernel. We need to represent first the practice (see the complete set of symbols in the Preface of this book) and the linkage with the alpha *requirements* (see Figure 2). Such a representation is based on the things we always use when writing software specifications, in this case the alpha *requirements*. This practice is intended to be linked to the area of concern *solution.*

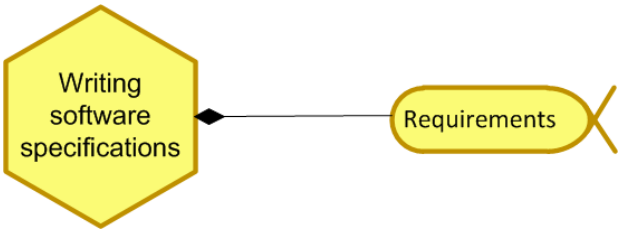


Figure 2. Representation of the practice "writing software specifications"

The software specification practice is performed to capture, understand, and analyze software requirements. The work products associated with the alpha *requirements* are the textual representation of the stakeholder domain and comprises several points of view of the software (structure, behavior, and interaction). Such points of view are represented by using UML diagrams. The composition of these work products allows for the analyst to transform the stakeholder needs into formal software specifications. The main elements of the SEMAT kernel are useful for representing the aforementioned information: work products are the UML diagrams and the formal specification, the only activity is *transform artifacts*, belonging to the activity space *understand the requirements*, and the pattern is useful for representing the role *analyst* (see Figure 3). Also, these elements are the basis for comparing and combining this practice with other practices and methods.

In the context of SEMAT-kernel-based representations, work products can be refined by using cards. The aim of the work product *domain representation* is to visually represent real-world objects in a stakeholder domain. So, the first draft of this work product is the *stakeholder discourse*, and the final artifact is the *knowledge representation.* Regardless the software development method used, the knowledge representation can be shaped by using pre-conceptual schemas, user stories, BPMN

(Business Process Modeling Notation) diagrams, and so on. Figure 4 depicts the *domain representation* card.

The structure of the software specification refers to the concepts found in the stakeholder domain and the relationships between them. The diagrams involved in the structure allow for modeling the main features of the concepts and the relationships between them. The level of detail of the diagrams is increasing while we are approaching to the system specifications. Indeed, the main structural features of the software system are more clearly defined in the component diagram than the object diagram. Composite structure diagram exhibits the highest level of detail, since this diagram is a mixture of several others. Figure 5 exhibits the *structure diagrams* card.

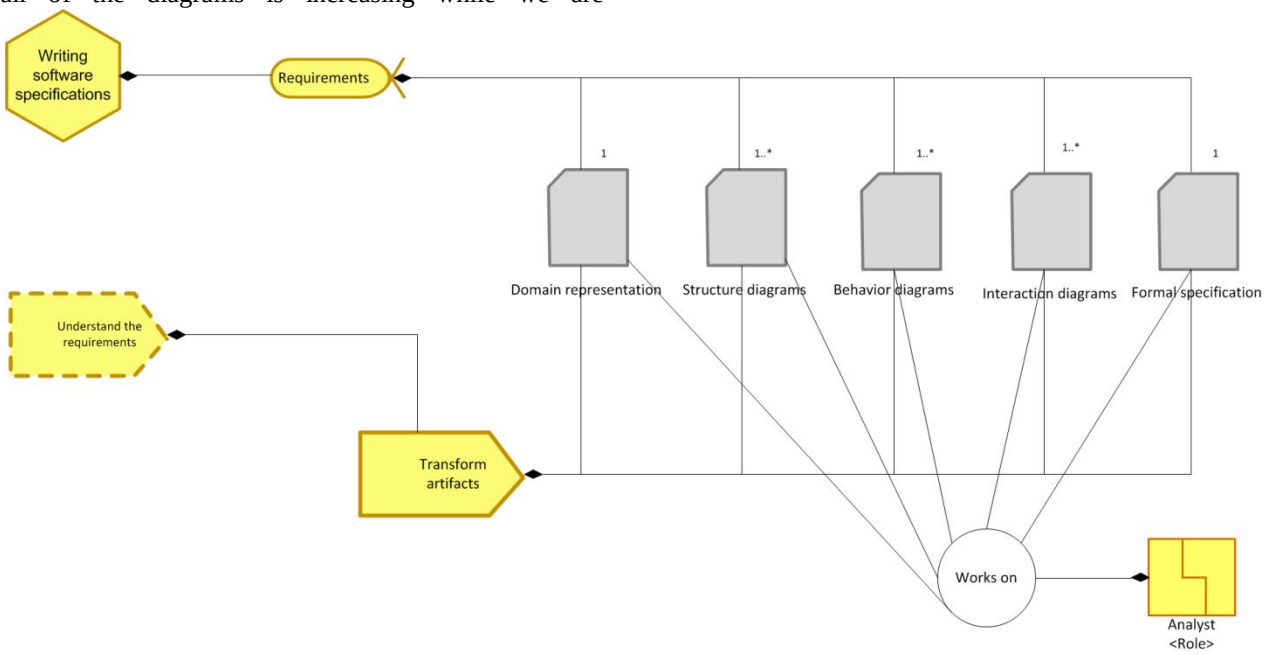


Figure 3. Representation of work products, activities, roles, and activity spaces of the alpha *requirements*.

Behavior diagrams are intended to specify the effects of actions and events. The interaction among objects—a subset of such effects—is represented by a subset of behavior diagrams, called interaction diagrams. Figure 6 and 7 respectively show the work products related to behavior diagrams and interaction diagrams. Such work products aim to model the dynamic aspects of the software specification.

The main goal of the *formal specification* work product is to precisely describe the properties the software should have, particularly, its specification. The card describing this work product is presented in Figure 8. The complementary statements can be written in languages like OCL (Object Constraint Language) and SQL (Structured Query Language) and the full formal specification can be written in languages like B, Z, Oasis, etc.

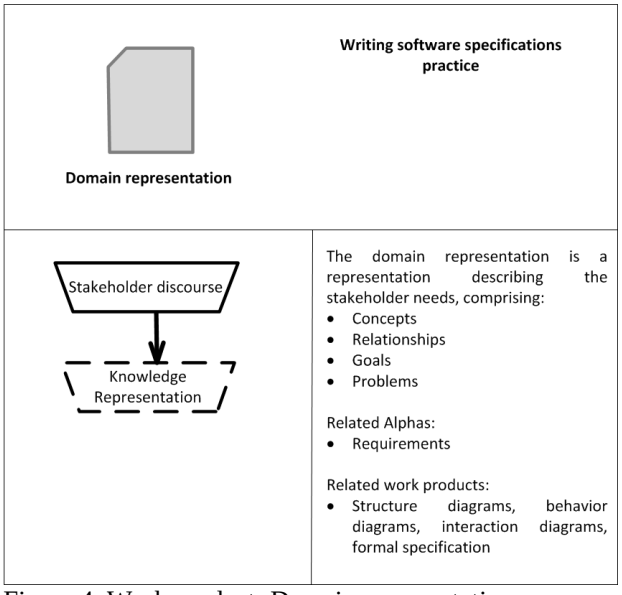


Figure 4. Work product: Domain representation

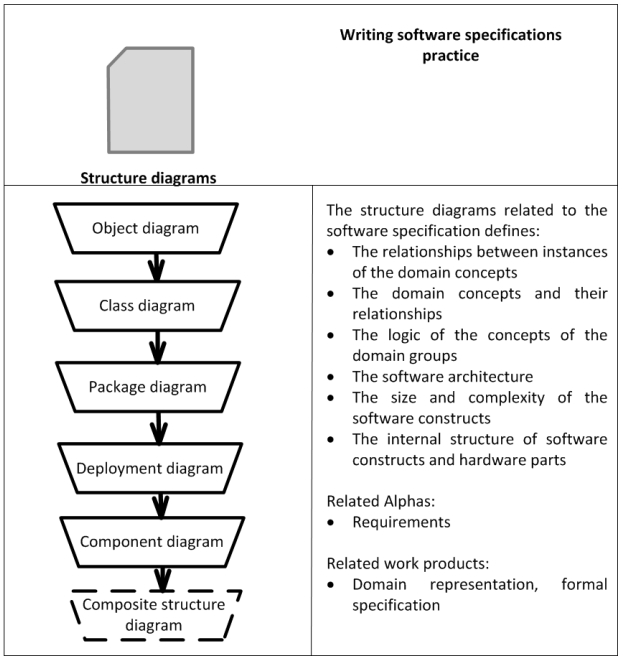


Figure 5. Work product: Structure

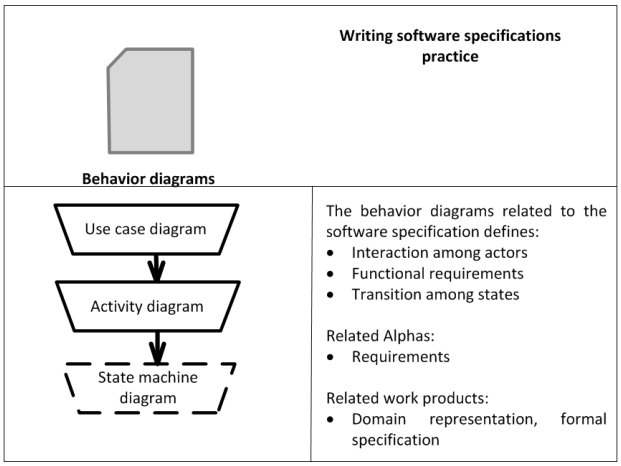


Figure 6. Work product: Behavior.

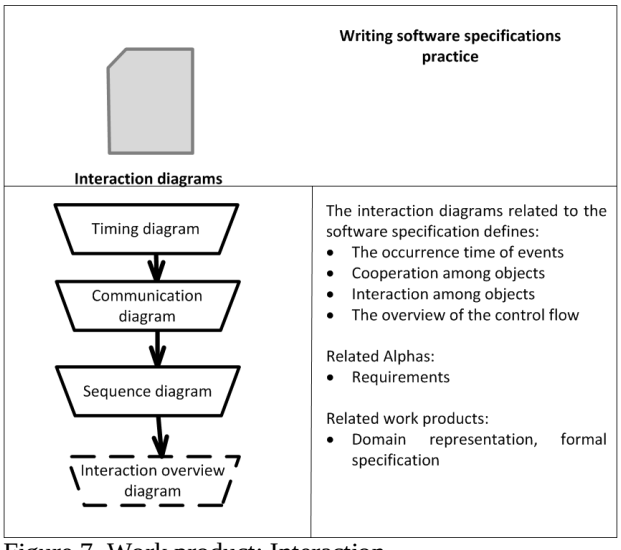


Figure 7. Work product: Interaction.

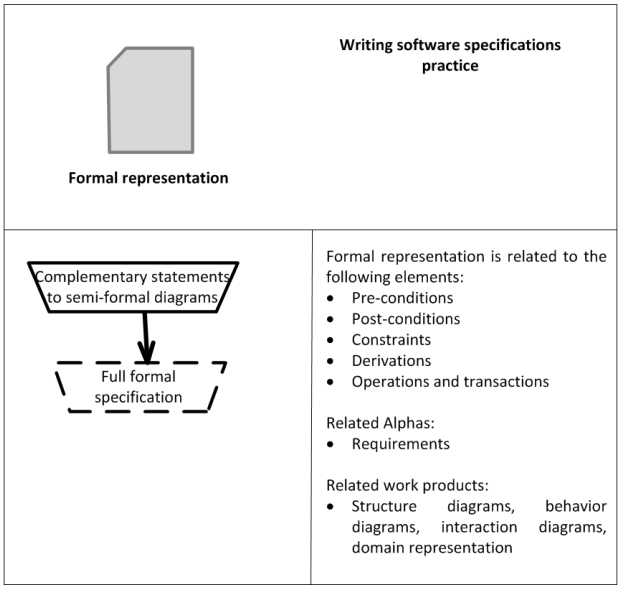


Figure 8. Work product: Formal specification.

As a summary, the *writing software specification* practice is an artifact transformation process from the stakeholder discourse to the formal specifications. Commonly, some parts of the entire process are supported by automated or semi-automated processes, but the entire process is not completely automated and the analyst is responsible for the final quality of the formal specification.

5. CONCLUSIONS AND FUTURE WORK.

Software specification comprises a set of artifacts ranging from high-level to low-level requirements. Since some authors fail in creating meta-models and representation artifacts for studying and combining software specifications in several software development methods, in this Chapter we proposed a SEMAT-kernel-based representation of the practice *writing software specifications*. The usage of pre-defined terms (*e.g.*, requirements) and actions (*e.g.*, understand the requirements) will function as a linkage between our representation and any other practice represented by using the SEMAT kernel. In this sense, the practices can be compared and combined if necessary.

Some future work we can devote for complementing our proposal is:

- Completing the representation of the *writing software specifications* practice with other elements from the SEMAT kernel. For example, we can use resources, competencies, and levels of competency for expressing some other information about the practice.
- Characterizing some other specification practices.
- Differencing and assembling this representation with other similar representations. In this sense, we can

combine practices, so we can create new methods based on the successful elements of previous methods.

- Defining assessment methods for promoting valitation of the SEMAT-kernel-based representations.

6. REFERENCES

Cobo, A. & Morales H. 2013. *Especificación de requisitos de software* (SISCoop, 2013). Available: http://dspace.espoch.edu.ec/bitstream/123456789/188/1/EspecificacionRequerimientosSoftware.pdf.

Estrada, H., Martínez, M., Pastor, O., & Sánchez J. 2002. Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios: un enfoque basado en metas. In *V Workshop on Requirements Engineering*. Valencia. Spain.

Fagalde P. 2011. *Artefactos de especificación de requerimientos de usabilidad*. Bachelor Thesis. Universidad de Buenos Aires. Buenos Aires. 81 p.

Jacobson, I. Booch, G. Rumbaugh, J. 2001. *El Proceso Unificado de Desarrollo de Software*. Addison Wesley.

Ji, X. Q. 2010. Software Engineering Knowledge Representation based on Ontology for Multisite Software Development Asia-Pacific Conference on Wearable Computing Systems.

OMG (Object Management Group). 2014. *OMG Unified Modeling Language* (OMG UML), Superstructure Version 2.4., Available: http://www.omg.org.

Prakash, R. & Rizwan, V. 2013. Representation of knowledge from software requirements expressed in natural language. In *Sixth International Conference on Emerging Trends in Engineering and Technology*. December 16-18, Nagpur, India.

Scacchi, W. 2001. *Hypertext for Software Engineering. Encyclopedia of Software Engineering*, 2nd. Edition, New York: John Wiley and Sons, Inc.

Szyrko, P. & Rubio, D. 2010. Definición de un metamodelo para la validación de procesos de software organizacionales basados en modelos estándares. In *XII Workshop de Investigadores en Ciencias de la Computación*. Calafate, Santa Cruz, Argentina.

Tamayo, P. A. 2007. *Elaboración de diagramas de caso de uso a partir de modelos verbales*. M.Sc. Thesis, Universidad Nacional de Colombia, Sede Medellín.

Wongthongtham, P., Kasisopha, N., Chang, E., & Dillon, T. 2008. A Software Engineering Ontology as Software Engineering Knowledge Representation. In *Third 2008 International Conference on Convergence and Hybrid Information Technology*. IEEE.

Zapata, C. 2007. *Definición de un esquema preconceptual para la obtención automática de esquemas conceptuales de UML*. Ph.D. Thesis. Universidad Nacional de Colombia. Medellín.

Chapter #5

Representation of TSP framework into the SEMAT kernel based on the best practices of Project management from CMMI-DEV

B. Manrique-Losada; G. P. Gasca-Hurtado & M. C. Gómez-Álvarez
Facultad de Ingeniería, Universidad de Medellín, Colombia

1 INTRODUCTION

The Team Software Process (TSP) framework guides engineering teams in the implementation of products in developing software projects. TSP includes practices from the CMMI-DEV model for improving the quality and productivity of engineering teams.

The SEMAT Essence kernel can be described as a set of practices of software development methods. A practice is the unit of adoption, planning, and execution of a process and it is given priority over the process as a composition of practices. Engineers should work out the details of team-building and team working for themselves, besides to execute the activities related to the software development. Since defining these details involves considerable skill and effort, engineering teams generally follow *ad-hoc* team-building and teamwork processes without a formal and specific guidance. The Essence kernel can be used in cases where team whether or not has a documented method. The elements of the kernel are always prevalent in any software endeavor (OMG 2012). In this regard, a systematic and guided specification of how the TSP practices should be developed can be suitable by using general notations as provided by SEMAT.

In this Chapter we define a representation of TSP framework into the SEMAT kernel, based on the best practices of project management from CMMI-DEV. This approach is a representation of the specific practices comprised in TSP as cycles and phases, by using the meta-model components of the kernel. The representation proposal of the TSP framework includes the main elements of the SEMAT Kernel such as: areas of concern, alphas, activity spaces, and competencies. This is an alternative to facilitate the TSP adoption in software development organizations.

The Chapter is organized as follows: in Section 2 we present the conceptual framework about TSP, and CMMI-DEV Model[1]. In Section 3 we describe the background related to approaches to knowledge representation in SEMAT Essence and graphical specifications of TSP. In Section 4 we present our representation proposal of TSP framework into the SEMAT kernel based on the best practices identified from the specific activities by TSP phase/cycle. Finally, in Section 5 we conclude and state future work.

2 CONCEPTUAL FRAMEWORK

2.1 *Team Software Process*

TSP was launched in 1996 by Watts Humphrey with the objective of providing an operational process for helping engineers to consistently make quality work (Humphrey, 1999). TSP is a framework guiding engineering teams in developing high quality software products. TSP is used for improving the quality and productivity of engineering teams and help to meet cost and schedule commitments (Huesca, 2010). TSP also provides a framework attached to the Personal Software Process (PSP) (SEI 2011).

The framework defines the steps needed for establishing an effective team working environment. Without specific guidance, engineers should work out the details of team-building and team working for themselves. Since the definition of these details involve considerable skills and effort and few engineers have the experience or time for working out all of the necessary details, engineering teams generally follow *ad-hoc* team-building and teamwork processes. This situation wastes time and it often produces poorly functioning teams (SEI 2011).

The main elements of the TSP process include both as PSP and TSP elements because the engineers must be trained in these skills before they can participate in TSP team building or follow the defined TSP process (see Figure 1; Humphrey, 1999).

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

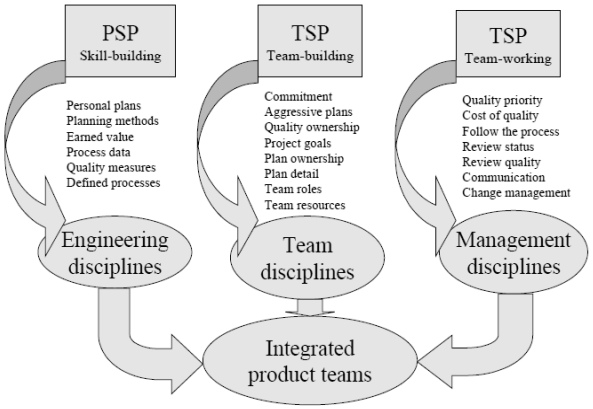


Figure 1. TSP Team-Building. (Humphrey 1999)

2.2 *CMMI-DEV Model*

The purpose of the CMMI-DEV model is to provide guidance for applying CMMI best practices in a software development organization. These best practices are focused on the activities necessary for developing high quality products and services (SEI 2010).

The CMMI-DEV model is based on the CMMI Model Foundation or CMF (*i.e.*, model components common to all CMMI models and constellations or interest areas) and incorporates work by development organizations to adapt CMMI for use in the development of products and services (ITSQC, 2005).

This model facilitates the work of interested people on process improvement in a development environment. This model also helps to understand the concept of Capability Maturity Models and provides information to begin improving the development processes, because this model is intended for organizations interested in a reference model for an appraisal of their development processes.

The structure of CMMI-DEV includes 22 process areas, distributed as follows: a) 16 core process areas, and one of them is a shared process area (see Figure 2), and b) 5 development specific process areas: Requirements Development (RD), Requirements Management (REQM), Technical Solution (TS), Product Integration (PI), Verification (VER) and Validation (VAL). These process areas describe practices focused on the activities of the developer organization.

CMMI for Development (CMMI-DEV) provides practices associated with the specific development by using best practices addressing development activities applied to products and services. Such practices are related to the product lifecycle from beginning to delivery.

Abbreviation	Name	Area
CAR	Causal Analysis and Resolution	Support
CM	Configuration Management	Support
DAR	Decision Analysis and Resolution	Support
IPM	Integrated Project Management	Project Management
MA	Measurement and Analysis	Support
OPD	Organizational Process Definition	Process Management
OPF	Organizational Process Focus	Process Management
OPM	Organizational Performance Management	Process Management
OPP	Organizational Process Performance	Process Management
OT	Organizational Training	Process Management
PMC	Project Monitoring and Control	Project Management
PP	Project Planning	Project Management
PPQA	Process and Product Quality Assurance	Support
QPM	Quantitative Project Management	Project Management
REQM	Requirements Management	Project Management
RSKM	Risk Management	Project Management

Figure 2. Process area Core from CMMI

3 BACKGROUND

3.1 *Approaches to knowledge representation in the SEMAT Essence kernel*

The members of the SEMAT initiative have presented some approaches for representing practices and methods by using the SEMAT Essence kernel. Some of them are as follows:

Jacobson *et al.* (2010) developed several graphical representations in the framework of the SEMAT initiative, by trying to explain and disseminate the principles of the kernel by means of examples. One of them is the approach to model the practices of use-case modeling by using SEMAT cards comprising: the competencies *analysts* and *customer representative*; the activities carried out in the activity space; and the related descriptive elements. In Figure 3, we show the example for the practice *find actors and use cases*. Also, Jacobson *et al.* presented as sample the general framework of RUP for trying to characterize the elements of a software methodology (see Figure 4).

Berre (2012) presented an approach for SCRUM Essentials Practices in terms of SEMAT elements. He adds two Alphas, *Requirements Item* and *System Element*, directed to the Requirements and the Software System, as well as a *Bug* for monitoring the health of the Software System. The approach is shown in Figure 5.

3.2 *Approaches for representing the TSP framework*

TSP is presented in the state of the art in terms of its organization and operation (Yu *et al.* 2009). The TSP structure is related to the content from the management aspects and the TSP process is associated with the technical aspects. Depictions of TSP are mainly based on the dynamic aspects, as a visual guide for showing how TSP uses several development cycles to build the final product, and usually they lack a

standard notation for facilitating the application of the framework.

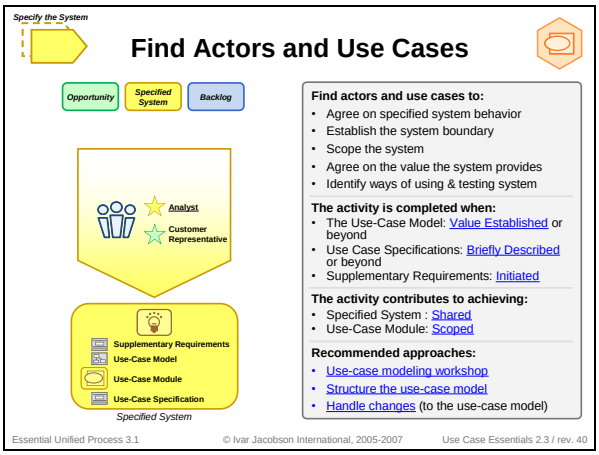


Figure 3. Practice of use-case modeling in SEMAT. (Jacobson *et al.*, 2010)

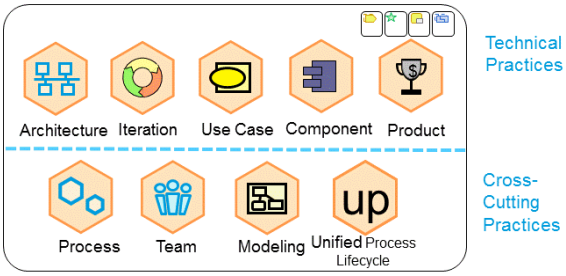


Figure 4. General Framework of RUP (Jacobson *et al.*, 2010)

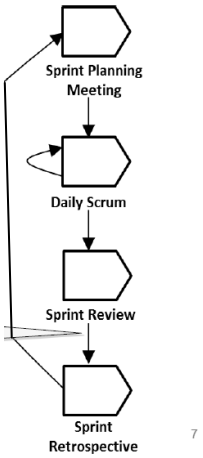


Figure 5. Approach for Scrum Essentials Practices in SEMAT (Berre, 2012).

Regardless of the organization, TSP has the following components: formation, launch, and work (Humphrey 1999). We have only found graphical representations about teamwork and its five parts of content of team, as we show in Figure 6.

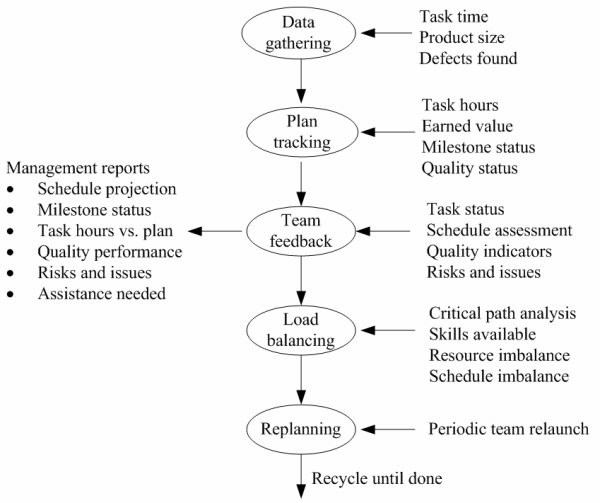


Figure 6. Representation of TSP team-working (Humphrey, 1999)

Regarding the TSP operation, McAndrews (2000) presents a chart of TSP process as a set of phases and activities, as we show in Figure 7.

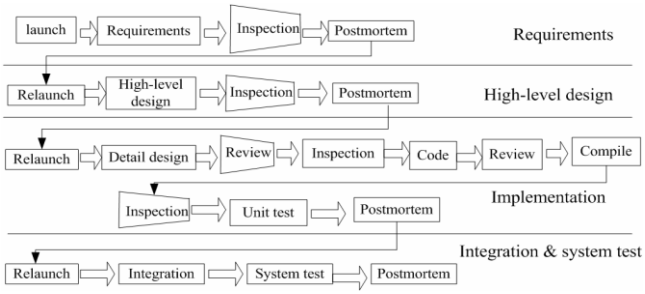


Figure 7. Representation of TSP Process (McAndrews, 2000).

In addition to the previous chart, some representations of the key points on TSP were developed in the TSP Symposium 2012 (SEI, 2012). We found interesting charts about context, related concepts, goals, key practices, and functions of TSP. Some of them are presented in Figure 8.

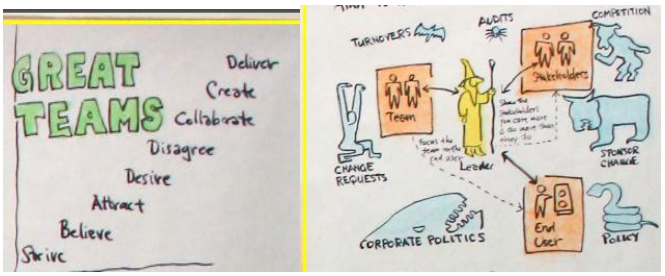


Figure 8. Representation TSP key points at TSP Symposium (SEI 2012).

Similar to the previous representations, several authors have developed illustrations for different application contexts. They try to specify the relationships between TSP and specific stages of software development, as we present in Figure 9

to show the usage of Architecture-Centric Engineering process in a TSP Project.

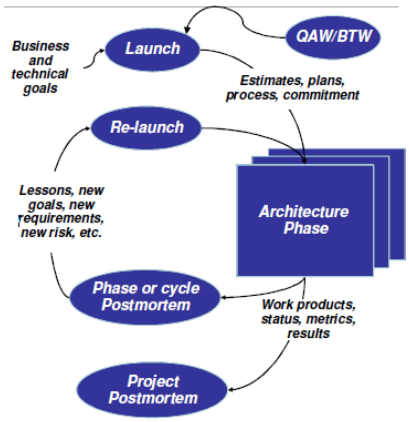


Figure 9. Architecture-Centric Engineering process on a TSP Project Representation (Carballo *et al.*, 2011).

No representations specifically oriented to TSP practices were found in the state-of-the-art review, related to the notation given either by the SEMAT Essence kernel or any other formalism.

4 PROPOSAL OF TSP REPRESENTATION INTO THE SEMAT KERNEL

4.1 *Method for defining the representation*

The following phases were defined to make the representation of a model or framework taking into account the SEMAT kernel structure:

Basic Phase: In this phase we studied SEMAT including kernel elements and language. This phase comprises two activities:

Activity 1.1 Study SEMAT Kernel structure; in this activity, the main elements of SEMAT Kernel were identified: areas of concern, activity spaces, alphas, alpha states, and competencies.

Activity 1.2 Understand SEMAT Essence Language; this activity allowed for accessing the knowledge of the graphical representation associated with each element of SEMAT Kernel.

Specific Phase: In this phase we analyzed the framework structure and the components of the framework to be represent into the SEMAT kernel. This phase comprises two activities:

Activity 2.1 Identify the model or framework to represent into SEMAT; in this activity, both CMMI Model and TSP Framework were studied. As a result of the analysis of SEMAT areas of concern, activity spaces and alphas; a similarity between the TSP and SEMAT objectives, from the point of view of the team management is identified. This similarity determines the relevance of representing TSP Framework into the SEMAT kernel in order to propose an alternative approach to the new proposal SEMAT for companies that adopt TSP framework for managing their software development teams.

Activity 2.2 Define the TSP framework components from SEMAT Kernel elements; in this activity, the following SEMAT elements were identified for each TSP framework phase: a) areas of concern, b) activity spaces, c) alphas, d) alpha states, e) activities, f) work products, and g) competencies. For example, in the *Planning* phase the following elements were identified for the alpha *work*:

- Activity space: coordinate Activity
- Alpha state: prepared
- Activities: (1) define the tasks plan and (2) define quality plan of the project.
- Work products: (1) project plan and (2) schedule form.
- Competencies: (1) leadership and (2) management.

Representation Phase: The objective of this phase is to represent TSP into SEMAT according to its language and kernel. These two activities are proposed for achieving this objective:

Activity 3.1 Design the graphical representation of TSP framework and its phases according to the SEMAT Essence language; we established a graphic structure for each of the phases of this framework, grouping them according to the scripts of TSP.

Activity 3.2 *Describe the graphical representation of each phase of framework*; in order to supplement the graphical representation, we provided an explanation that facilitates the understanding of the representation proposal.

4.2 *Representation Proposal*

According to the method defined and presented in the previous Section, we design a general framework of the proposal, based on the TSP structure and flow—in terms of cycles and sequential phases as we show in Figure 10. We define the set of activities for each phase as practices for the representation in the SEMAT kernel.

The general framework was designed by specifying for each activity/cycle the alpha states progressed by the activities comprised in the TSP structure, as we present in Figure 11. Thus, for each TSP phase, we identified the alpha involved and its respective alpha states. Such alpha states are reached by means of the development of a set of activities, forming practices. We represent in a specific chart the practices for each alpha (α) by means of activity spaces composing by the sequence of activities, the work products produced or updated, and the competences involved.

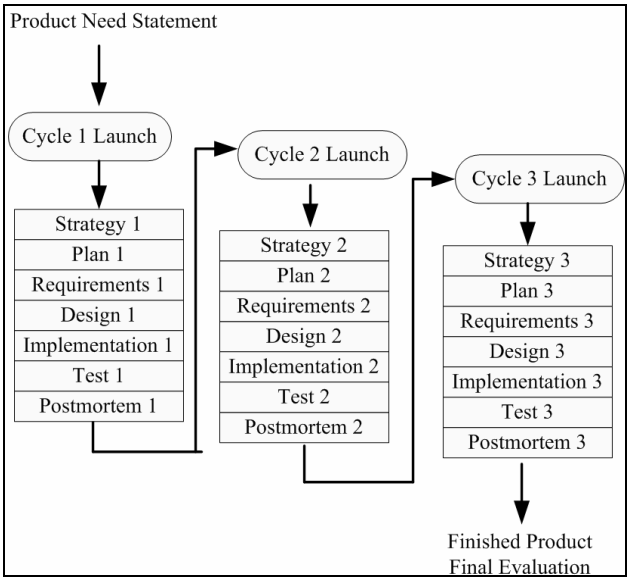


Figure 10. TSP structure and flow (Yu *et al.*, 2009)

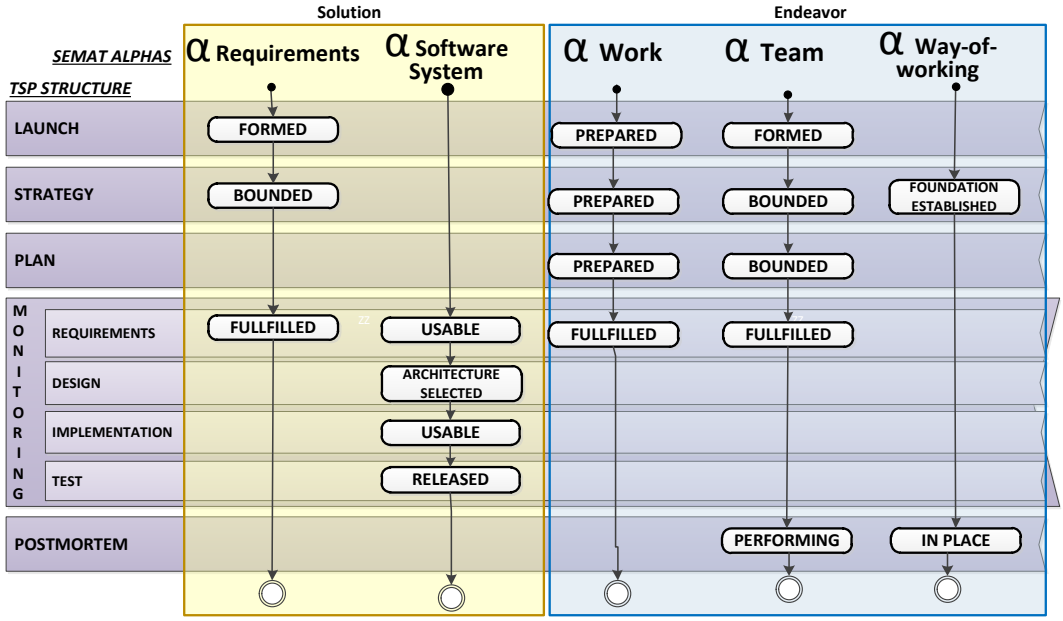


Figure 11. General framework of the TSP representation in the SEMAT kernel

The *Planning phase* of TSP is represented in Figure 12. As an example for reading and interpreting our representation proposal, we will explain such a representation as follows: In the planning phase a team wants to achieve a goal by progressing the alphas *work* and *team*. In order to do this, the competencies *management* and *leadership* assess the current state of each alpha (*e.g.*, α *work* is *prepared*) and then, they have to develop a set of activities (in the frame of an activity space *coordinate activity*) in order to act as the goal of the development effort. According to the previous dynamic, we present the other TSP phases in Figures 13-16.

5 CONCLUSIONS AND FUTURE WORK

In this Chapter we proposed an approach for representing the TSP framework by using the SEMAT kernel. SEMAT is a set of composed practices of software development methods while TSP is a proposal that guides engineering teams in developing software intensive products.

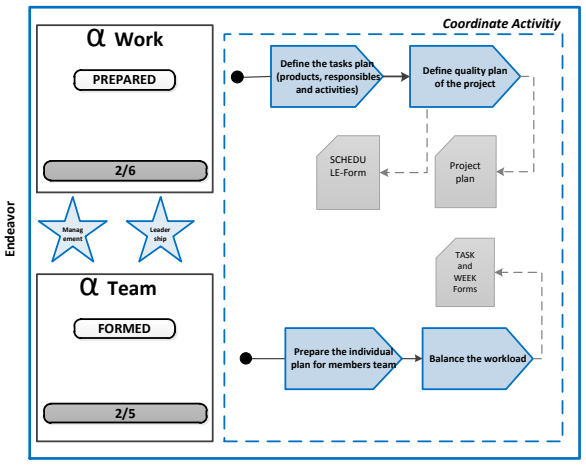


Figure 12. Planning Practices

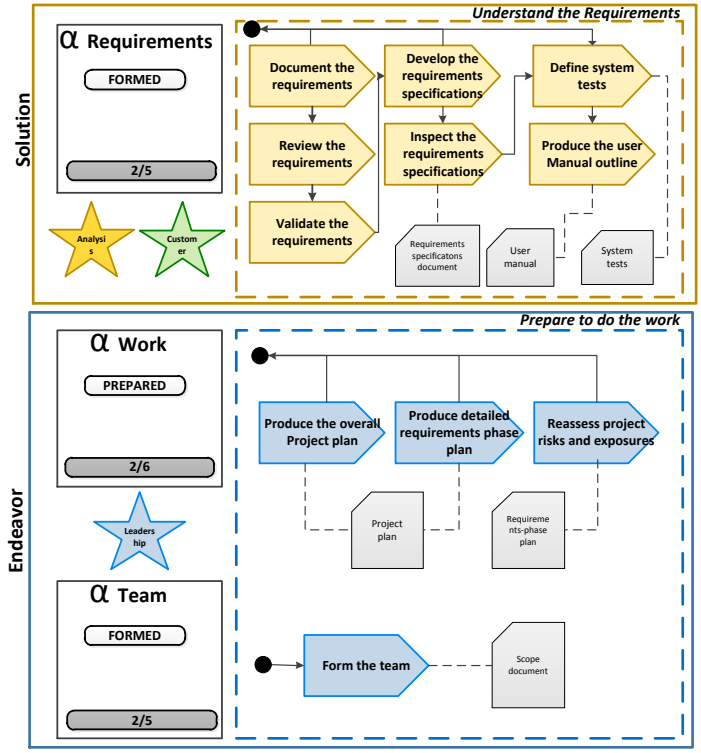


Figure 13. Launch Practices

Both proposals are oriented to improve the development of software process. The TSP-proposal is focused on achieving the implementation of the CMMI model in the software development organizations. Nowadays, the industry and the academy have to incorporate new initiatives, such as SEMAT, in order to improve the implementation of best practices in development software organizations. This new form includes the discipline of the development software team to define the kernel approach from SEMAT.

For including our approach (TSP into SEMAT) was necessary some organization about the cycles, phases, and activities from TSP for obtaining a SEMAT representation in terms of practices. The cycles and phases were defined from the study and understanding both SEMAT and TSP main concepts. From this basis, we analyze the components of the SEMAT kernel and we define the representation of each TSP phase during the development process.

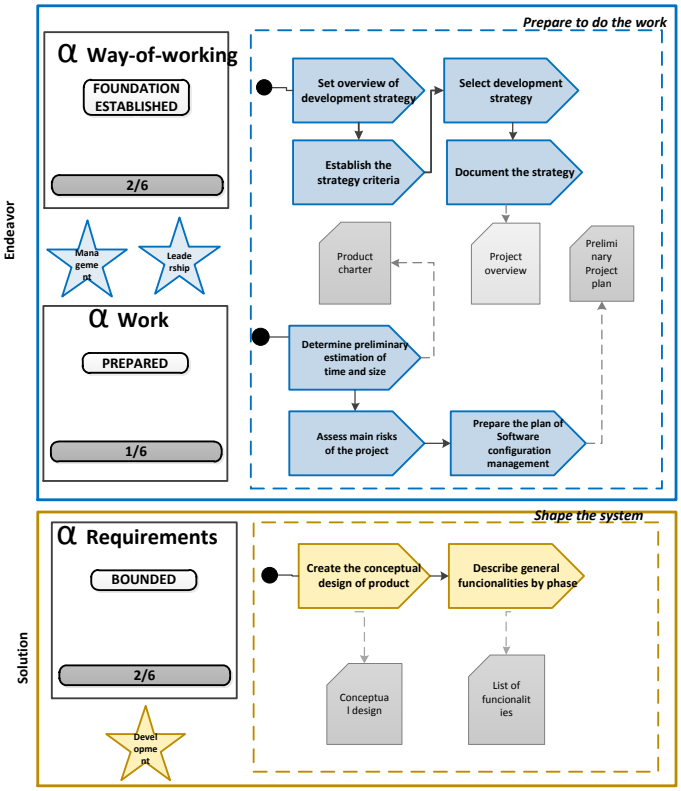


Figure 14. Strategy Practices

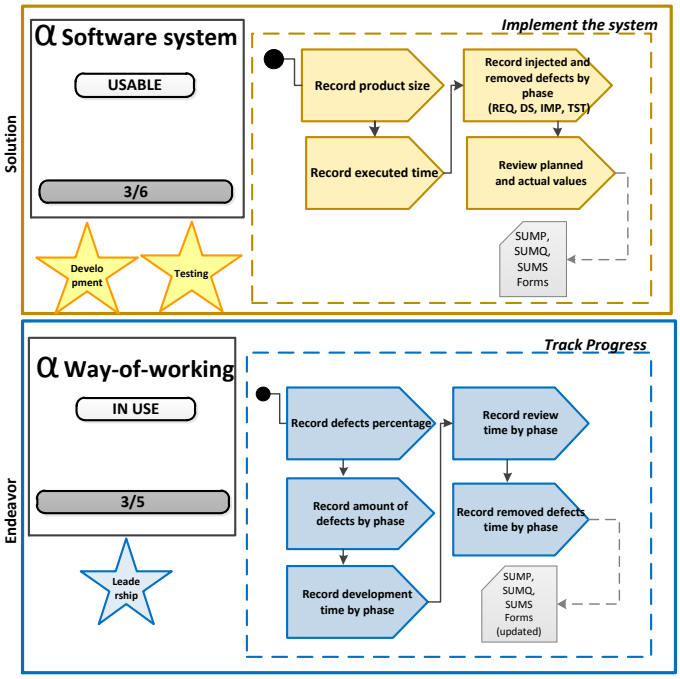


Figure 15. Monitoring Practices

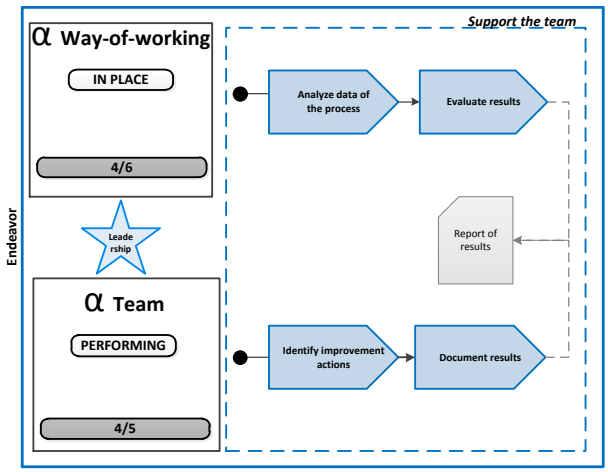


Figure 16. Postmortem Practices

The proposed representation can be used as a guideline and overview of the method for incorporating the SEMAT kernel components in software development teams including the TSP discipline.

The general framework and specific practices comprising the representation proposed are focused on the following elements of the SEMAT kernel: areas of concern, activity spaces, alphas, alpha states, work products, and competencies. For designing the general framework of the TSP representation in the SEMAT kernel, we chose the essentials phases of TSP and for each one we defined practices associated. Then, we represented them by using the elements of the SEMAT kernel (see Figure 11).

Future work includes the application of our representation in a development software team in educational context and the analysis of the improvement opportunities in the representation. When this representation is updated, we expect to apply the way-of working of SEMAT in a real project, taking into account the phases, practices, and works products of TSP.

REFERENCES

Berre, A. 2012. Essence – Kernel and Language for Software Engineering Methods. The joint submission for the OMG FACESEM standard "A Foundation for the Agile Creation and Enactment of Software Engineering Methods".

Carballo, L; McHale, J. & Nord, R. 2011. Using Architecture-Centric Engineering on a TSP Project.

Humphrey, W.S. 1999. *Introduction to the Team Software Process*. Massachusetts: Addison Wesley.

Huesca, A.M. 2010. Best Practices for TSP Implementation On Outsourced Application Development Projects. Keynote of SEPG-NA.

ITSQC Carnegie Mellon.2005.Comparing The eSCM – CL and CMMI V1.1.

Jacobson, I.; Meyer, B. & Soley, R. 2010. Software Engineering Method and Theory – A Vision Statement.

Jacobson, I.; Ng. P.; McMahon, P.E.; Spence, I. & Lidman, S. 2012. The Essence of Software Engineering: The SEMAT Kernel. *Communications of the ACM* 55(12): 42-49

McAndrews, D.R. 2000. The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices.

Object Management Group (OMG). 2012. Essence – Kernel and Language for Software Engineering Methods.

Software Engineering Institute (SEI). 2010. CMMI for Development, Versión 1.3.

Software Engineering Institute (SEI). 2011. A dedication to Excellence. In *Proceedings of the 6th Annual Software Engineering Institute (SEI) Team Software Process (TSP) Symposium*.

Software Engineering Institute (SEI). 2012. TSP Symposium 2012. St. Petersburg, FL.

Yung, H.; Bao, X. & Yang, S. 2009. Research and Improvement of Team Software Process. In *2009 World Congress of Computer Science and Information Engineering*.

Zapata, C.M.; Maturana, G.V. & Castro, L.F. 2013. Tutorial sobre la iniciativa Semat y el juego MetriCC. In *Proceedings of the 8CCC—Octavo Congreso Colombiano de Computación*.

This page intentionally left blank

Chapter #6

PSP Implementations for agile methods: a SEMAT-based approach

D. E. Brown

I.A. Hatchery, Los Angeles, CA

W. A. Arévalo Camacho & J. O. Muñoz Rengifo

Universidad Nacional de Colombia, Medellin

1. INTRODUCTION

The practices described in the Personal Software Process, PSP, show their usefulness in improving the quality of software products by improving the personal processes of developers (Humphry, 2002). The importance of having processes and methods that improve software product quality leads researchers to look for ways to adapt PSP practices and principles to existing development methodologies.

Shen *et al.* (2013) carried out a systematic state-of-the-art review in which they identified PSP adaptations for the software development methods SCRUM, RUP (Rational Unified Process), DSDM (Dynamic Systems Development Method), and XP (eXtreme Programming), as well as XP (eXtreme Programming). These adaptations demonstrate the uses of PSP practices combined with the practices used by each of these methods. Nevertheless, the number and form of the adaptations of these approaches for software development is different. These differences make the comparison of commonalities between the PSP adaptations—in the context of the aforementioned software development methods—difficult, and impede the use of PSP practices in software development methods as well as its re-utilization in other development methodologies.

In this Chapter we present a review of some PSP adaptations to software development methods and we propose a PSP representation in the SEMAT kernel. Based on the above, SCRUM-PSP and PXP (Personal Extreme Programming) are analyzed in Section 2, which adapt agile development methods such as SCRUM and XP. Then, based on that analysis, in Section 3 an adaptation is proposed that allows the re-utilization of PSP practices in different software development methods. In Section 4 we present an implementation of the proposed solution by using the SCRUM development method. Conclusions are discussed in Section 5. Be advised that the theoretical framework needed to understand the SEMAT-kernel-based representation is presented in the Preface of this book.

2. PRIOR ADAPTATIONS

2.1 SCRUM-PSP

This PSP adaptation of the SCRUM development method combines the administrative practices of an agile method with the individual practices of PSP, in order to improve the estimation and quality control capabilities of software products (Rong *et al.*, 2013).

Within each SCRUM-PSP iteration, the iterative cycle of SCRUM is utilized and elements of PSP are incorporated. The main element is the record of programmer efficiency data. In order to carry out the recording of this data, PSP templates are used, as described in the *introduction to PSP* (Rong *et al.*, 2013). This data is employed to improve the following SCRUM aspects:

- Estimation: the historic data obtained via PSP helps to determine if the reach of the project and the number of programmers is adequate to complete the project within the estimated time.
- Task planning: the programmers—by using PSP—are able to carry out better planning of activities that are to be performed during an iteration.
- Risk Reduction: the building a history of the factors that reduce programmer efficiency allows for actions that eliminate or reduce the effect of these factors.
- Reviews: the PSP reviews allow for the identification of software errors and defects produced by programmers. Also, by combining these with the SCRUM team practices, the developers learn about the errors committed by other team members.

2.2 PERSONAL EXTREME PROGRAMMING—PXP

PXP is presented as a software development methodology that, though based on XP, does not include all XP practices. On the other side, PXP utilizes some PSP practices but also does not use all these PSP practices (Dzhurov *et al.*, 2009).

The XP practices used by PXP are:

- Continuous Integration.

- Simple design.
- Short version release cycles.
- Refactoring.
- Test oriented development.
- Spike Solutions.

The practices taken from PSP are:

- Time recording.
- Defect history.
- Defect type standardization.
- Size Estimation.
- Process improvement proposals.
- Code reviews.

"Coding Conventions" are an additional PXP element. These standards are agreed upon by team members so that everyone uses the same techniques in code generation tasks.

PXP employs PSP practices in order to build a history of programmer efficiency and, with that information, obtain a better estimation of the project and to be better able to identify what needs to be improved gaining efficiency in the software development process. Nevertheless, it does not use the PSP templates for that information history.

PXP uses the Microsoft Visual Studio Team System (VSTS) to perform the information recording. It uses Unit tests done on the different software components in order to measure software product quality. The more unit tests done on a component the better its quality will be.

3. PSP REPRESENTATION IN THE SEMAT KERNEL

In the state-of-the-art review, evidence is encountered that the need exists to implement, in the interior of the development methods, practices like those proposed by PSP. One example of this evidence is encountered in Flacid SCRUM (Williams, 2012). It has been said that a development team employs Flacid SCRUM when they understand the SCRUM principles and how to apply them, however low quality code is being developed.

The above brings project software delays with it, due to the need to continually rewriting code that is not well designed. Adaptations, such as SCRUM-PSP and PXP employ PSP practices in order to help programmers to generate high quality code and, in addition to this, be able to do more precise estimations. Nevertheless, differences exist between these adaptations: SCRUM-PSP employs PSP elements as they are described in the Introduction to PSP while PXP only uses those PSP elements that the authors feel are necessary to obtain better software product quality (Dzhurov *et al*, 2009).

Additionally, differences exist in the ways in which software quality is measured in these methods: SCRUM-PSP measures quality based on the number of lines of code and PXP measures quality based on the number of unit tests.

Also, the language employed in these two methodologies is different: for SCRUM-PSP the changes to processes are realized within the "risk" element while in PXP, these changes are done within the "proposal for process improvement" element. This is also the case for recording history data: in SCRUM-PSP the history is produced in the "software development process," while in PXP the history is divided among "time history" and "defect history."

In order that the PSP practices can be reutilized these need to be expressed in a common language that permits their application to any software development methodology. SEMAT presents a common language by which any software practice can be expressed, regardless of the software development method in which it is utilized. This characteristic permits the representation of the software practices in the SEMAT kernel that is presented below.

PSP is shown in Figure 1 as an alpha *way of working*. The seven levels of PSP are represented as states of a new sub-alpha named *PSP Compliance*. The representation of the levels of PSP as states of the new sub-alpha allows for the development team to know the state in which PSP is encountered. For example, the development team can present the following characteristics:

- An ordered process for carrying out tasks.
- A process that allows for a basic time measurement that these tasks take and the defects introduced in the development of these tasks.
- Coding conventions inside the team.

For the case presented in Figure 1, the development team complies with the activities that are proposed by PSP0 and is currently realizing one of the activities proposed by PSP0.1

4. PSP AND SCRUM IN THE SEMAT KERNEL

The representation of PSP in the SEMAT kernel describes the activities that need to be carried out by the development team in abstract form in each of the states of the sub-alpha *PSP Compliance*. Nevertheless, as the PXP developers demonstrated, the activities of PSP can be developed without the necessity of employing templates described in the Introduction to PSP (Rong *et al*., 2010).

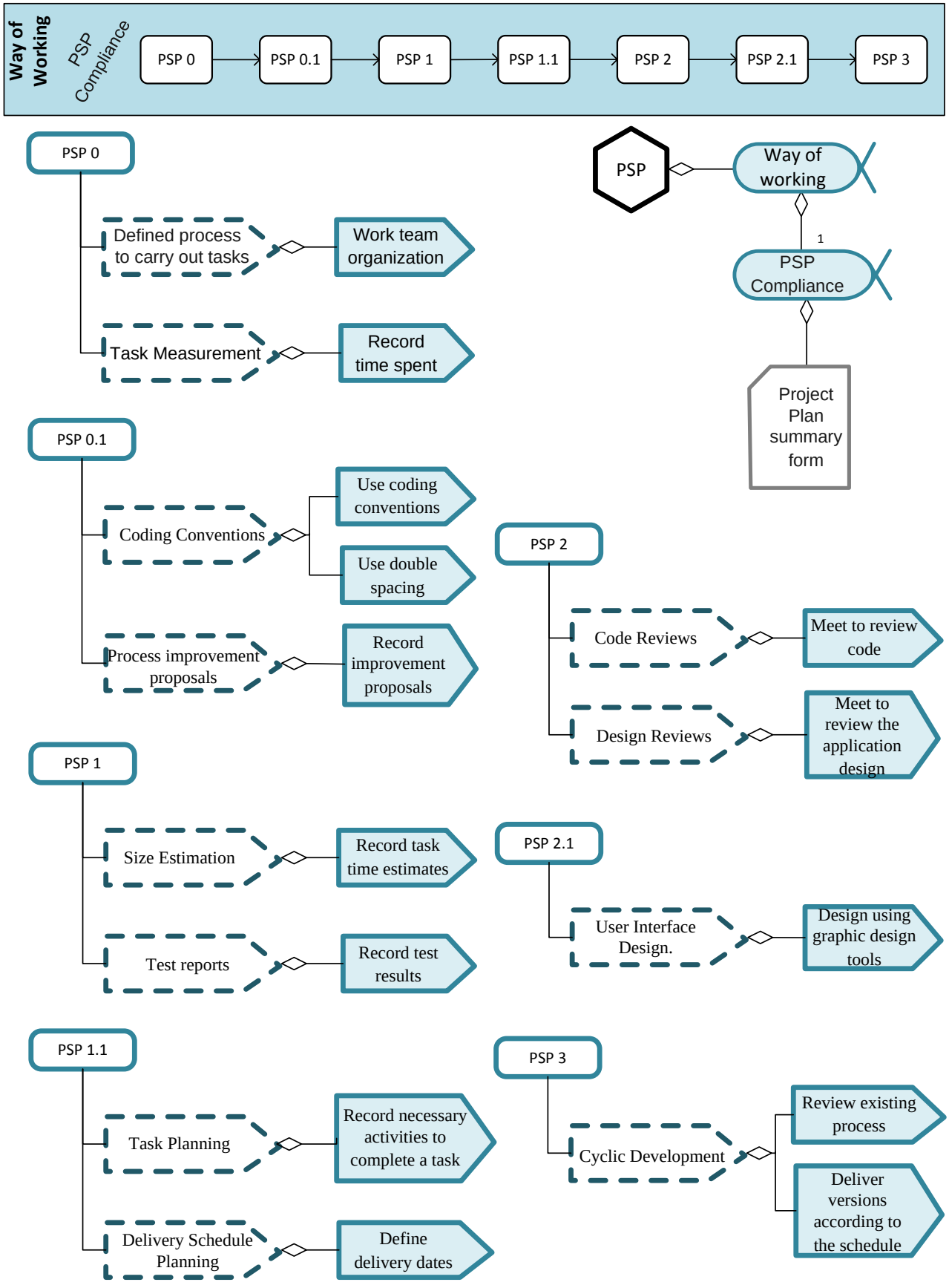


Figure 1. PSP Representation in the SEMAT Kernel

The history of measurements, the estimations and the code revisions, the schedules and the improvement processes can be carried out via software applications designed to support these activities. Below, the use of the PSP practices applied under the SCRUM framework will be shown. In order to carry out the measurement, planning, scheduling and improvement processes, the Jira administration software will be employed (Codina Navarro, 2010).

The use of Git to document design reviews may be problematic as it is a structure without any predefined format and many of the comments may relate to other issues than just design review. A Git Pull can be to begin a new branch or a new feature not related to existing design. This could get lost in the plethora of changes tracked by Git. However, we used Github, a cloud implantation of GIT version control (Dabbish, et all, 2012), in order to support code review activities.

In Figure 2, a SCRUM in SEMAT approach is presented without including the PSP elements.

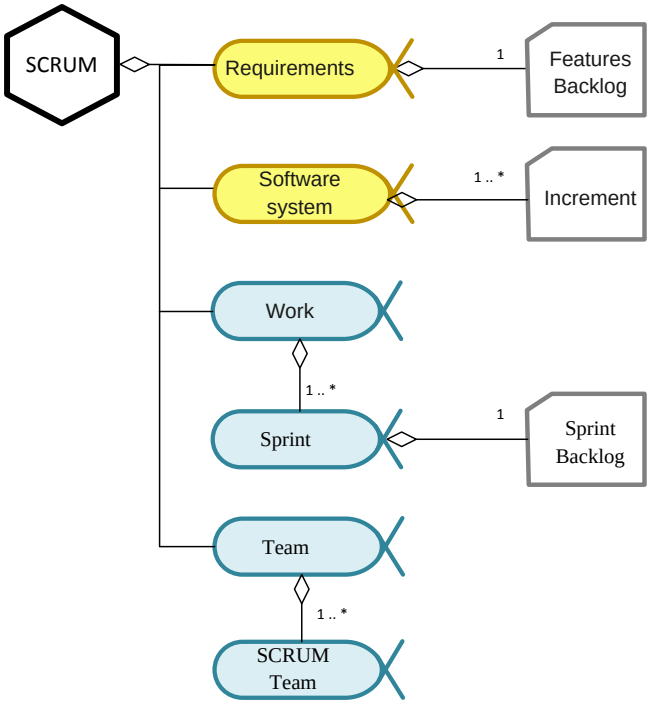


Figure 2. SCRUM Representation without PSP elements

In Figure 3, the SCRUM respresentation is presented employing SEMAT elements. When this last Figure is compared with the PSP representation in Figure 1, it can be seen that the activities described in the sub-alpha states of *PSP compliance* are concrete activities that are carried out in Jira or in Github. Thus, having an abstract representation of PSP practices in SEMAT, we can achieve a concrete representation of those practices in a software development method. The above shows the flexibility of the PSP representation in SEMAT and its capacity to be reusable with different software development methodologies.

5. CONCLUSIONS

The PSP practices are useful in reducing defects in software products. Adaptations of these practices exist for diverse agile development methods, but this diversity makes the comparison of the form in which these methods employ PSP practices difficult.

Our representation employs the SEMAT kernel to facilitate the realization of PSP practices in different software development methods. The representation defines, within the alpha workflow, a new sub-alpha, *PSP compliance*. The PSP level of the team can be identified by comparing the practices and activities that are being utilized, and with the activities described in each of the *PSP compliant* sub-alpha states.

This solution was adapted to the SCRUM development method where its adaptability can be appreciated. This was achieved due to the fact that the representation defines the activities that need to be carried out by the team to achieve the PSP levels instead of needing to define tools that should to be used to obtain said levels.

In the example, two tools are employed: Jira for the estimation of tasks and the recording of the time spent in these tasks, and Github, to carry out code reviews by the team members. This flexibility allows for different tools to be utilized for achieving the objectives proposed by PSP and, at the same time, facilitates the adaptation of PSP practices to different software development methodologies.

6. REFERENCES

Humphrey, W. 2002. Three process perspectives: organizations, teams, and people. *Annals of Software Engineering* 14(1–4): 39–72.

Shen, M, Rong, G & Shao D. 2013. Integrating PSP with agile process:a systematic review. In *Proceedings of the 2nd International Conference On Systems Engineering and Modeling*, 805–811.

Rong, G, Shao, D & Zhang, H. 2010. SCRUM-PSP: Embracing Process Agility and Discipline. In *Proceedings of the 2010 Asia Pacific Software Engineering Conference*, 316–325.

Dzhurov, Y, Krasteva, I & Ilieva, S. 2009. Personal Extreme Programming–An Agile Process for Autonomous Developers. In *Proceedings of the International Conference on Software, Services & Sematic Technologies*.

Williams, L. 2012. What agile teams think of agile principles. *Communications of the ACM* 55(4): 71–76.

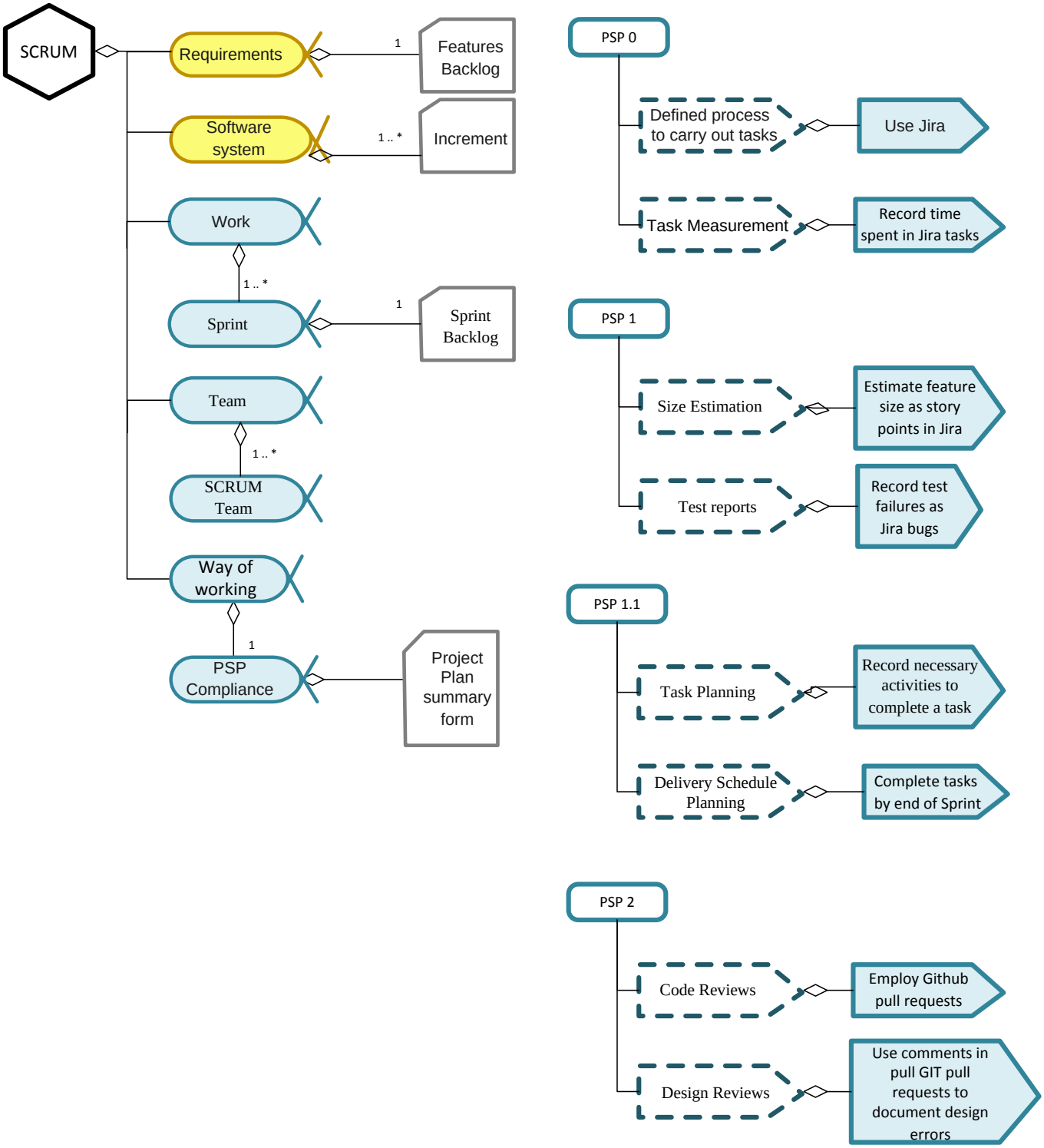


Figure 3. Scrum-PSP Representation with SEMAT elements

Codina-Navarro, F. 2010. *Desarrollo de una aplicación para la gestión de calidad de los procesos en el entorno JIRA*. Universitat Politècnica de València.

Dabbish, L, Stuart, C, Tsay, J & Herbsleb, J. 2012. Social coding in GitHub. In *Proceeedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW*, 1277–1286.

This page intentionally left blank

Chapter #7

Toward a standardized representation of RUP best practices of project management in the SEMAT kernel

María Eugenia González-Pérez
Institución Universitaria Salazar y Herrera, Medellín, Colombia
Carlos Mario Zapata-Jaramillo
Universidad Nacional de Colombia- sede Medellín, Medellín, Colombia
Liliana González-Palacio
Universidad de Medellín, Medellín, Colombia

1 INTRODUCTION

SEMAT is an initiative for redefining software engineering by means of the establishment of a widely accepted kernel of elements and a solid theoretical foundation of proven principles and best practices from industry, academia, researchers, and users (OMG, 2012). RUP (Rational Unified Process) has some project management practices related to PMBOK (Project Management Body of Knowledge). Project management is recognized as one of the most critical areas of the software development process and it can be represented by using the SEMAT kernel.

SPEM (Software and Systems Process Engineering Metamodel) is another proposal for representing methods and practices, but it exhibits greater complexity in its use compared with the SEMAT kernel. Also, the SEMAT kernel provides support for monitoring and tracking the project process by using the "things that we always work with"—called alphas—and the "things we always do"—called activity spaces.

In this Chapter, we propose the SEMAT-kernel-based representation of one of the best practices of RUP called *developing software iteratively* by using PMBOK as the main reference for project management. This is a contribution to the standardization of methods, methodologies, and processes most recognized in the software development within a common framework to take from every initiative useful elements according to the need of each project.

To the extent of meeting this goal, in Section 2 the main concepts associated with the proposal are defined (Project Management, PMBOK, and RUP, since the SEMAT kernel elements are presented in the Preface of this proposal). In section 3 we set out other attempts to solve the challenge of standardization, stating their strengths and weaknesses. In Section 4 we show the representation of the practice *developing software iteratively* in the Semat kernel. Section 5 contains the conclusions and future work that will continue the standardization of best practices from different methods.

2 CONCEPTUALIZATION

During the software development process, both management and technical activities are necessarily carried out (Contreras *et al.*, 2011). Seeking to provide guidelines for the implementation of the tasks associated with the project management and administration, a set of proposals, methodologies, and methods have emerged.

2.1 *Project Management and PMBOK*

The PMBOK® Guide identifies a set of good practices, knowledge, skills, tools, and techniques related to project management for enhancing the chances of success on several projects (IEEE Std 1490-2003, 2004; PMI, 2013). The PMBOK® Guide also defines a common vocabulary, like an essential element of a professional discipline (PMI, 2013). For this purpose, this guide has a set of knowledge areas (see figure 1).

Figure 1. PMBOK Knowledge Areas. Source: Adapted from (PMI, 2013)

PMBOK is an essential reference when someone is looking for the best practices related to project management. However, best practices specifically related to the software development process can be found in other proposals, such as the Rational Unified Process (RUP).

2.2 *RUP*

IBM Rational Unified Process® (RUP®) is a comprehensive process framework including industry-tested practices for software and systems delivery and implementation. Also, effective project management is included in this framework (Rational, 1998). RUP comprises four phases and nine disciplines (see figure 2). Project Management is recognized as one of the cross-cutting, most critical disciplines of the software development.

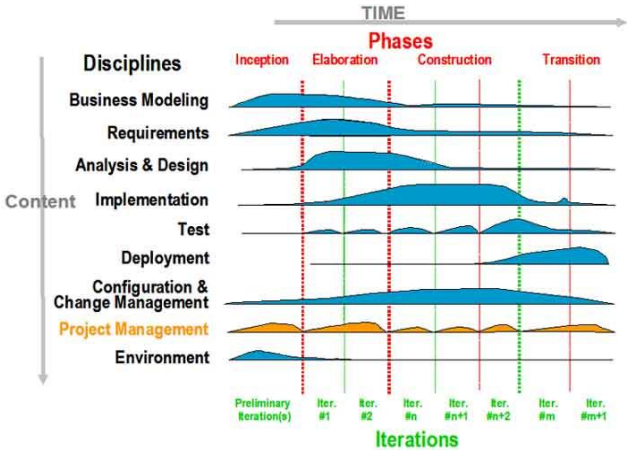


Figure 2. RUP Structure. Source: Rational (1998)

3 RELATED WORK

Previous to the SEMAT kernel, the Software and Systems Process Engineering Meta-model (SPEM) was the other approach for providing a common framework to represent practices, methods, and software development models. According to OMG (2008), SPEM "is a process engineering meta-model as well as conceptual framework, which can provide the necessary concepts for modeling, documenting, presenting, managing, interchanging, and enacting development methods and processes. An implementation of this meta-model would be targeted at process engineers, project leads, project and program managers who are responsible for maintaining and implementing processes for their development organizations or individual projects."

SPEM adds additional complexities to the method and practice representation. Consequently, among the software industry and the software professionals SPEM recognition and adoption is still far away. SPEM Specification has focused on organizations with a separate group of people in charge of maintaining the processes. Specifically, SPEM target audience has been focused on process engineers, project leaders, and project and program managers who are responsible for maintaining and implementing processes (OMG, 2008).

Compared with SPEM, the SEMAT kernel exhibit some similarities in the authoring capabilities provided by the two specifications, but key differences arise in the method architecture with respect to support for enactment (Elvesæter *et al.*, 2013). SPEM exhibits some other disadvantages like the vague semantics (Schuppenies & Steinhauer, 2004; Shengjun *et al.*, 2007). SPEM major components are included in Figure 3.

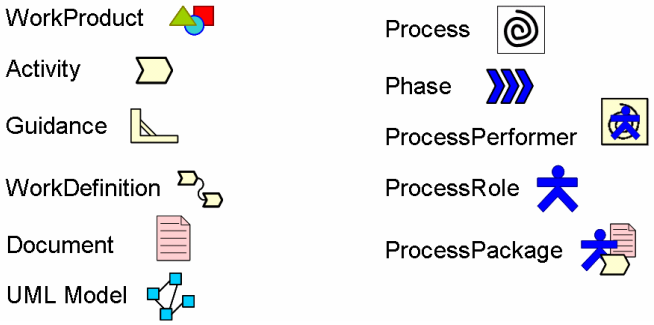


Figure 3. The most important SPEM stereotypes. Source: Schuppenies & Steinhauer (2004)

According to Elvesæter *et al.* (2013), three main differentiators between the SEMAT kernel and SPEM are: underpinning values, support for enactment, and ease of learning and use. While SPEM is more concerned with the processes and engineers who carry out these processes, the SEMAT kernel includes all of the events and actors involved in the software development life cycle by using items such as alphas, activity spaces, competencies, etc.

4 PROPOSAL

The process to be followed for representing the RUP practices related to project management in the SEMAT kernel is graphically summarized in Figure 4. The steps of the process are:

- Step 1: we identify the six best practices of RUP (Rational, 1998).
- Step 2: since we are interested in management practices, a correlation to PMBOK should be established.
- Step 3: we also need to correlate the practices to the SEMAT kernel area of concern *endeavor*.
- Steps 4 and 5: this analysis provides the necessary elements to develop a table showing the consolidated activities of the project management discipline in RUP and their work products, which are paired with the SEMAT kernel alphas. A graphical representation of alphas and work products is then provided.

- Step 6: Finally, the relationship between the activities of RUP and the activity spaces in SEMAT is determined, and the corresponding graphical representation is made.

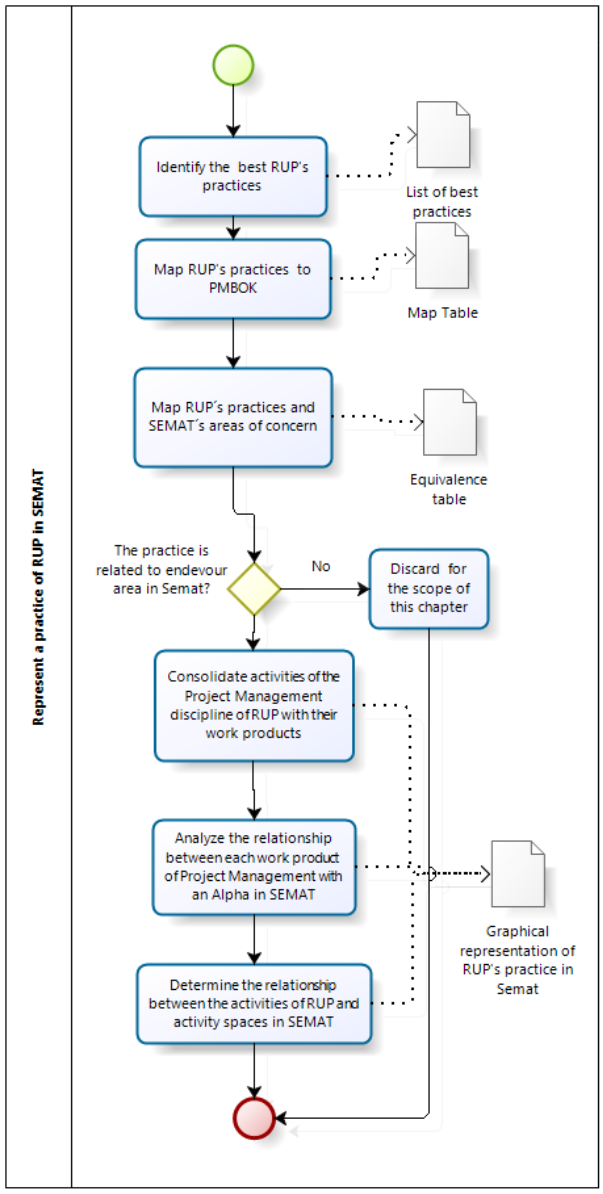


Figure 4. Process diagram for the SEMAT-kernel-based representation. Source: The authors

The remainder of this Section shows in more detail the most important steps in the process.

4.1 *Identification the best RUP's practices*

RUP authors determined the following six best development practices (Rational 1998): develop software iteratively, manage requirements, use component-based architectures, visually model software, and control changes to software.

4.2 *Mapping RUP best practices to the PMBOK*

Only the RUP best practice *control changes to software* is related to the PMBOK knowledge area *quality management*. The five remaining can be related to *integration management*.

4.3 *Mapping RUP best practices and SEMAT kernel areas of concern*

According to the description of each of the six best practices, we have determined what SEMAT area of concern can be assigned to the respective practice. In Figure 5, the comparative analysis carried out between the definitions of each of the six practices and the definitions of each area of interest, allowed for us to identify the relationship between two of the best practices in the area of concern *endeavor* (blue) and the remaining four in the area of concern *solution* (yellow) as follows:

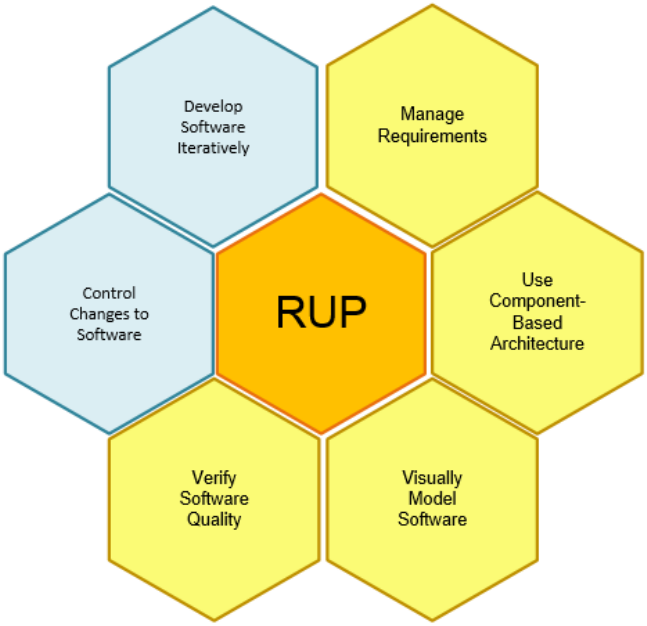


Figure 5. Six RUP best practices in the SEMAT kernel notation. Source: The authors

- Area of Interest *endeavor*:
  - ✓ Develop software iteratively.
  - ✓ Control changes to software.
- Area of Interest *solution*:
  - ✓ Manage requirements.
  - ✓ Use component-based architecture.
  - ✓ Visually model software.
  - ✓ Verify software Quality.

We selected the practice *develop software iteratively* to be represented in this Chapter.

4.4 *Consolidation activities of the RUP project management discipline with their work products. Correlation with the SEMAT-kernel alphas*

According to Rational (1998), a process describes *who* is doing *what*, *how*, and *when*. RUP is represented by using four primary modeling elements: workers ('who'), activities ('how'), artifacts ('what'), and workflows ('when'). The main RUP activities related to the discipline *develop software iteratively* are depicted in Figure 6. Also, the artifacts associated with the RUP activities and their descriptions are included in Table 1.

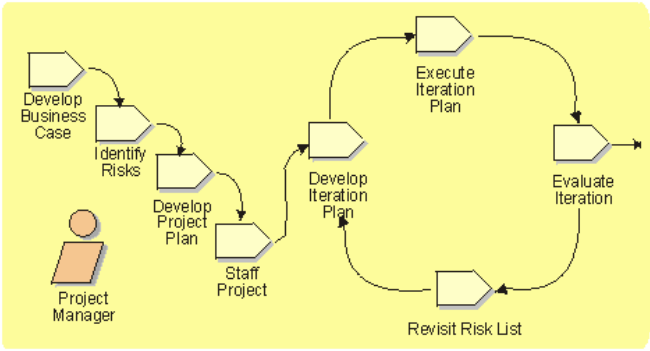


Figure 6. RUP activity overview of the discipline *develop software iteratively*. Source: Rational (1998)

We identified a close relationship of the practice *develop software iteratively* and the three alphas belonging to the area of concern *endeavor*, as shown in Figure 7.

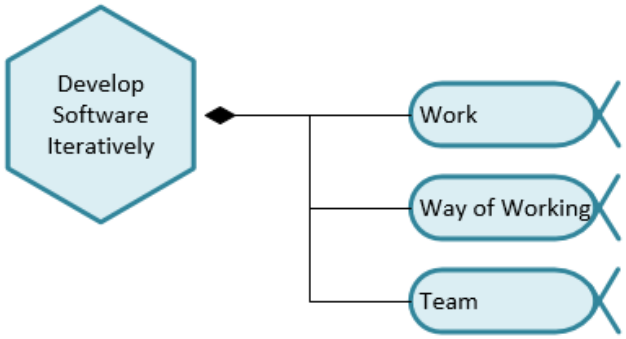


Figure 7. Relationship between the RUP practice *develop software iteratively* and the area of concern *endeavor*. Source: The authors

The RUP documentation provides the following artifacts related to the practice *develop software iteratively:* business case, risk list, implementation model, iteration assessment, project plan, measurement plan, updated project plan, and iteration plan.

By analyzing the description of each artifact derived from the RUP best practice and comparing it with the description of each alpha, we discovered the relationship of *develop software iteratively* with other SEMAT alphas located in the three SEMAT areas of concern. Also, we can map the RUP artifacts to SEMAT work product, as shown below.

Table 1. Artifacts associated with the RUP activities. Source: Rational (1998)

Activity	Artifact	Artifact description
Develop Business Case	Business Case	Provides the necessary information from a business standpoint, to determine whether or not this project is worth investing in
Identify Risks	Risk list	A sorted list of known, open risks to the project, sorted in decreasing order of importance, associated with specific mitigation or contingency actions
Develop Project Plan	Project Plan	Defines the overall schedule for the project over time: dates for the phases and the associated major milestones, and dates for the iterations with their major objective.
	Measurement Plan	Defines the measurement goals, the associated metrics, and the primitive metrics to be collected in the project to monitor its progress
Staff Project	Updated Project Plan	Redefines the overall schedule for the project over time: dates for the phases and the associated major milestones, and dates for the iterations with their major objective.
Develop Iteration Plan	Iteration Plan	A time-sequence set of activities and task, assigned to resources, containing task dependencies, for the iteration; a fine-grained plan.
Execute Iteration Plan	Implementation Model	Collection of components, and the implementation subsystems that contain them. Components include both deliverable components, such as executables, and components from which the deliverables are produced, such as source code files.
Revisit Risk List	Risk List	A sorted list of known, open risks to the project, sorted in decreasing order of importance, associated with specific mitigation or contingency actions.
Evaluate the Iteration	Iteration Assessment	The iteration assessment captures the result of an iteration, the degree to which the evaluation criteria were met, and lessons learned and changes to be done.

Opportunity: the set of circumstances that makes it appropriate to develop or change a software system (Jacobson *et al.*, 2013). We assigned here the business case and the risk list, as shown in Figure 8.

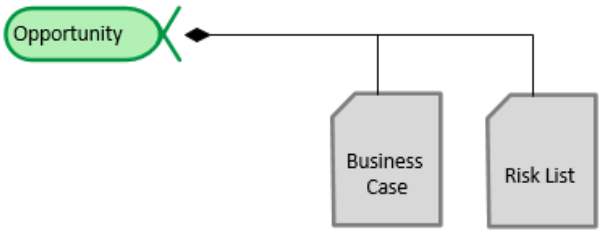


Figure 8. Work products of the alpha *opportunity*. Source: The authors

Software System: a system made up of software, hardware, and data that provide its primary value by the execution

of the software (OMG, 2012). We assigned here the implementation model and the iteration assessment, as shown in Figure 9.

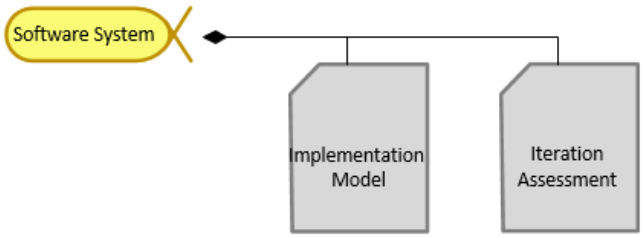


Figure 9. Work products of the alpha *software system*. Source: The authors

Way-of-Working: the tailored set of practices and tools used by a team to guide and support their work (OMG, 2012). We assigned here the project plan, the measurement plan, the updated project plan, the iteration plan, and the iteration assessment, as shown in Figure 10.

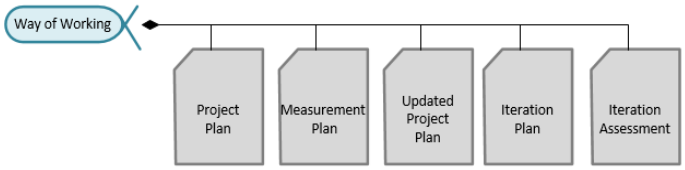


Figure 10. Work products of the alpha *way of working*. Source: the authors

4.5 Determination of relationship between the RUP activities and the SEMAT activity spaces

Once you have fully identified the work products associated with each alpha, we could proceed to identify the SEMAT activity spaces in which each RUP activity is associated within the discipline *develop software iteratively*, as shown below.

Explore Possibilities: explore the possibilities presented by the creation of a new or improved software system. This includes the analysis of the opportunity to be addressed and the identification of the stakeholders (OMG, 2012). We assigned here the activities *develop business case*, *identify risks*, and *revisit risk list*, as shown in Figure 11.

Implement the System: Build a system by implementing, testing and integrating one or more system elements. This includes bug fixing and unit testing (OMG, 2012). We assigned here the activity *execute iteration plan*, as shown in Figure 12.

Test the System: Verify that the system produced meets the stakeholders' requirements (OMG, 2012). We assigned here the activity *evaluate the iteration*, as shown in Figure 13.

Coordinate Activity: co-ordinate and direct the team work. This includes all on-going planning and re-planning of the work, and re-shaping of the team (OMG, 2012). We assigned here the activities *develop project plan*, *staff project*, and *develop iteration plan*, as shown in Figure 14.

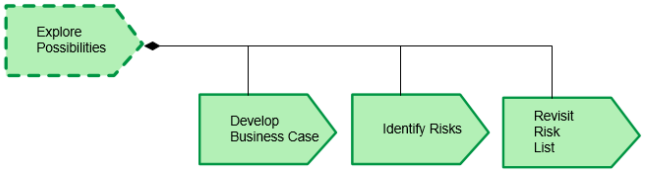


Figure 11. Relationship between the activity space *explore possibilities* and the activities of the discipline *develop software iteratively*. Source: the authors

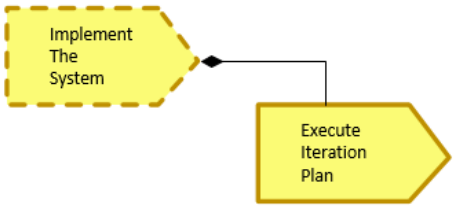


Figure 12. Relationship between the activity space *implement the system* and the activities of the discipline *develop software iteratively*. Source: the authors

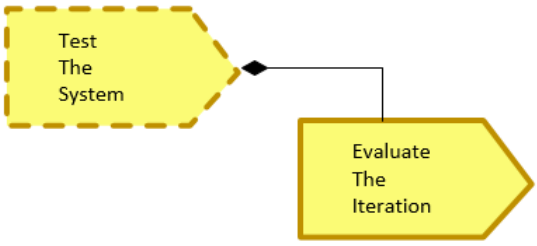


Figure 13. Relationship between the activity space *test the system* and the activities of the discipline *develop software iteratively*. Source: the authors

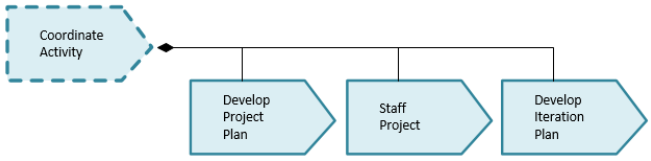


Figure 14. Relationship between the activity space *coordinate* and the activities of the discipline *develop software iteratively*. Source: the authors

5 CONCLUSIONS AND FUTURE WORK

In this chapter we generated a SEMAT-kernel-based representation of one of the RUP best practices—named *develop software iteratively*—for managing a software project. In this way we wanted to contribute to the standardization of all the models and methods used in software engineering by using a common language.

The practice *develop software iteratively* only relates to one alpha of the area of concern *endeavor*, while the activities and work products are connected to the three areas of interest. This fact is not counterintuitive, since the project management discipline is cross-cutting to the whole process and the selected practice is related to the realization of iterations.

Some future work can be proposed in order to continue the proposed representation of this Chapter. For example, considering the fact that none of the six RUP best practices is related to each other, at least not explicitly, we can propose carrying out the integration of the remaining five RUP best practices into the SEMAT kernel. Also, we can combine them with the best management practices belonging to a different software development method—*e.g.*, SCRUM. In this way we could ensure the focus of software development for projects is not centralized into processes, considering the importance of the people who make the work.

We also intend to promote, by including the SCRUM best management practices, the importance of considering previous practices when trying to adopt a new method. In the case of SCRUM, some previous RUP practices could support the new activities promoted by SCRUM, even though the companies are moving toward new fashions like the agile methodologies.

A final future work to be proposed is the addition of several other elements of the SEMAT kernel for representing additional information of the RUP practices. For example, the patterns can be used for representing the roles and phases of the RUP method, and competencies can be considered for adding information to the RUP activities.

6 REFERENCES

Contreras, M., Villamizar, L., & Orjuela, A. 2011. Modelo de integración de las actividades de gestión de la guía del PMBOK, con las actividades de ingeniería, en proyectos de desarrollo de software. *Revista Avances en Sistemas e Informática* 8(2): 97-106.

Elvesæter, B., Benguria, G., & Ilieva, S. 2013. A Comparison of the Essence 1.0 and SPEM 2.0 Specifications for Software Engineering Methods. PMDE.

IEEE Std 1490-2003. 2004. IEEE Guide Adoption of PMI Standard a Guide to the Project Management Body of Knowledge IEEE Std 1490-2003 (Revision of IEEE Std 1490-1998):_1-216.

Jacobson, I., Pan-Wei, N., McMahon, P., Spence, I., & Lidman, S. 2013. *The Essence of Software Engineering—Applying the SEMAT Kernel*. London: Addison-Wesley.

Johnson, P., Ekstedt, M., & Jacobson, I. 2012. Where's the Theory for Software Engineering? *IEEE Software* 29(5): 96-96.

Object Management Group (OMG). 2008. Software Process Engineering Metamodel (SPEM). Available: http://www.omg.org/spec/SPEM/2.0/

Object Management Group (OMG). 2012. Essence – Kernel and Language for Software Engineering Methods (Vol. ad/2012-11-01): 283).

Project Management Institute (PMI). 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Fifth Edition*. Newtown Square: Project Management Institute.

Rational. 1998. *Rational Unified Process: Best Practices for Software Development Teams*. (TP026B), 21.

Schuppenies, R., & Steinhauer, S. 2004. *Software Process Engineering Metamodel*. Hasso-Plattner-Institut.

Shengjun, W., Longfei, J., & Chengzhi, J. 2006. Represent Software Process Engineering Metamodel in description logic. *In Proceedings of the World Academy of Science, Engineering and Technology*, Vol. 11.

Chapter #8

RSM as SEMAT Kernel Extension for Reliability

M. J. Simonette & E. Spina
Universida de de São Paulo, São Paulo, Brasil

1 INTRODUCTION

Software development teams use a set of practices and tools to guide and support their work. These practices and tools help integration of team members and support the understanding of what should be implemented.

Team management should be done with focus on the delivery of software that achieves both the customer needs and expectations, considering the functional and non-functional requirements of customer demands. Furthermore, team management should be aware that software errors may occur due to software developers.

On whatever occasion that a software developer makes a mistake in their work, faults are injected into the software application. Developer errors can occur at any stage of the software development life cycle, and can be classified in two broad categories (Stutzke & Smidts, 2001):

- Errors made during the activities of analysis, design, and coding.
- Errors made during attempts to remove faults identified during the activities of verification and validation.

Software development process is a human-labor-intensive activity, and software developers suffer pressure to be efficient in building the right software without faults. The large amount of distinct elements in software systems makes the development of such systems more complex than any other type of human construction (Weinberg, 2011; Poppendieck & Poppendieck, 2003; Brooks, 1997).

Humans are essential to the software development process although the repertory of models and techniques for avoiding faults being inserted by developers is very limited. There are several software reliability models (Musa, 2004; Rekab *et al*., 2013; Amin *et al*., 2013; Okamura *et al*., 2013); nevertheless, the number of models and techniques to deal with software developer errors are very limited, and do not reflect the fact that a single developer error can inject multiple faults into a software system, as software systems are non-linear systems (Stutzke & Smidts, 2001; Xion & Li, 2013).

Team managers need to identify the factors of the development environment that have influence in developer errors, and to consider the different parts of the software system to be developed. Some of these factors are (Stutzke & Smidts, 2001; Rasmussen & Vicente, 1989):

- Lack of resources and tools for the team, which can be caused by insufficient knowledge, or lack of consideration of proper preconditions or side effects of a decision.
- Team ability.
- Time pressure under which team are required to work.
- Familiarity of the team with the type of system and business rules.

Soft system engineering approach considers people as system elements with a holistic view. Such an approach can help team managers to deal with the factors of development environment that influence the developer error because it considers the people working, their individual interests, objectives, attitudes, and mutual interactions (Hitchins, 2008; Checkland, 1999).

In this Chapter we consider the Hitchins' Rigorous Soft Method (RSM), a soft system engineering approach that allows for the knowledge construction about the development environment. This knowledge is necessary for identifying the factors that facilitate the occurrence of developer errors in the development environments.

The Chapter is organized as follow: in Section 2 we describe RSM. In Section 3 we discuss the SEMAT[1] kernel extension to describe the RSM practices in kernel language, in a way that team managers can apply the RSM in software de-

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

velopment to deal with developer error; the last section concludes the chapter.

2 SOFT SYSTEMS ENGINEERING

Soft systems engineering uses *systems thinking* to understand the nature of a problem, seeking practical experiences and interactions with the problem. It proposes solutions that may not solve the problem, although it can provide solutions for improving the problem understanding, and the development of a solution, which is the best at the moment. According to Senge (2006), systems thinking is a discipline to see a system in its totality, a kind of framework to view the interrelationship of system components more than to view the components, to view the system patterns of change more than to view system static images. Hitchins (2008) argues that systems thinking caught the attention of engineers when they realized that the Cartesian approach—the most successful technique used by engineering—have difficulties to deal with systems that include people.

Software development is a human activity, and the software development environment is a system in which the human dimension has a complexity which demands team managers to adapt their practices according to behavior of team members and environment changes. Soft systems engineering methods can help team manager in this task, as these methods deal with a variety of elements, which can be analyzed. Besides, these methods are designed and organized in a way that is more qualitative than quantitative, and they transform the information obtained to enable managers to deal with the factors of the development environment that have influence in developer errors.

2.1 *Soft system engineering methods*

Hitchins' RSM is one of the methods used by soft system engineering. It is based on the General-Purpose Problem Solving paradigm. Derek Hitchins states that RSM is designed to address complex problems and issues, and to support the conception of potentials solutions (Hitchins 2008).

Checkland's soft systems methodology (SSM) is another software system engineering method (Checkland 1999). It promotes the agreement of the multiple problem views and multiple interests of the people that are involved in a problematic situation. Besides, it addresses complex issues and problems related to the presence of people in unstructured—problematic—situations.

SSM and RSM are context free. Both methods provide support to the knowledge construction about the problem domain. They bring information about the issues or problem into the method, which can generate large amounts of data and information. Unlike SSM, RSM is rigorous, as it employs defined tools and processing methods to handle, organize, and process information. Hitchins (2008) points out that is the "process methods" of RSM that allows for the transformation of disordered source data into specific solution information.

2.2 *Hitchins' Rigorous Soft Method (RSM)*

Rigorous Soft Method addresses problems using hierarchies of "symptoms" caused by the problem. As a system method, it addresses whole systems at once, rather than work with particular aspects of the problem. RSM is suitable for team-based working, and generates requirements for problem resolution.

RSM has a seven-step process. Each step invokes the usage of particular techniques that are chosen so that the output from one technique forms is required as input by the following techniques. This fact moves the process forward. The seven steps are (Hitchins, 2008):

- Nominate issue and Issue domain—in which the problem issues are identified and a description of the situation is made.
- Identify issue symptoms and factors—that identifies the symptoms of the problem, and the factors that make them significant to be explored.
- Generate implicit systems—each symptom implies the existence of at least one implicit system in the problem situation.
- Group into containing system—at this step, the implicit systems are aggregated to form clusters, one cluster for each symptom, called containing system, which can generate a hierarchy of systems, highlighting issues related to the problem.
- Understanding containing systems, interactions, and imbalances—at this step, the interactions between the containing systems are evaluated.
- Propose containing systems imbalance resolution—this step uses the differences between an ideal world, where the symptoms do not exist, and the real world, to propose socio-technical solutions to the imbalances identified in the previous step.
- Verify proposal against original symptoms—at this step, the system model are tested to see if they eliminate the symptoms identified at step two and the imbalance found at step six.

3 EXTENDING SEMAT KERNEL TO DEAL WITH DEVELOPER ERRORS

Albeit the *things we always work with* and the *things we always do* of the SEMAT kernel are a small set of essential things that are universal in software development environments, they are not concerned with the software developer errors. The absence of developer errors is not a problem of the kernel, because the kernel is defined as the essential elements of software engineering, not as "all elements" a development team need to deal with.

Developer errors bring risks to the software development endeavor. Development team managers need to identify the factors of the development environment that have influence in developer errors. RSM is a method that can help team managers to have a development environment resilient to developer errors. According to Hitchins (2008) RSM has tools and defined processing methods. These features are not clear or defined in other soft system engineering methods, which are dependent on the skills and insight of those who use the methods to find the right tools and processes to be used.

To make use of RSM, team managers need some guidelines usable by inexperienced team members, and also promoting a common understanding among team members of how to conduct the RSM in software development process. The SEMAT kernel can be scaled to address this challenge by providing guidance beyond the essential elements provided by the kernel; this guidance comes in form of what SEMAT refers to as practices.

3.1 *RSM Practices*

Developer errors are generated both in area of concern *endeavor* of the SEMAT kernel, and in the relationship among the endeavor alphas and the alphas the other two areas of concern. Figure 1 shows all the relationships of the alphas belonging to *endeavor*:

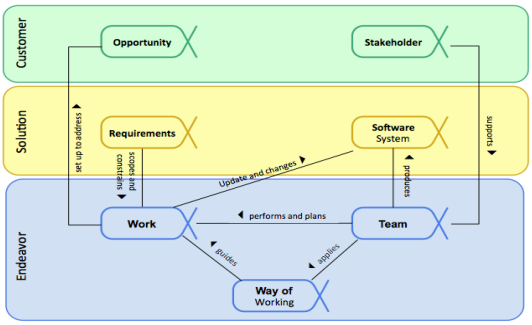


Figure 1. Alphas belonging to *endeavor* and their relationship with other alphas.

- Work has relationship with requirements, opportunity and software system.
- Team has relationship with software system and stakeholders.
- Way of working lacks relationship with any other alphas than the *endeavor* ones. However, way of working is an essential link between team and work, and it is under direct influence of the factors mentioned by Clarke & O'Connor (2012).

RSM practice can be applied to the area of concern *endeavor* in terms of "thing to work with" (Figure 2) and "things to do" (Figure 3). Considering the "things to work with," RSM practice provides guidance to clarify the environment factors to team members and manager. This practice provides guidance on how to conduct RSM activities, namely:

- Nominate the potential development environment factors that may have influence in developer errors. This is made by executing the first four steps of RSM.
- Understand the interactions between the environment factors that were nominated in first four steps by executing the steps five and six of RSM. It includes the identification of the core activities required for developing a resilient environment by means of the comparison of an ideal model of environment with the team perception of the real endeavor environment. These activities are grouped in Support Action Plan.
- The Support Action Plan developed at the end of RSM step six is implemented by executing the step seven of RSM.
- Observe the development environment as a way to identify the necessity of a new cycle of the RSM to control or to improve some action in the environment for dealing with the factors that may have influence in developer errors.

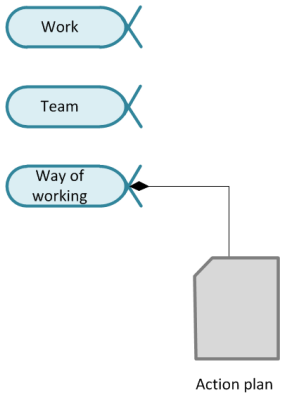


Figure 2. RSM practice: Things to work with

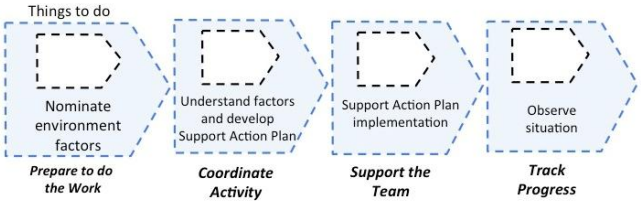


Figure 3. RSM practice: Things to do.

The team guidance in what to do for achieving a particular state of the way of working is the alpha state checklist (Hitchins, 2008). The mapping from the way-of-working alpha states to the RSM activities that must be part of the checklist is showed in Figure 4. The mapping indicated in the figure means that the practice recommends conduct the activity "Understand factors and develop Support Action Plan" (by executing the steps five and six of RSM) to achieve the state of Foundation Established.

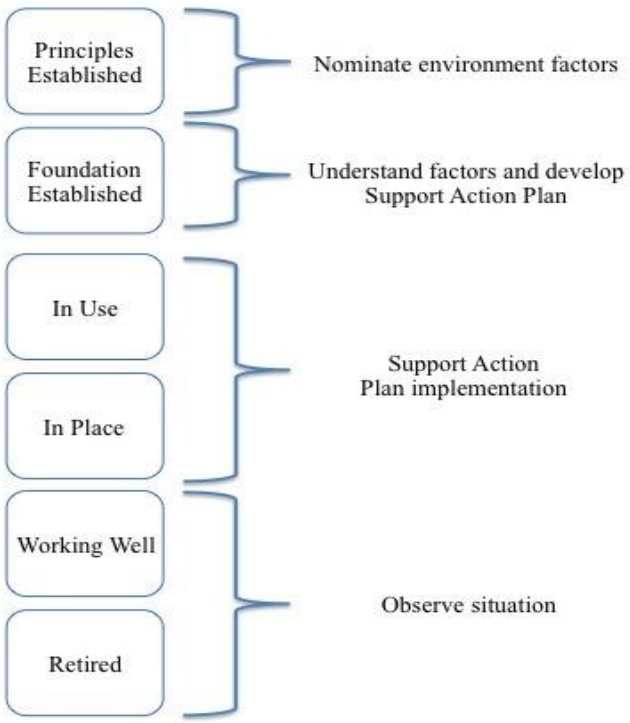


Figure 4. Using SSM practice: Mapping way-of-working alpha states to activities.

4 CONCLUSION AND FUTURE WORK

Among software quality attributes, reliability is commonly one of the most important attributes. Reliability quantifies software faults and failures, which can lead to serious consequences in software systems. Software developer errors have direct influence in the software system reliability, as on whatever occasion that a software developer makes a mistake in their work, faults are injected into the software.

Developer error may happen independently of the development method adopted by the software development team. RSM practice is a software engineering approach to enable software development managers to understand development environment that have influence in developer error, and develop a Support Action Plan to deal with it.

The authors of this paper are conducting researches about extensions of SEMAT kernel alphas in order to offer an answer that considers the inherent complexity of SSD process, in which developer error has presence.

5 REFERENCES

Amin, A., Grunske, L. & Colman, A. 2013. An approach to software reliability prediction based on time series modeling. *Journal of Systems and Software* 86(7): 1923-1932.

Brooks, F.P. 1997. No silver bullet: essences and accidents of Software Engineering. *IEEE Computer* 20(4): 10-19.

Checkland, P. 1999. *Systems Thinking, Systems Practice: Includes a 30-year Retrospective*. Chicheste, New York: John Wiley & Sons.

Clarke P., O'Connor, R.V. 2012. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology* 54(5):433–447.

Hitchins, D.K. 2008. *Systems Engineering: A 21st Century Systems Methodology*. Chichester: John Wiley & Sons.

Jacobson, I., Ng, P.W., McMahon, P.W., Spence, I., & Lidman, S. 2012. The Essence of Software Engineering: The SEMAT Kernel. *Queue 10* 10(10).

Jacobson, I., Ng, P.W., McMahon, P.W., Spence, I., & Lidman, S. 2013. *The Essence of Software Engineering: Applying the SEMAT Kernel*. New Jersey: Addison-Wesley Professional.

Musa, J. 2004. *Software Reliability Engineering: More Reliable Software, Faster, and Cheaper*. 2nd ed., Bloomington: Author House, Bloomington.

Okamura, H., Dohi, T., & Osaki, S. 2013. Software reliability growth models with normal failure time distributions. *Reliability Engineering & System Safety* 116: 135-141.

Poppendieck, M. & Poppendieck, T. 2003. *Lean software development: an agile toolkit*. Upper Saddle River, NJ: Addison-Wesley.

Rasmussen, J., & Vicente, K.J. 1989. Coping with human errors through system design: implications for ecological interface design. *International Journal of Man-Machine Studies* 31(5): 517–534.

Rekab, K., Thompson, H. & Wu, W. 2013. A multistage sequential test allocation for software reliability estimation. *IEEE Transaction on Reliability* 62(2): 424-433.

Senge, P. 2006. *The Fifth Discipline: The Art & Practice of the Learning Organization*. New York: Currency Doubleday.

Stutzke, M.C & Smidts C.S. 2001. A stochastic model of fault introduction and removal during software development. *IEEE Transactions on Reliability* 50(2): 184-193.

Weinberg, G.W. 2011. The psychology of computer programming. Silver Anniversary Edition. [Kindle for Mac, version 1.10.3]. Retrieved form Amazon.com.

Xiong, J., & Li, L. 2013. Nonlinear and Quantitative Software Engineering Method Based on Complexity Science. In *Recent advances in Computer Science: Proceedings of the 17th International Conference on Computers*.

Chapter #9

SEMAT-kernel-based formulation of the HAR'D Snow project practice

R. Sánchez-Dams, N. Amaya-Tejera, & A. Jaramillo-Fuenmayor
Universidad de la Costa, Barranquilla, Colombia

1 INTRODUCTION

Technological project formulation and execution involves challenges, risks, and problems. Some of them are controlled objects and, consequently, study objects in development methodology formulation (Project Management Institute, 2009). Several existing methodologies are based on work environments, models and theories for technological development projects, especially in the software area (Fulbright, 2013).

Particularly, some technological projects covering software and hardware are submitted as trends. This modality does not guarantee proposal award or approval for project execution, since no return on investment and effort in project formulation are supported (Gordillo, 2003). Additionally, such modality implies some limitations in project scope, total cost of the proposal, and project development time (Correa, 2003).

Project formulation is a crucial phase for enterprises. At this phase, achieving clarity and definition of the project is essential, since reliability is reached by projecting costs, investments, and real profits, the basis for project execution (Project Management Institute, 2009). Also, during this stage organizational resources should be efficiently used, because contract award is not guaranteed by the formulation phase.

In this Chapter se present a practice for project formulation in a formal way, involving integration and/or development of electronic and software solutions. The formal way is an adaptation of SEMAT-kernel-based practice formulation.

This Chapter is organized as follows: in Section 2 we present some background about practices[1]; in Section 3 we propose the general structure of the HAR'D Snow project formulation practice; in Section 4 we define a way to implement the practice; in Section 5 we discuss conclusions and future work.

2 PRACTICE BACKGROUND

The method used for practice definition of project formulation was defined by Sánchez Dams (2013). The entire lifecycle is mainly focused on hardware, and some compatibility with software co-design is provided. Sánchez Dams (2013) translates project formulation procedures to a kernel, constituting the practice presented and the subject of this Chapter. The method is organized under a Work Breakdown Structure (WBS; Project Management Institute, 2009) and it is based on a careful observation of enterprises with activities involved in electronic and software development systems. Additionally, specific approaches are adopted during research phases.

Data collection is the first phase. Specialized databases were included, as well as online resources, references materials, books, articles, and magazines as a secondary source of information. We aim to establish the current state of the art about formulation of development projects involving hardware and software. Thus, a theoretical base on existing concepts and approaches helping practice formulation by using a kernel was obtained.

Once the team identified the reference framework with data collection, an instrument was developed in order to capture primary data to characterize strategies and best practices (Sánchez Dams, 2013). The instrument comprises a survey and an interview, and some research guidelines were provided (Sampieri *et al.*, 2006), starting from variable and indicator definition for the measurement process. Also, a technical evaluation standard stepwise model called CMMI-DEV v1.3 (Chrissis *et al.*, 2011) was used. The survey comprises closed questions for objectively identifying the current state of companies and comparing results. Instead, interview comprises open questions for identifying internal entrepreneurial processes for project formulation. The whole instrument is a

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

comprehensive assessment about second level of CMMI maturity. Also, the instrument addressed all process engineering areas belonging to CMMI level three. After the design, the instrument was reviewed by two peers in order to ensure relevance.

A non-probabilistic sample was selected to implement the instrument. Enterprises in the sample were engaged with hardware/software development, commonly performing embedded solutions, automation, control systems, and electronics in general. The designated staff was contacted in each organization, and we use the instrument in seven enterprises according to the guidelines established.

Once the information was obtained, we perform a quantitative analysis (closed-question-based survey) and a qualitative analysis (open-question-based interview) for comparing the findings and the concepts available in the current state of the art. So, we obtained a particular characterization of the identified organizations, including proper procedures, problem areas, solutions, and complementary aspects available in current state of the art.

Finally, the practice was synthesized by using a heuristic approach by using brainstorming with the researchers involved in the project. Besides, we added the experience of several researchers in companies which were involved, but they were not part of the implementation of the instrument. By the time of this book, the practice has been successfully used in the development of projects within the research group where was conceived.

3 OVERVIEW OF THE HAR'D SNOW PROJECT FORMULATION PRACTICE

Similarly to the Figure 1 of the Chapter 2 of this book, where the modern and traditional lifecycles are compared by Huang and Ng, we propose the lifecycle depicted in Figure 1. This is an intermediate approach to the other two lifecycles, since some cost, scope, and time constraints are considered. The HAR'D snow project formulation practice restricts the scope by complying with the opportunity presented, setting time and price. The practice effort is focused on a suitable formulation of the project, while the organization only employs the strictly necessary staff and resources.

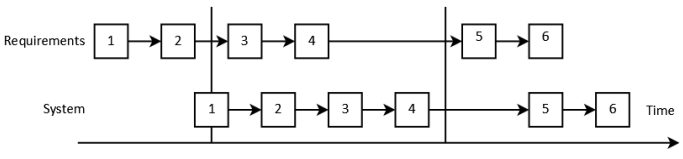


Figure 1. Proposed practice lifecycle of the formulation of the HAR'D Snow project practice

.

Concerning this practice, the project requirements are articulated and defined for addressing the opportunity, so the alpha takes the state *bounded*. However, the requirements are not refined, so the consistency is not reached. Related to the alpha *software system*, the state *architecture selected* is partially addressed, because we need to propose a project architecture including hardware, since the hardware price can be representative in projects, increasing the effort devoted to the requirements development. Once the project is approved, the proposed architecture should be adjusted.

Advantageously, the practice developed is lifecycle-independent, so it is compatible with both modern and traditional approaches. Compatibility is possible because the conception of the project is divided into two stages: in the first one—addressed by the practice itself—project scope and cost are defined; in the second one—related to project development—the finer aspects of planning and technical implementation activities are discussed. In the particular case of the waterfall process, once approved and funded the project, from the practice formulation we can completely define the requirements and make a fine planning. In modern projects, work products serving as inputs for planning iteration are obtained from the practice.

The project formulation is early applied early in the lifecycle, aiming to the project design. The proposed practice is oriented towards organizations developing electronic and computing technology. We aim to generate a global project development planning for defining milestones, scope, resources, time, and costs. Some criteria for a suitable formulation from the developer perspective are defined as (Sánchez Dams, 2013): (i) compliance with the reference terms; (ii) generations of benefits provided to the developer from design; and (iii) efficiency in resource usage. In addition, the formulation is considered successful if approved or funded.

Under the restrictions described to formulate the project, the developer needs to understand the domain of the problem and the opportunity. From the area of concern *customer*, the requirements are set forth and the purpose is identified in favor of defining the scope of the *software system*. Developers should analyze the *software system* to conceptualize the solution and the architecture. We also need to define the project *endeavor*, resources, and costs. A project overview is also desirable including general planning, milestones, and *software system* releases. In this way, the project formulation practice addresses the next advance in the kernel alphas.

The six alphas involved are depicted in Figure 2. The alpha *stakeholders* is required to be recognized and represented. *Opportunity* advances to solution needed, and depending on the formulation conditions, to value established (the state in dotted line). *Requirements* advance two states without reaching the state coherent, because the description of the requirements is postponed until the project execution. In *software system*, the state architecture selected is reached. Final-

ly, *work* and *team* have reached, respectively, the states initiated and seeded. The alpha *way of working* is not addressed by the practice, because its states are reached once the project is approved.

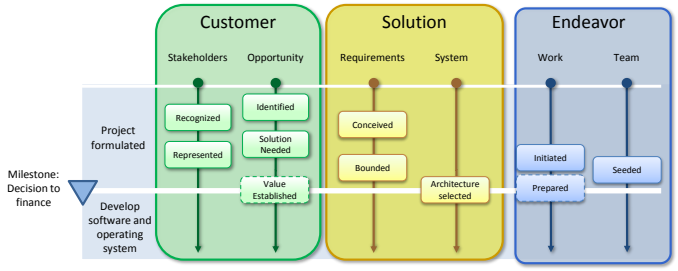


Figure 2. Global planning board: alpha progress to milestone *project formulated*.

Project formulation practice allows for identifying the defined elements in the context of the SEMAT kernel: alphas and sub-alphas involved, work products bringing evidence to the progress in the alpha states, and the roles and activities linked to the product development.

The Alphas directly related to the practice are: requirements, software system, and work. Figure 3 depicts the practice structure.

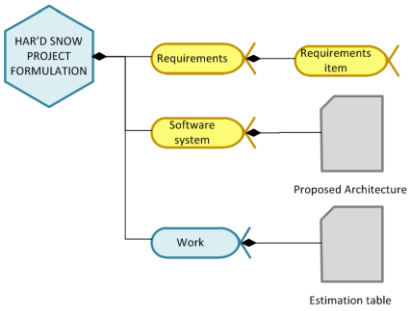


Figure 3. Outline of the practice: HAR'D Snow project formulation.

Sub-alpha *requirements item* is added to the alpha *requirements* as defined in the standard SEMAT kernel (Submitters, 2014). Such sub-alpha represents the advance in the state of a single requirement. The work product *proposed architecture* is assigned to alpha *software system*. Finally, we propose a work product for organizing all the formulation and we assign it to the alpha *work*.

The sub-alpha reaches the first state, and enters a second state called *estimated*. The proposed architecture is an artifact of the analysis result suggesting a general solution at the level of analysis, defining the architectural elements of the global solution. The estimation table is a work product developers make as evolving the practice. Initially, the estimation table is the container of statements belonging to requirement items, but it is the device proposed to define the whole project formulation.

We define some activities and competencies (Submitters, 2014) related to the HAR'D Snow project formulation and we classify them into areas of concern, as depicted in Figure 4.

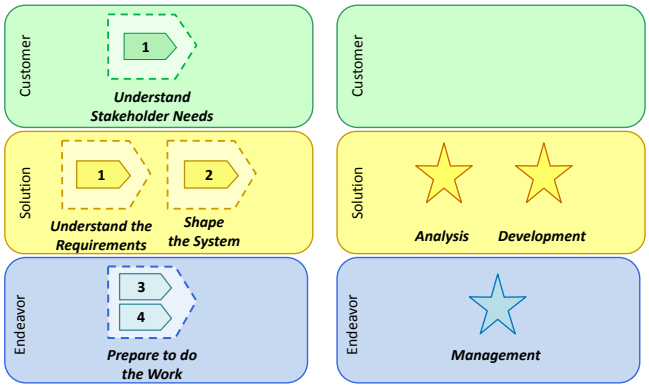


Figure 4. Activities and competencies of the HAR'D Snow project formulation.

The proposed activities for practice are listed below:

1) **Prioritize requirements items.** Categorize the relative priority of requirements items in terms of the delivering value to the customer and the importance to the project.
2) **Propose architecture.** Define the technical path to take on the project.
3) **Estimate project effort.** Calculate the effort required for each requirement item.
4) **Estimate scope, time, and cost.** Quantify the total scope of the project, considering time, cost, and utility.

The SEMAT kernel (Submitters, 2014) includes a set of competencies for development, which are established by ability, essential skills, and levels of apprehension. Practice makes use of the following competencies: analysis, development, and management. Competencies are proposed from the perspective of the developer organization, so the stakeholders should be coordinated by the customer.

4 IMPLEMENTATION OF THE HAR'D SNOW PROJECT FORMULATION PRACTICE

In this section we explain the usage of the practice. We use activities as guides to describe how the alpha states are reached for completing the entire formulation. As the SEMAT kernel promotes, we use the "things we always do," the "things we always work with," and the roles who participate.

4.1 *Requirement item*

The sub-alpha *requirements item* is used as an artifact representing progress in the states of a single requirement. The

state *identified* is not addressed in the project formulation practice, so another practice should address the accomplishment of this state. We added the state *estimated*, increasing the checklist with the following criteria: globally prioritized, definite endeavor, proposed resources, and assigned and estimated cost. The remaining statements are outside the scope of this practice and they are: described, implemented, and verified.

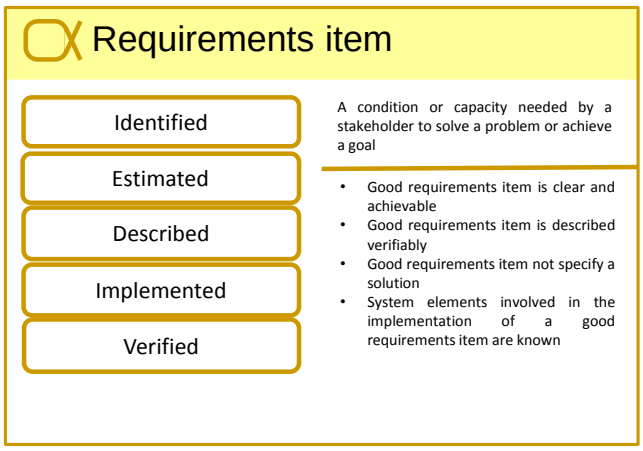


Figure 5. Card of sub-alpha requirements item.

4.2 *Checklist of the alpha state cards*

At the beginning of a project, the seeded team is not recognizing or fully understanding the opportunity. Also, stakeholders are unable to express the opportunity in terms of requirements limiting the total project effort. For the sake of addressing these challenges, we need to invest some effort in the initial stage of the requirements elicitation and definition prior to estimate the project. In this practice, the mechanism for identifying requirements is freely chosen. Due to resource constraints established to develop the project, the state *described* is not reached. The practice only has brief descriptions of the requirements items, unless more detail is needed for any item. So, time spent in defining requirements items is postponed to the state *estimated*. Therefore, developer members can understand the requirements so that everyone can agree on the nature of the software system to be built.

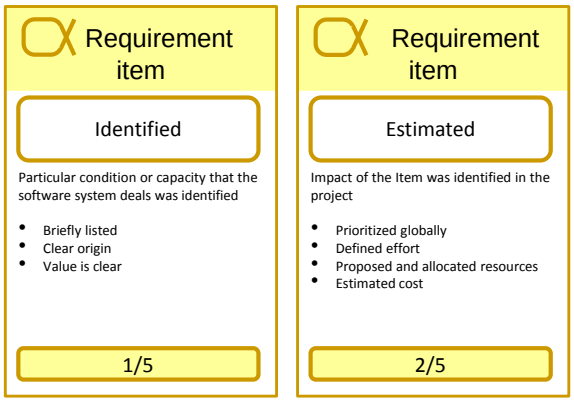


Figure 6. Cards of states of requirements items, used in practice.

4.3 *Estimation table*

Requirements items are organized and estimated in the estimation table, as shown in Table 1. Requirements items are prioritized, and values indicating difficulty, extension, and complexity are provided. Indicators are used to calculate the effort, the hours, and the cost.

Estimation table							
Requeriments item	Priority	Difficulty	Extension	Complexity	Effort	Hours	Cost
Requeriments item 1	5	3	4	1,5	18	122	$ zzzz
Requeriments item 2	5	3	2	1	6	41	$ zzzz
Requeriments item 3	4	5	4	1,2	24	163	$ zzzz
Requeriments item X	3	5	2	1,1	11	75	$ zzzz
Requeriments item L	2	2	2	1,3	5,2	35	$ zzzz
Requeriments item M	1	1	2	2	4	27	$ zzzz
Requeriments item Z	1	3	4	1,3	15,6	106	$ zzzz
Total					**83,8**	**568**	**$ YYYY**
Averages					**12**	**81**	

Table 1. Estimation table

4.4 *Practice activities*

In Figure 7, the contribution of activities for advancing the alpha states directly involved in practice is shown. Some guidelines of the practice share all of the activities, described as follows. Each activity should be developed at least by two people, including analyst and developer roles. The number of participants in the developer organization should be limited to the lower possible. We recommend two-to-three people, with the possibility of including specific technical roles, according to the necessity and difficulty of the project. In the last guideline is established that each activity has steps should be performed in sequence, but not necessarily a step immediately after the other.

Estimation can be calculated with the help of a moderator during group sessions. Consensus can be reached by using several techniques like planning poker.

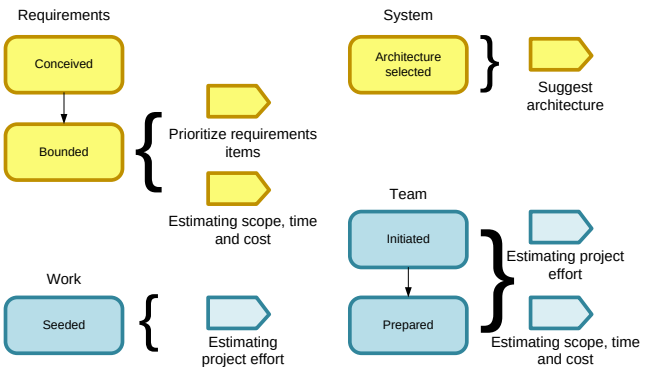


Figure 7. Contribution of the project development activities to the states of the alphas.

4.5 *Prioritize requirements items*

This activity demands the opportunity identified and stakeholders recognized and represented. We aim to develop a shared understanding of the product among developers and stakeholders. This activity is divided into three stages: prepa-

ration, meeting with stakeholders, and adjustments. The requirements are organized and priority is assigned.

In pursuance of this activity, the requirements should be necessarily conceived prior to the developer group meeting. At this meeting, each requirements item is reviewed for identifying short phrases. Besides, items are placed in any order in the estimation table. Meetings with the representatives of stakeholders are scheduled, either at the beginning or the end of the workday. The stakeholder meeting is limited to two hours of continuous and uninterrupted work. The moderator leads the meeting, and the rest of the group is responsible for updating the estimation table and taking notes. The meeting begins with an explanation of the technique, as follows. An item is chosen, and priority is assigned. Requirements items are prioritized by using a 1-to-5 scale in the table, being 5 the highest priority value. The moderator asks explanation of the rating given to each item, since he/she should assign the prioritization uniformly, and avoiding situations like the choice of most high-priority requirements. Finally, once the stakeholder meeting is completed, the developer group will meet with the proposed architecture in mind for adjusting the priority set. Architectural aspects are important in the prioritization because they establish technical support to help to develop other requirements items, and thus establish the order of priority.

4.6 *Propose architecture*

The architecture—in the context of the formulation—helps to estimate the requirements item and adjust its priority. The architecture also determines the hardware should be paid for, leaving details of their final selection for execution. This activity is interrelated to the other, as the developer group increases their knowledge while the proposed solution is formulated. This activity is not enough to fully understand all the checklist of the state selected architecture. However, an initial architecture is proposed. The architecture will evolve once the project is approved and development is started, subject to changes in the hardware and software features.

The architecture is the set of elements for defining the solution and the structure, *e.g.*, programming languages, computers, platforms, logical processing of data, and hardware and software functionality. The developer group is free to choose the strategies they will use. Architecture should conceptualize the solution in large functional blocks, setting its configuration and organization. Some examples are the block diagrams for hardware and the use cases and the user stories for software (Jacobson *et al.*, 2011).

Every project involves risks, so from project formulation we should identify and define how to mitigate risks during development. By identifying such risks, the proposed architecture is analyzed and the estimation activity is enriched. The proposed architecture also limits the tools to use in the development and affects the effort estimation. The selected basic hardware and the tools should facilitate the work, affect the estimation of the difficulty and extent of the development on each requirements item. Examples of aspects to consider are learning curves and previous experience of the group.

4.7 *Estimate Project effort*

The effort is an estimated investment of resources needed to satisfy a measuring related to a requirements item. This measure has no units and is relative to the other items of the project, which involves difficulty, extension, and complexity. The difficulty relates to the appreciation the development team has on how easy or problematic is the estimated requirements item. This can be chosen based on previous experience, familiarity of the team members with an item. The difficulty is linked to the identified risks or inconveniences that may result in the item. Likewise, difficulty degree rises with a growing number of disadvantages or risks. Extension refers to how long is the solution implementation, once the way to do it is understood. When more time is required, then the indicator is increased. The last aspect of the effort is complexity, which is an iteration measurement of one requirements item compared to others. High grades of dependence imply high complexity.

Difficulty and extension are rated between 1 and 5. However, complexity is rated in a continuous range between 1 and 2 (*e.g.*, 1, 1.5, 2, 1.3, etc.). Conducing to make these estimates, a two-hour session is performed in similar conditions to the activity *prioritize requirements items*. The stakeholder participation is optional depending on the terms of the opportunity, the way of working of the developer organization, and the relationships with the customer. The estimation is done by a consensus discussion to choose the score or alternatively using the planning poker technique. Once estimates of these indicators are made, we can determine the effort required for each item, calculated as the product between extension, complexity, and difficulty indicators. As a result, the stress increases geometrically with increasing some aspect involved.

4.8 *Estimate range, time, and costs*

Understanding of this activity estimation is explained from a conceptual perspective. Computing details are available in spreadsheet templates, beneficial to ease the practice implementation.

4.9 *Estimate simplified description*

For the sake of simplification, we can assume the development team has enough experience in estimating effort and work hours, and the approached project has low risk, because they have previously executed similar projects. So, the team has no need to divide the work for each item of the project.

The development group meets in a work session as mentioned above. The main difference with previous meetings described is the participation of all available, involved in the

project staff from the developer group. The moderator with the working group, according to his/her experience chooses a few requirements items (*e.g.*, 1-3) as a reference. The criterion of choice is familiarity with its implementation. The reference requirements item is marked in the estimation chart. Then, the estimated man-hour dedication to complete each reference requirements item is calculated. Next, a cross-multiplication is applied to estimate other requirements item, based on the pattern set.

After that, we can estimate the gross cost of the project. The estimation is based on workdays per week, for example, five days on and two off. The times in Table 2 are estimated at 100% productivity of a business day, but developers have some rest to drink water, read emails, and meet with each other, among other things. For example, in an 8-hour labor day, labor productivity is almost 75%. Consequently, we can assume only 6 effective hours per labor project day. In addition, the average salaries of people involved in the project should include all legal aspects of social benefits and services. Also, the total value of the staff should be added to the cost of purchasing equipment and infrastructure, resulting from the proposed architecture, and to the cost of additional resources such as transport and subsistence. All of these additional costs are released in the calculation procedure.

Estimation table							
Requeriments item	Priority	Difficulty	Extension	Complexity	Effort	Hours	Cost
Requeriments item 1	5	3	4	1,5	18	122	
Requeriments item 2	5	3	2	1	6	41	
Requeriments item 3	4	5	4	1,2	24	163	
Requeriments item X	3	5	2	1,1	11	75	
Requeriments item L	2	2	2	1,3	5,2	35	
Requeriments item M	1	1	2	2	4	27	
Requeriments item Z	1	3	4	1,3	15,6	106	
Pattern average estimate					14,5	98,3	
Total					**98,3**	**666**	**$ YYY**
Averages					**12**	**83**	

Table 2. Man-hour dedication estimation.

In terms of scope, according to the available time and cost limits established for the project, all the requirements items needed to deal with or solve the problem are included. The expected project scenario comprises such items. Then, developers consensually established two scenarios—pessimistic and optimistic—based on a success rate lower than expected. For example if the expected scenario reaches the requirements item "k" with 1000 endeavor hours, and if the worst scenario is set to 80% can only get to the requirements item, "i" equivalent to 800 work hours. The project scope is defined between the expected and the pessimistic scenario, but the costs are estimated with the best scenario, as depicted in Table 3.

After obtaining the gross cost, we can find the final price of the project. This is an administrative and financial aspect, a non-technical issue, so freedom is given in the calculation method. Project price depends on the internal policies of the developer organization such as profit margins, infrastructure maintenance, and prestige, among others. According to this fact, the organization charges an additional percentage on the project cost. This also includes an assessment of the organizational infrastructure that holds the project, such as secretaries and support staff, space, accounting departments, go shopping, etc.

Estimation table					
Requeriments item	Priority	Effort	Hours	Cost	
Requeriments item 1	5	18	122	$ zzzz	
Requeriments item 2	5	6	41	$ zzzz	
Requeriments item 3	4	24	163	$ zzzz	**Pessimistic scenario**
Requeriments item X	3	11	75	$ zzzz	
Requeriments item L	2	5,2	35	$ zzzz	**Expected scenario**
Requeriments item M	1	4	27	$ zzzz	
Requeriments item Z	1	15,6	106	$ zzzz	**Optimistic scenario**
Pattern average estimate		14,5	98,3		
Total		**98,3**	**666**	**$ YYYY**	
Averages		**12**	**83**		

Table 3. Estimation scenarios: pessimistic, expected and optimistic.

4.10 *Plan globally*

This is optional activity to the practice, as depicted in Figure 8. The planning starts from the activity of prioritize requirements items with the premise of implementing the first thing delivering greater value to stakeholders, in pursue of ensuring compliance range. Despite this, with cut development in the *worst case scenario* implementation to know the opportunity is planned. The project opportunity is limited to the terms of reference. By defining the range, we are looking forward to adjust the level, depth, and ease of use solution. The defined range minimally complies with the proposal opportunity, but still achievable with the resources, cost, and time defined.

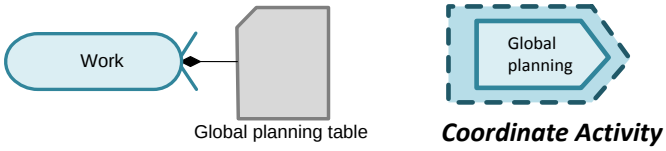


Figure 8. Optional aspects of the practice

The overall project planning sets important milestones in the project, and the states of progress of the alphas. A milestone is defined as a point of interest in the project, which provides a significant advance in the proposed solution, established by stakeholders and developers. This approach does not require a detailed plan of the entire project, but it sets an agenda and a plan of release indicating the moment of partial deliveries of the solution. Jacobson *et al.* (2013) recommend the usage of generic milestones—skinny system available and usable system available—but the developer group can add other relevant ones. These milestones determine the lifecycle and structure the *global project planning*. According to the needs of the organization we can make the final planning under either the traditional waterfall (*e.g.*, by using a Gantt diagram; Clark, 2012) or the iterative approach. Well-established milestones serve as control points for the SEMAT kernel alphas. Thus, a balanced development for guiding the team endeavor in the implementation of the project is guaranteed. For

this purpose, a set of states of the alphas to be achieved in each release are chosen. Finally, the planning is captured in the work product *global planning board*, as shown earlier in Figure 2. Jacobson *et al.* (2013), an additional help that uses cards statements is provided.

5 CONCLUSIONS

In this Chapter, we formulated the HAR'D snow project practice. Formulating a practice by using the SEMAT kernel made possible to explain the structured technique, concentrating on the relevant aspects of project formulation. We used the principle of separation of concerns belonging to the SEMAT kernel in order to formulate the practice. In this way, we could integrate different styles of organizational work. For example, the practice can be used in conjunction with iterative development methods, waterfall approach, or combined with practical use cases or feature-based requirements elicitation. We also estimated prioritized requirements items in terms of difficulty, extension, and complexity. Estimation was performed for effort, dedication, and cost.

The practice has been applied successfully in the research group GIACUC. Within the group the practice usage has facilitated the development of projects, raising the indicator of funded projects. Project implementation has also benefited, nearby to estimates being executed within the expected range and the pessimistic scenario. The future work can be devoted to formulate other techniques used by the research group under the SEMAT kernel approach to practices. We can also quantify the benefits of implementing the practice presented, establishing a procedure for verification of future practices formulated by the group. Finally, we identified the need for establishing a mechanism to share best practices with the academic community and industry, defined by SEMAT cards and related documentation.

6 ACKNOWLEDGEMENTS

We acknowledge the contributions of Carlos Mario Zapata Jaramillo—Chairman of The SEMAT Latin American Chapter—and Heyder Páez Logreira to this work.

7 REFERENCES

Amaro Calderón, S. D., & Valverde Rebaza, J. C. (2007). *Metodologías Ágiles*.

Chrissis, M. B., Konrad, M., & Shrum, S. (2011). *CMMI for Development: Guidelines for Process Integration and Product Improvement ; [CMMI-DEV, Version 1.3]*. Addison Wesley Professional

Clark, W. 2012. *The Gantt Chart: A Working Tool of Management*. Nabu Press.

Correa, I. 2003. *Manual de licitaciones públicas*. Naciones Unidas, CEPAL, ILPES.

Fulbright, R. 2013. Incorporating Innovation into Iterative Software Development Using the Inventive Problem Solving Methodology. *International Journal of Innovation Science* 5(4): 203–212.

Gordillo, A. A. 2003. *Tratado de derecho administrativo: Parte general*. Fundación de Derecho Administrativo.

Jacobson, I., Ng, P.-W., McMahon, P. E., Spence, I., & Lidman, S. 2013. *The essence of software Engineering: applying the SEMAT kernel*. Addison-Wesley.

Jacobson, I., Spence, I., & Bittner, K. 2011. *USE-CASE 2.0 The Guide to Succeeding with Use Cases*. Ivar Jacobson International SA.

Project Management Institute. 2009. *Guia De Los Fundamentos Para La Direccion De Proyectos /A Guide to the Project Management Body of Knowledge (PMBOK Guide): Official Spanish Translation (4a ed.)*. Project Management Inst.

Sánchez Dams, R. D. 2013. *Metodología ágil estandarizada para el desarrollo o ejecución de proyectos de sistemas embebidos.* M.Sc. Thesis. Universidad del Norte, Barranquilla, Colombia.

Submitters, O. M. G. 2014. *Essence–Kernel and Language for Software Engineering Methods. Beta 2.* Available http://www.omg.org/cgi-bin/doc?ptc/2014-02-26

This page intentionally left blank

Part III: Teaching

For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable.

— Edsger W. Dijkstra (My hopes of computing science, EWD 709, 1979)

This page intentionally left blank

Chapter #10

Identifying the scope of Software Engineering for Beginners course using ESSENCE

G. Ibargüengoitia & H. Oktaba
Facultad de Ciencias, Universidad Nacional Autónoma de México, Mexico City, Mexico

1 INTRODUCTION

1.1 *Motivation*

Software Engineering has evolved over the last few years. The description of its foundations was improved in SWEBOK (IEEE, 2014), the models (CMMI, 2010), and standards (ISO/IEC 29110, 2011) has been updated, and the Agile Manifesto has made practices more flexible (Agile Alliance, 2001).

In addition, the Software Engineering Method and Theory initiative (SEMAT 2014) called for a proposal to develop a new OMG (Object Management Group) standard that defines the kernel of basic concepts that should be addressed when developing software systems. As a result of the call for action, the ESSENCE proposal, directed by Ivar Jacobson (Jacobson *et al*., 2013), emerged. ESSENCE incorporated KUALI-BEH, which is a Mexican proposal.

Teaching Software Engineering to the new generations should evolve addressing these changes as quickly as possible. For the past 12 years the authors have taught Software Engineering to students at the undergraduate level by using the educational proposal of TSP (Humphrey, 1999), Unified Process (Jacobson *et al*., 1999), and UML (Rumbaugh *et al*., 1998).

In 2013 the authors used KUALI-BEH for reviewing the Software Engineering course. They expressed a method and practices for maintaining the essentials of traditional process and including agile practices primarily based on SCRUM (Schwaber, 2011) and KANBAN (Anderson, 2010).

The new *Software Engineering for Beginners* course teaches how to understand and experiment a set of social, managerial and development practices, working in teams in a software project. This new proposal has proved to be successful with two generations of students. The success was reflected in an increased student involvement in the project, a better team collaboration and an improvement of the quality of software systems delivered at the end of the course. These results were compared to the performance of students in generations preceding the update.

One of the SEMAT[1] initiatives is to promote the use of ESSENCE (Jacobson *et al.*, 2013) for educational purposes. With this in mind, an interest has emerged to analyze which ESSENCE kernel elements are covered within the *Software Engineering for Beginners* course. The objective is to identify the alphas and their states, activity spaces and competencies covered by the *Software Engineering for Beginners* course. In other words, we use the ESSENCE kernel to assess the scope of our educational proposal.

Our hypothesis is that even though *Software Engineering for Beginners* course has learning limitations, it allows for converting all alphas (but not all their states), activity spaces, and several competencies until the level 2 (*Applies*).

In Section 2 we shortly describe the KUALI-BEH extension to the alpha *way of working*, which is used for the course method and practices definition. In Section 3 we describe the *Software Engineering for Beginners* course, including preconditions, method, and work guidelines. In Section 4 we use the ESSENCE kernel in order to identify the scope of *Software Engineering for Beginners* course. The coverage of alphas and alpha states, activities spaces, and competencies is analyzed. In Section 5 we discuss the findings of the previous section. Finally, in Section 6 we present the final conclusions.

2 KUALI-BEH MAIN CONCEPTS

KUALI-BEH (KUALI: Nahuatl word meaning *good*, *fine* or *appropriate*, BEH: Mayan word meaning *way*, *course* or *path*) is an extension to the ESSENCE kernel developed by a group from the Universidad Nacional Autónoma de México

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

(UNAM). The KUALI-BEH extension provides four additional Alphas to allow teams the expression of their way of working and the progress of their work in software projects. For the purpose of this Chapter we will use only Method and Practice Authoring alphas as sub-ordinate alphas of way of working.

According to the KUALI-BEH, "The Practice Authoring Alpha allows the practitioners to express work units as practices. The Method Authoring Alpha can compose these practices as methods. Practice and Method Authoring Alphas help to articulate explicitly the practitioners' Way of Working" (Jacobson *et al*., 2013).

The definitions of the Practice and Method Authoring Alphas and its corresponding states are the following (Jacobson *et al*., 2013):

- Practice Authoring: It is the defined work guidance, with a specific objective, that advises how to produce a result originated from an entry. The guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result. The completion criteria associated with the result are used to determine if the objective is achieved. Particular competences are required to perform the practice guide activities, which can be carried out optionally using tools. In order to evaluate the practice performance and the objective achievements, selected measures can be associated to it. Measures are estimated and collected during the practice execution. The Practice Authoring states are: *Identified, Expressed, Agreed, In Use, In Optimization* and *Consolidated*.
- Method Authoring: 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. The Method Authoring states are: *Identified, Integrated, Well Formed, In Use, In Optimization* and *Consolidated*.

For details of Practice and Method Authoring Alphas see the ESSENCE Annex B (Jacobson *et al*., 2013).

3 SOFTWARE ENGINEERING FOR BEGINNERS COURSE DESCRIPTION

3.1 *Course description*

The purpose of the undergraduate course is to introduce students to software engineering practices balancing the use of disciplined and agile methods (Boehm *et al*., 2004). The central focus of this course is to *teach by doing*. The students will learn software engineering by doing teamwork and by developing a software system, which should be functional at the end of the course.

The authors believe that the most important thing for a first course of software engineering is to teach software development, teamwork, and project management practices. The exposure of students for dealing with customers is postponed for a later course. For this reason, at the beginning of the course the professors provide the students with a *Statement of Work*, which includes the objective, scope and features of software system to be developed.

3.2 *Course preconditions*

Course preconditions are:

- Students have knowledge about programming, data structures, algorithms and databases.
- A professor and two assistants are assigned to a group of about 25 students.
- The initial method for software development is defined in order to guide the way of working of the students.
- The course duration is 16 weeks including 3 hours of theory and 4 hours of practice per week.
- The course script, including the week schedule of activities, is defined.
- A Statement of Work (ISO/IEC 29110, 2011) including the description of software system to be developed by each team during the course is defined.
- The type of software system to be developed is a web application.
- The architectural pattern used by students is the Model View Controller (MVC; Buschmann *et al*., 1996).

3.3 *Course method*

The method of the *Software Engineering for Beginners* course, called *Initial Method for Software Development*, is a blend of social, management and development practices required by teams for developing a software system.

Professors used KUALI-BEH Practice and Method Authoring Alphas for describing the content of the Initial Method for Software Development. First, social, management and development practices were identified, based on teaching experience and by using a combination of TSP (Humphrey, 1999), Unified Process (Jacobson *et al*., 1999) enriched with Agile and ISO/IEC 29110 Basic Profile (ISO/IEC 29110, 2011) proposals. The practices were agreed on by the professors and expressed using the KUALI-BEH practice template. Once the individual practices were selected, expressed and agreed, they were integrated into the method, using the corresponding template.

The objectives of the practices, their entries, and results were revised and adjusted for accomplishing the coherency, consistency, and completeness properties of the method. The *Well-Formed* state of the Method Authoring Alpha and the *Agreed* state of the Practice Authoring Alpha were reached. The *Initial Method for Software Development* was ready to

put it *In Use* state to guide the students in their Way of Working during the *Software Engineering for Beginners* course.

The Initial Method for Software Development purpose is:

- Introducing students in Software Engineering discipline and agile practices.

The Method entries are:

- Students have enough programming knowledge for developing a software system.
- A Statement of Work for the software system is defined.
- A Course *Script* is defined.

The Method expected results are:

- Students have learned and performed software engineering basic practices.
- Student teams developed the functional software systems.

The Method practices are:

Social practices:

- (SP1) Forming the Team and Putting the team name and logo. Each team member selects one of the following roles: Team Responsible, who is accountable for encouraging team members to do their work, helping to solve issues and risks, holding up the teamwork, and interacting with the professor; Quality Responsible, who helps to enforce standards and product integrity; Technical Responsible is in charge of all technical support in development decisions; Collaboration Responsible, who helps to maintain communication and agreements of the team by means of collaborative tools. Additionally, every team member plays the role of Developer, taking care of at least one use case for identification, description, coding, unit testing and integration.
- (SP2) Defining Team Communication. This practice is based on SCRUM Daily Meeting (Schwaber, 2011) and it includes the work item card states updating on the virtual card wall (Anderson, 2010).
- (SP3) Creating a Common Repository. This practice is based on activities proposed in ISO/IEC 29110 (2011) and consists of setting up the team documents and code repository.
- (SP4) Iteration Retrospective. This practice is based on the SCRUM Retrospective.

Management Practices:

- (MP1) Project Planning. A plan is created according to activities (ISO/IEC 29110, 2011) and the *Course Script*. Also, teams create the General Use Case Diagram for the *Statement of Work*.
- (MP2) Iteration Planning. The use cases for the iteration are selected, at least one use case for each team member is assigned, and the virtual card wall with initial work items cards is created.
- (MP3) Executing the Plan. The planned work items are executed and the work item card states on the virtual card wall are updated.
- (MP4) Assessing and Controlling the Iteration. The progress of the work is assessed by reviewing the states of the work items cards on the virtual card wall. Eventually, new work item cards are defined to attend issues and risks.
- (MP5) Closing the Iteration. The partial software system and the documentation are delivered by the teams in order to be evaluated by the professors.
- (MP6) Closing the Project. The final software system and the documentation are delivered by the teams for final course evaluation.

Development Practices:

- (SDP1) Software Requirements. The iteration use case diagram is built. Each team member selects one or more use cases, details them, creates the prototype of the user interface and provides test cases. All the team members define non-functional requirements for the software system.
- (SDP2) Software Design. The software system is a web application and the professors impose the usage of MVC architectural pattern. The students describe the architecture with UML package and deployment diagram and define the implementation environment. Each student builds class and sequence diagrams for each use case. The state machine diagram for user interface navigation and the database Entity-Relation diagram are designed by the team.
- (SDP3) Software Construction. Each team member codes her/his use case classes and performs unit tests with the test cases defined in requirements.
- (SDP4) Software Integration and Testing. The Technical Responsible supports the team for integrating all use cases in a software system and testing.
- (SDP5) Software Delivery. The software system and documentation are delivered to the professors in order to be evaluated.

Figure 1 shows the general structure of the *Initial Method for Software Development.*

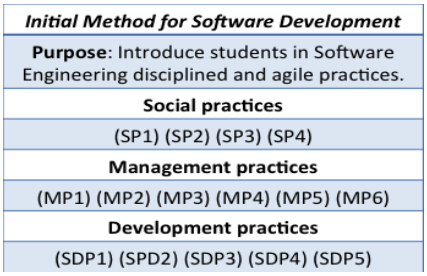


Figure 1. Initial Method for Software Development purpose and practices.

3.4 *Course work guidelines*

The course work guidelines are:

- The professor initially lectures the theoretical software engineering classes and explains the concepts by using the method practices. The practices specify the activities to be performed. Professor requests students to carry out the practices and deliver documents or code, which prove that these activities were performed properly and on time.
- The assistants support students on the generation of practice documents and on the technical aspects involved in the software system development.

A *Course Guideline* contains the general course plan:

- Weeks 1, 2: software engineering introduction, course *Method* presentation, and the execution of the social practices: (SP1) Forming the Team, (SP2) Defining Team Communication and (SP3) Creating a Common Repository.
- Week3: Management Practices: (MP1) Project Planning, (MP2) Iteration Planning, creation of the virtual card wall corresponding to (MP3) Executing the Plan.
- Weeks 4, 5: (SDP1) Software Requirements.
- Weeks 6, 7, 8: (SDP2) Software Design.
- Weeks 9, 10: (SDP3) Software Construction.
- Week 11: (SDP4) Software Integration and Testing.
- Week 12: (SDP5) Software Delivery, (MP5) Closing the iteration and (SP4) Iteration Retrospective.
- Weeks 13 to 16: Second iteration, adding or modifying some use cases, (MP6) Closing the Project.

4 ESSENCE KERNEL USED FOR IDENTIFYING THE SCOPE OF SOFTWARE ENGINEERING FOR BEGINNERS COURSE

4.1 *Alphas and its states*

The purpose is to identify the initial alpha states at the beginning of the course, describe their changes due to the practices execution, and recognize the final states reached at the end of the course. In the following text *cursive* letters are used for distinguishing the alpha states.

- Opportunity: At the beginning of the course the opportunity is *viable* because the professors defined the scope of the software system to be developed. They were based on their experience and the knowledge assumptions of the students. At the end of the course the opportunity is *addressed*. The *benefit accrued* state is out of the course scope.
- Stakeholders: The main educational decision of the *Software Engineering for Beginners course* avoids the exposure of the students to the real life customers. With this assumption, the professors and assistants play the stakeholder role. They are *recognized* and *represented* at the beginning of the course, are *involved* and *in agreement* during the course, and they judge (evaluate) if the systems developed by the students reach the *satisfied for deployment* state. *The satisfied in use* state is out of the course scope.
- Requirements: At the beginning of the course, the software system requirements are defined in the *Statement of Work document*, prepared by the professors. Also, they are *conceived, bounded, and coherent.* The software development practices: (SDP1) Software Requirements, (SDP2) Software Design, and (SDP3) Software Construction, lead the Requirements to the *acceptable* state. The practices: (SDP3) Software Construction and (SDP4) Software Integration and Testing, lead the Requirements to the *addressed* state. Finally, (MP5) Closing the Iteration and (MP6) Closing the Project practices leave the Requirements in the *fulfilled* state.
- Software System: Another important decision made by the professors of the *Software Engineering for Beginners course* is the selection of the architectural pattern MVC, which is used by the students to develop the web application software system. Therefore, we start the course with the *architecture selected* and *demonstrable* due *to the wide* usage of this pattern for this kind of applications. The execution of the (SDP2) Software Design, (SDP3) Software Construction, and (SDP4) Software Integration and Testing practices lead the Software System to the *usable* state. Finally, the (SDP5) Software Delivery, (MP5) Closing the Iteration, and (MP6) Closing the Project practices leave the Software System in the *ready* state. The *operational* and *retired* states are out of the course scope.
- Team: At the beginning of the course, the students are split into groups of 4(+-1), so the Team alpha is *seeded.* The social practices (SP1) Forming the Team, (SP2) Defining Team Communication and (SP3) Creating a Common Repository drive the Team to be *formed.* The management practices (MP1) Project Planning, (MP2) Iteration Planning and (MP3) Executing the Plan allow for learning how to become *collaborating Team.* The (MP4) Assess and Control the Iteration practice shows

how to be *performing Team*. Finally, (MP6) Closing the Project practice allows the movement to the *adjourned* state at the end of course.

- Work: At the beginning of the course all the students start with *initiated* Work. The (MP1) Project Planning and (MP2) Iteration Planning practices lead them to have the Work *prepared*. The (MP3) Executing the Plan practice makes Work *started* and (MP4) Assessing and Controlling the Iteration practice sets Work *under control*. Finally, the practices (MP5) Closing the Iteration and (MP6) Closing the Project lead to *conclude and close* the Work.
- Way of Working: Designing the *Software Engineering for Beginners course* the professors have *established foundation* for the Way of Working of the students. In our case, we used the KUALI-BEH Practice and Method Authoring alphas for defining the *Well Formed Initial Method for Software Development*. It means that this method is composed by the coherent, consistent and complete set of practices. During the first iteration, the Way of Working defined by the method is *In Use* and *In Place* learned *and* experienced by the students for the first time. After the first iteration (SP4) Iteration Retrospective practice the method begins to be *Working Well* for the second iteration. Finally, at the end of the course the method is *Retired*, but *we hope the* method is not forgotten by the students.

In summary, students learn and apply practices that allow for them to achieve all Team and Work alphas states. They experiment almost all states of Stakeholders by means of the interaction with the professor and the assistants, except the *Satisfied in Use* state.

Due to the *Statement of Work* predefined by the professors, they do not learn how the Opportunity can be *Identified*, *Solution Needed*, *Value Established* and *Viable*. The Opportunity state of *Benefit Accrued* is also unachievable. Predefined *Statement of Work* keep off the student experience the *Conceived*, *Bounded*, and *Coherent* Requirements alpha states. In a similar way, the pre-selection of MVC architectural pattern prevents students to learn how to lead Software System alpha to *Architecture Selected* state. The course time boundary cut off the possibility to learn how the Software System alpha can be *Operational* or *Retired*.

Finally, the Way of Working *Principles and Foundation* are *Established* by the professors by using the course method, but students learn and apply them by following the method in two iterations.

Figure 2 shows the alpha states course coverage. The column cells represent the alpha states in increasing order. The green color is assigned to states reached by students, professor reaches the orange states, and the red ones are out of the course scope.

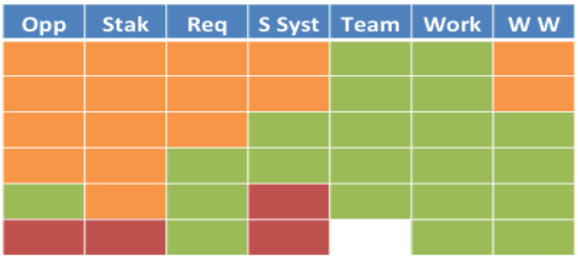


Figure 2. Coverage of alphas states.

4.2 *Activity Spaces and practices*

The Activity Spaces are analyzed for identifying the course practices related to its description. The purpose is to understand the coverage of Activity Spaces by the course practices. In the following text *cursive* letters are used to distinguish the name of Activity Spaces.

- Customer Activity Space: The *Explore Possibilities* is unrelated to any practice because the professors predefine the *Statement of Work* and there is nothing to be explored by the students. The *Understand Stakeholder Needs* is experienced by students during (SDP1) Software Requirements practice. The (SDP5) Software Delivery, (MP5) Closing the Iteration and (MP6) Closing the Project practices help the students to *Ensure Stakeholder Satisfaction* (the professor in this case). The *Use the System* activity space is out of the course scope.

Figure 3 shows the Customer Activity Space coverage. The abbreviations are the first capital letters of the activity space names. The colors interpretation is the same as for Figure 2.

Figure 3. Customer Activity Space coverage.

- Solution Activity Space: The students develop the *Understand the Requirements* activity space during (SDP1) Software Requirements practice, The *Shape the System* they cover in the (SDP2) Software Design practice. The activity spaces *Implement the System* and *Test the System* are related to (SDP3) Software Construction and (SDP4) Software Integration and Testing practices. The *Deploy the System* and *Operate the System* activity space elements are out of the course scope.

Figure 4 shows the Solution Activity Space coverage. The abbreviations are the first capital letters of the activity space names. The colors interpretation is the same as for Figure 2.

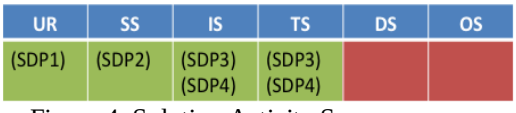


Figure 4. Solution Activity Space coverage

- Endeavour Activity Space: From the teaching point of view, this activity space group is interesting. The professors and assistants during the course *Prepare to do the Work*, *Coordinate Activity*, *Support the Team*, *Track Progress and Stop the Work*, playing the mixture of the consultants, projects managers and Scrum Masters roles.

On the other hand, the students experiment the same activity space by using the method practices. They *Prepare to do the Work* executing (SP1) Forming the Team, (SP2) Defining Team communication and (SP3) Creating a Common Repository practices. They *Coordinate Activity* doing (MP1) Project Planning, (MP2) Iteration Planning and (MP3) Executing the Plan. They learn how to *Support the Team* in (MP3) Executing the Plan practice and (SP4) Iteration Retrospective. The practice (MP4) Assessing and Controlling the Iteration help them *Track Progress* of the project. Finally, the (MP5) Closing the Iteration and (MP6) Closing the Project practices shows them how to *Stop the Work*.

Figure 5 shows the Endeavor Activity Space coverage. The abbreviations are the first capital letters of the activity space names. The colors interpretation is the same as for Figure 2.

	PW	CA	ST	TP	SW
Professor					
Students	(SP1) (SP2) (SP3)	(MP1) (MP2) (MP3)	(MP3) (SP4)	(MP4)	(MP5) (MP6)

Figure 5. Endeavour Activity Space coverage.

In summary, it is difficult for students to learn the Customer *Explore Possibilities* activity space because of the *Statement of Work* pre-definition. Similarly, the *Use the System* activity space is not covered. The *Deploy the System* and *Operate the System* Solution activity space are also out of the course scope.

4.3 *Competencies reached by the course method*

The ESSENCE competencies are analyzed in order to identify their elements likely acquired by the students during the *Software Engineering for Beginners* course. The Stakeholder Representation competency is not analyzed because the students are not involved. In the following text *cursive* letters are used to distinguish the ESSENCE competency names, their descriptions, specific competencies, and skills.

- *Analysis:* This competency encapsulates the ability to understand opportunities and their related stakeholder needs, and transform them into an agreed and consistent set of requirements. During the course, every student—as a team member—executes the (SDP1) Software Requirements practice. It means that every student, starting with *Statement of Work*, collaborates in the identification and agreement on the general use case diagram, and develops for at least one use case, its detailed description, test cases and user interface prototype.

With this experience, repeated in two iterations, we observed that, by the end of the course, the students have the following analytical competencies: *Capture, understand and communicate requirements, Create and agree on specifications and models, Visualize solutions and understand their impact*. But we cannot expect that they know how to: *Identify and understand needs and opportunities* and *Get to know the root causes of the problem.*

The essential skills we teach include: *Requirements capture, Ability to separate the whole into its component parts, Ability to see the whole by looking at what is required, Verbal and written communication, Ability to observe, understand and record details* and *Agreement facilitation*.

- *Development:* This competency encapsulates the ability to design and program effective software systems by following the standards and norms agreed by the team. By means of the execution of (SDP2) Software Design practice, (SDP3) Software Construction and (SDP4) Software Integration, and Testing practices students experiment how to use the selected architectural design pattern (MVC); how to model the static and dynamic solutions to use cases by using class and sequence diagrams; how to model the interface navigation using states diagram; and how to model the database using Entity-Relation diagram. Finally, they code and practice unit tests, experiment code integration and system testing.

After the repetition in two iterations of those activities, they acquire the development competency for: *Design and code software systems* but not for: *Formulate and/or evaluate strategies for choosing an appropriate design pattern or for combining various design patters, Design and leverage technical solutions*, and *Troubleshoot and resolve coding problems.*

The essential skills they improve include: *Knowledge of technology, Programming Knowledge of programming languages*. But we are not sure they reach *Critical thinking*.

- *Testing:* This competency encapsulates the ability to test a system, verifying that it is usable and that it meets the requirements. The (SDP4) Software Integration and Testing practice introduces the students to some basic testing techniques, so we can say that they acquire the testing competencies of: *Test the system, Create the correct tests to efficiently verify the requirements, and Evaluate whether the system meets the requirement*, but not how to *Decide what, when and how to test* and *Find defects and understand the quality of the system produced*.

They do not experiment enough testing for obtaining essential skills such as *Keen observation, Exploratory and destructive thinking, Inquisitive mind* and *Attention to detail*.

- *Leadership:* This competency enables a person to inspire and motivate a group of people for achieving a successful conclusion to their work and meeting their objectives. The Social and Management practices allow for some students, particularly the Team and Collaboration Responsible, developing the leadership competency to: *Inspire people to do their work, Make sure that all team members are effective in their assignments, Make and meet their commitments, Resolve any impediments or issues holding up the team work* and *Interact with stakeholders to shape priorities, report progress and respond to challenges.*
-

Also, they develop essential skills like: *Inspiration, Motivation, Negotiation, Communication* and *Decision making.*

- *Management:* This competency encapsulates the ability to coordinate, plan and track the work done by a team. The Management practices help the students to: *Proactively manage risks, Interact with stakeholders to report progress, Coordinate and plan activities, Account for time* but not for Money spent.

The essential skills developed by students include: *Communication, Administration, Organization and Resource planning* but not *Financial Reporting*.

In summary, the Analysis has shown the students can likely acquire most of the competencies and skills, but in the case of Development and Testing is difficult to reach them in beginner course. The Leadership competency can be developed during the course but it depends on the student personal abilities to reach them. Finally, the Management competency can be learned and applied pretty much during the course except financial aspects.

Figure 6 shows the high level competencies coverage. The green color is assigned to competencies mostly reached by students, the yellow colored competencies are reached partially and the red one is out of the course scope.

Figure 6. Competencies coverage.

We can argue that the level of competencies covered by the course is level 2 (Applies) because our students "Demonstrate a basic understanding of the concepts and can follow instructions" and are "Able to apply the concepts in simple contexts by routinely applying the experience gained so far" —as defined in ESSENCE.

5 DISCUSSION

5.1 *Alphas*

The ESSENCE kernel has seven alphas with 41 states in total. Analyzing Figure 2 we note that 21 states (51%) are covered by the student activities, 16 (39%) are covered by professor/assistants, and 4 (10%) are not covered.

The Team and Work alphas are covered completely by the student activities. This fact means they practice how to organize and manage the collaboration in teams. They do so by following the method defined by professors as Way of Working, exercising the states *In Use*, *In Place*, *Working Well* and *Retired.*

The Requirements states *Acceptable*, *Addressed and Fulfilled* and Software System *Usable* and *Ready* states are reached by the students by following the course method software development practices. In our opinion, these practices are the essence of any software engineering course for beginners.

Additionally, the *Addressed* state of the Opportunity alpha is also reached because at the end of the course students deliver a functional and documented software system in order to be evaluated by the professors. Getting a good grade is the student "opportunity" of this course.

The alpha of Stakeholder is covered mainly by the professor and assistants because they are involved in the course by supporting the student teams and ensure that an acceptable software system is produced.

The Software System alpha is partially covered by the professor because of the decision to impose the architectural pattern MVC, which is used by the students to develop the web application.

The Way of Working alpha states, *Principles* and *Foundation Established*, are also covered by the professors because they defined the content of *Initial Method for Software Development.*

The states of the alphas *Benefit Accrued* of Opportunity, *Satisfied in Use* of Stakeholders, *Operational* and *Retired* of Software System are out of the scope of the course because the software system developed by students will not be put in real life operation.

The students reach many states of the alphas due to the method defined for the course (Way of Working), which includes the necessary basics for software development. Thanks

to this method, students learn the essentials for Requirements and Software System alphas. Also, the course covers the basic activities for teamwork and work management (Team, Work). Hence, we would like to highlight the importance of practice selection and its organization in a coherent method.

5.2 *Activities spaces*

The students experiment the practices related to 73% (11 out of 15) of ESSENCE kernel activity spaces, except 1 (7%) covered by the professor and 3 (20%) out of the course scope (see Figure 4 and Figure 5).

The Costumer *Explore Possibilities* activity space is not experienced by the students because the professors predefine the *Statement of Work* and there is nothing to be explored by them. However, the *Understand Stakeholder Needs* is experienced by students because they have to *Ensure Stakeholder Satisfaction* (the professors in this case) at the end of the course (see Figure 4).

The development practices let the students to *Understand the Requirements, Shape the System, Implement the System* and *Test the System* of the Solution activity space (see Figure 4). In similar way, the social and management practices let the students to experiment all Endeavor activity space elements (see Figure 5).

The execution of *Deploy the System, Operate the System, Use the System* activity spaces are out of the course scope (see Figure 3 and Figure 4) for the reason we have already mentioned in section 4.1.

5.3 *Competencies*

The six ESSENCE kernel competencies were achieved in the following way: two are mostly reached by students, three are partially reached, and one is out of the scope of the course (see Figure 5).

We can argue that the *Analysis* and *Management* competencies are reinforced by several practices, but we are aware that *Development* and *Management* competencies are only reached at very basic level. The evaluation of *Leadership* competency is difficult. We think that several social practices and the role definitions help to reinforce this competency, but we are not sure all students achieve it at the end of the course.

The software Engineering course presented is addressed to beginners therefore the expected level of competencies achieved by the students is 2.

6 FINAL CONCLUSIONS

The objective of the Chapter was to identify the scope of the ESSENCE elements covered by the *Software Engineering for Beginners* course. The relationship between the ESSENCE kernel alphas, state changes in the alphas, activity spaces, competencies and the course elements was analyzed in section 4 based on teaching experience.

The discussion in Section 4 confirms our hypothesis that, even though *Software Engineering for Beginners* course has learning limitations, it allows covering all alphas (but not all their states), activity spaces, and it is possible to reach several competencies until level 2. ESSENCE can be covered in a course for beginners so that students can learn and practice the basics of this discipline and have first experience as practitioners.

ESSENCE is a good tool for software industry for measuring the health of their projects by using the checklists defined as criteria for alphas states. It is assumed that practitioners already know how to get from one state to another. On the contrary, the professors should guide the students on how to achieve healthy states. The tasks outlined in activity spaces are too general to serve as a guide of what to do. For example, there is a gap between *Demonstrable* and *Usable* states of Software System alpha due to missing hints in the activities spaces of how to fill the construction activities.

Nevertheless, we can conclude that ESSENCE can be a useful guide for experienced professors used for designing a software engineering course.

7 REFERENCES

Agile Alliance. "Manifiesto for Agil Software Engineering". Available http://www.agilealliance.org/the-alliance/the-agile-manifesto/ [03/04/2014]

Anderson, D. J. 2010. *Kanban. Successful evolutionary change for your technology business*. Sequim: Blue Hole Press.

Bohem, B. & Turner, R. 2004. *Balancing Agility and Discipline. A guide for the Perplexed*. Boston: Addison Wesley.

Buschmann, F., Meunier, R., Rohnert, H. & Sommerlad, P. & Stal, M. 1996. *Pattern-Oriented Software Architecture*. Volume 1: A System of Patterns. Wiley

CMMI. CMMI Version 1.3 (CMU/SEI-2010-TR-033). Carnegie Mellon University. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661 [03/04/2014]

Humphrey, W. 1999. *Introduction to Team Software Process*.Massachusetts: Addison Wesley.

ISO/IEC 29110. 2011. ISO/IEC TR 29110-5-1-2 Software engineering - Lifecycle profiles for Very Small Entities (VSEs) - Management and Engineering Guide: Generic profile group: Basic profile.

Jacobson, I., Booch, G. & Rumbaugh, J. 1999. *The Unified Software Development Process*. Massachusetts: Addison Wesley.

Jacobson, I., Ng, P., McMahon, P.E & Spence, I. & Lidman, S. 2013. *The ESSENCE of Software Engineering. Applying the SEMAT Kernel*. Indiana: Addison Wesley.

ESSENCE. "Essence—Kernel and Language for Software Engineering." Available: http://www.omg.org/spec/Essence/1.0/Beta1/PDF/ [03/04/2014]

Rumbaugh, J., Jacobson, I. & Booch, G. 1998. *The Unified Modeling Language*. Reference Manual. Addison Wesley.

Schwaber, K. S. 2011. "The Scrum Guide—The Definitive Guide to Scrum: The Rules of the Game". Available: http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011.pdf

SEMAT. "Call for action". Available: http://semat.org/?page_id=2 [03/04/2014]

SWEBOK. "Guide to the Software Engineering Body of Knowledge v3.0". IEEE Computer Society. Available: http://www.computer.org/portal/web/swebok/v3guide [03/04/2014]

This page intentionally left blank

Chapter #11

On the use of the SEMAT kernel within a software engineering course

D. F. Cifuentes Gil
Universidad Nacional de Colombia, Bogotá, Colombia

J. S. Hernández Serrato
Universidad Nacional de Colombia, Bogotá, Colombia

J. H. Aponte Melo
Universidad Nacional de Colombia, Bogotá, Colombia

1 INTRODUCTION

How to teach software engineering has been an endless concern for lecturers, practitioners, and researchers in the field of software engineering. As a result, there are nowadays several international peer-reviewed conferences and workshops in which researchers, educators, and industry trainers around the world have been publishing their research for several decades. The covered topics within these venues range from technical subjects to social and cultural issues that often arise during team work and project management. Apart from that, there have been several initiatives led by guilds such as IEEE and ACM that have served to improve and standardize the topics that should be taught to future software engineers. For instance, the SWEBOK Guide, created by means of cooperation among several professional bodies and members of industry and published by the IEEE Computer Society, has played a decisive role in shaping the software engineering curriculums and the content, emphasis, and didactic strategies of training programs around the world.

Over the years, as part of a continuous evolution process, the practice of software engineering in industrial contexts and the way it is taught at universities and institutes have coevolved and have influenced one another. On the one hand, academic institutions have been responsible for preparing new cohorts of software engineers who will eventually lead the software industry; and simultaneously, supporting research groups provide insights into the most complex and controversial issues in the area. On the other hand, from the large group of software practitioners have emerged industry leaders who have provided academia with practical knowledge on how to improve the way we build and maintain software systems and deal with the challenges of our ever changing field.

One of the most influential industry contributions to the field of software engineering is what nowadays is known as the Agile Movement (http://agilemanifesto.org/). Although agile methods started to be really important since mid-1990s, their practices, principles and values have been around long time ago (Larman, 2003). Within the old ideas reused by agile methods, it is worth to mention the iterative and incremental development, used since 1950s in NASA and the IBM Federal Systems Divisions; the crucial role of user feedback, that years later became the core of agile principles and values; the frequent delivery, that encourages the developers to complete analysis, design and test in each step; and the response to change, which clearly stated that software development is an adaptive process rather than a predictive one, and therefore, software teams must deal with changes in requirements, technology, and even, modifications in the agile method itself.

Although several studies have reported improvements in productivity, quality, and customer satisfaction by using agile methods, there have been also identified some deficiencies and space for improvements. In 2009, the SEMAT initiative was launched aimed at reshaping software engineering in such a way that it qualifies as a rigorous discipline (http://semat.org/). As agile methods, SEMAT promotes iterative and incremental development, self-organized teams, and the same philosophical principles. However, SEMAT is not one more agile method and does not come to replace or compete with existing methods. Conversely, the current SEMAT elements provide developers with a new way of looking at the domain of software engineering in which is feasible for software teams to perceive and track the progress and health of development efforts, to combine agile practices according to the particular circumstances of each project, and to use a common reference model and language to talk and improve their ways of working. Thus, as Jacobson, Spence and Ng say, *"combining agile and SEMAT yields more advantages than either one alone"* (Jacobson *et al.* 2013).

At Universidad Nacional, Bogotá campus, the teaching of software engineering for bachelor students is concentrated in two consecutive courses, one pre-requisite of the other: software engineering I and software engineering II. The first course covers several topics, including the software engineering fundamentals, a brief introduction to project management, object-oriented design using UML, a brief review of agile

methods along with frameworks for process assessment and improvement such as CMM, SPICE, and Cobit. The second course is mostly a hands-on experience, and therefore, is focused on the development of a software system from scratch, usually a web application, by following an agile method chosen by the software team. This Chapter describes a preliminary experience using SEMAT in combination with the agile practices that traditionally have been used by the students in this second course during the last years. Specifically, the Chapter discusses several issues including how to teach the SEMAT essentials, how to organize the teams, which tools could be used for supporting the diverse activities of the software project, and how to monitor and assess the work of the students.

The Chapter is organized as follows. In Section 2 we present the work related to this Chapter. In Section 3 we describe how agile software development is taught and practiced within a software engineering course at Universidad Nacional de Colombia, and also, how the SEMAT[1] kernel can be introduced and coupled with the existing topics. In Section 4 we describe two software projects that were carried out under the context and methodology described in the previous section. In Section 5 we summarize the student feedback about the course and the use of SEMAT in academic contexts. Finally, in Section 6 we give conclusions and sums up future work.

2 RELATED WORK

As was mentioned above, there are a number of publications regarding software engineering education. This is a clear indication that teaching software engineering, like the practice of software engineering, suffers from the lack of a general framework and a common language that allow for lecturers and researchers to work together and compare their pedagogical alternatives. In this Section we restrict ourselves to the most relevant work that has used agile principles and hands-on approaches for teaching software engineering, and mostly, to pedagogical proposals that combine agile methods with the SEMAT kernel and language.

Since the early 90s, several researchers and lecturers have reported experiences with project-based approaches for teaching software engineering. For instance, Oudshoorn and Maciunas (1994) describe an experience where the students are required to build a software application to satisfy the requirements defined by the lecturing staff. Although the projects were carried out by following a traditional process model, the authors highlight the success of the course in terms of educative outcomes and popularity among the students. Pierce (1992) points out the benefits of assigning maintenance exercises in a project-based course in software engineering. In a more recent work, Gnatz *et al.* (2003) argue that teaching software engineering requires covering the technical skills and providing students with the opportunity to deal with typical non-technical issues such as working in a team, conflict resolution, organizing the division of work, tracking activities, etc.

In the same vein, Budd & Ellis (2008) argue that is far more effective to involve student teams directly in a project than the approach where case studies of existing systems are used to study the methods applied to develop these systems. Razmov (2007) presents a package of pedagogical practices that supports the learning goals in a project-based software engineering course and allows for instructors to implement a continuous improvement of the course. As an even more recent experience, we can mention Bavota *et al.* (2012) who present an approach for teaching simultaneously two project-based courses, namely *software engineering* and *software project management*, in which students of both courses are mixed to form the teams.

The adoption of the agile philosophy within software engineering courses has been reported in several papers. It is worth mentioning the work reported by Alfonso and Botia (2005) in which they explain how an agile process serves as the backbone for teaching the course and for the incremental learning of both technical and managerial matters that arise in a typical software project. Muller and Tichy (2001) and Shukla and Williams (2002) present classroom experiences where they analyze, assess, and integrate extreme programming practices into a software engineering course. As a more general example, Rajlich (2013) explains how iterative software development is taught at Wayne State University, and highlights the fact that the students learn the skills they require to work as developers on software projects.

Regarding the combination of SEMAT and agile processes, our work is completely aligned with the ideas presented by Jacobson et al. (2013). They point out the similarities between the philosophies promoted by these initiatives and explain what SEMAT adds to agile methods and *vice versa*. Thus, our work can be described as a preliminary effort for mixing agile principles, values, and practices with the kernel and language provided by the SEMAT initiative, in an academic context. In this regard, it is noteworthy that Ng and Huang (2013) present the preliminary feedback given by seven Chinese universities about the challenges software professionals face and how the SEMAT kernel and language may help them to provide students with the fundamentals and cognitive tools required to learn and understand the diversity of software industry, and also, to overcome those future challenges.

There are recent teaching experiences of the SEMAT kernel in universities and companies as was mentioned by Kajko-Mattsson *et al.* (2012). In this regard, the SEMAT website keeps an up to date list of publications. Within this set of con-

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

tributions, we consider highly relevant the proposal published by Zapata and Jacobson (2014) where they present a curriculum for teaching a SEMAT course. By contrast, this Chapter discusses the integration of the SEMAT kernel to an existing project-based undergraduate course of software engineering that teaches further topics related to agile methodologies.

3 COMBINING AGILE METHODS AND SEMAT

This section explains how agile software development is taught and practiced within the second course of software engineering at Universidad Nacional de Colombia, in Bogota. Moreover, we discuss the particularities of the academic context in which the students carry out the software projects. Lastly, we describe how the SEMAT kernel and language were introduced and coupled with the existing subjects, and how they were used by the teams and the lecturer.

3.1 *Agile projects within a software engineering course*

Traditionally, the second software engineering course begins explaining software life cycle models, with focus on agile methods. Specifically, during the first three weeks (12 classroom hours) the course covers the values and principles promoted by the agile manifesto, an overview of the agile methods defined so far, and a detailed description of Extreme Programming and Scrum as these are two of the most popular methods.

During these three initial weeks other important things happen in parallel with the lectures. First, the students by themselves form teams of 4-to-6 people. The lecturer intervenes only in those cases in which a student fails to join a group or when it is necessary to join up two small groups. The teams are encouraged to discuss and make decisions about managerial aspects of the endeavor, such as deciding who the team leader is, the frequency and type of meetings they would attend, and establishing the set of values and principles they are willing to adopt for their software endeavor. Second, the students hold brainstorming sessions in which they explore possible software projects they could develop during the semester.

The selected project is often a system in which the team is really interested, and therefore, the motivation is usually high along the entire project lifecycle. Sometimes the project addresses real needs of a private company or a public institution. In any case, by the second week of classes every group must have an approved project to work on. Lastly, each team assesses the resources they have and the risks of the project by appraising the available time per week of each member, hardware and software assets, and knowledge and programming skills of each person; as well as the skills, knowledge, and technology required for developing the system.

As a deliverable of this starting phase, each team has to hand in a document presenting the problem to solve, the solution proposed, the team and its resources, the technology required, and the agile method to be used. The software method should be adapted to the specific circumstances of the academic context where the software endeavor takes place. For instance, adopting extreme programming may require modifying or ignoring some practices such as *the 40-hour of work per week* and *the client is here*. These adaptive changes of the agile method are due to the circumstances described in the next section.

Once the teams have been organized and the projects have started, the rest of the course focuses on the technical aspects in order to support the development effort. Thus, the main subjects covered are control version systems, frameworks for web development, design patterns, testing, and software quality.

In addition, there are three formal presentations of all the projects at weeks 7, 11, and 16. The presentations show the work performed so far, in chronological order. One concrete way of doing that is showing the planning of each of the iterations performed, and also, describing the software artefacts resulting from this work: UML diagrams, user interfaces, test cases, etc. Each presentation ends with the execution of the latest stable version of the system in order to show the functionality effectively implemented so far. Besides that, at the end of the semester all groups participate in an open exhibition where they have the chance of presenting their project to the general public, and the university community in particular.

3.2 *The particularities of the academic context*

When comparing an academic project with a true software project, the first circumstance to consider is that the students are not competent developers yet and they are immersed in a learning process where the risks and pressures of a real project are usually not present. Moreover, this academic environment should allow for them to study the technical subjects required by the project, learn how to build software, and practice the learned subjects to develop the skills required by the software endeavor.

There is no client. Although the lecturer and a teaching assistant occasionally act as clients, the absence of a real client limits the practice of several important agile practices in which the role of the client is crucial in guiding the project and deciding which the most important requirements are.

The time devoted to the project by the students and the schedule of their meetings are also variables that depend on multiple factors like the number of team members, the number of courses in what they are enrolled, and their individual class schedules. In our course, the teams are between 4 and 6 members, each person devotes between 4 and 8 hours per week to the project, the teams conduct at list one face-to-face

meeting a week, supplemented with several types of computer-mediated communications.

Thus, the academic context hinders or even prevents the teams to put into practice some agile principles related to high client involvement, fixed weekly working time on the project, daily scrum meetings, reaching a pace at which the development team can comfortably work for the entire project life span, and maintaining collocated teams to improve coordination among the members.

Finally, for the lecturer is difficult to monitor and appraise the progress of each individual project. Each semester there are 7 or 8 projects that cover a wide variety of domains, use different technologies, and implement dissimilar practices and team dynamics. For instance, students have used several frameworks for developing a number of web applications, several game engines for making action, strategy and casino games, APIs for developing mobile applications, mostly for the Android platform, and the SDK for Kinect applications.

3.3 *Introducing and using SEMAT*

The first change we made to our traditional approach was the introduction of the SEMAT kernel and language. We use the first 4 weeks of classes to teach agile and SEMAT by using the schedule shown in Table 1. This plan includes slide presentations, readings, questionnaires, as well as the manufacturing of a set of physical kernel cards. The readings are the first and second parts of the Essence book, and also, some of the resources available at the SEMAT website.

Table 1. Schedule for introducing agile and the kernel

Week	Subject
1	Agile principles; overview of Scrum and XP The SEMAT initiative
2	The Kernel and its alphas
3	Practices and kernel benefits
4	Planning, doing, and checking with the kernel Running an iteration

Apart from that, the structure of the document that each team has to hand in describing the project was changed according to the SEMAT alphas. Thus, the document has to include the following sections:

Opportunity alpha. Describing the circumstances that make it feasible and desirable to construct the system you are proposing. Suppose that your arguments should be convincing enough for a group of potential sponsors.

Stakeholders alpha. Listing the people, groups, or organizations that would be affected either positively or negatively by the system you plan to implement. Likewise, students should list the persons, groups or organizations that could affect the project. In each case, they explain how that impact would be.

Requirements. Listing the major real needs that the system should satisfy.

Team. Making a table to describe each team member with the following columns: (i) name, (ii) hours per week dedicated to the project, (iii) technical features of the computer she is going to use for working on the project, (iv) technical skills such as developing tools, programming languages, methodologies, prior experience in software development, etc. In addition, specify who the coach (leader) is.

Way of working.

- Defining the list of values and principles that will guide the development of the project.
- Defining the agile practices adopted for the project.
- Defining the set development, administration, and communication tools that will be used during the project.
- Defining the set of core artifacts built during the project (*e.g.,* user stories, UML diagrams, product backlog, sprint backlog, etc.)

Since at that point the team has not started the system construction, the *System alpha* is not included in this project definition document.

The three formal presentations of all the projects mentioned above were also modified according to the kernel alphas. Thus, each team should show the iterations performed so far, in chronological order, and the status of the project within each of the seven dimensions or alphas. The impact of using SEMAT for monitoring progress and health of the projects is remarkable. The SEMAT alphas allow for the lecturer and the students to track and assess the progress of each project, identify issues that hinder state transitions, and also, compare the ways of working of the teams. In other words, SEMAT provides a common setting in which all projects can be displayed and compared at the same time.

4 TWO PROJECTS THAT USED SEMAT

In this Section we present the first two projects carried out under the context and methodology described in the previous Section. The students used practices and artifacts from the Scrum method and SEMAT as a tool for measuring progress and planning the work. Both teams were enrolled in a course called *Advanced Topics in Software Engineering*, where the principal topics treated were SEMAT, agile methods, and FLOSS.

The first part of the course takes into account two parallel tasks: learn about how to use and implement the SEMAT kernel, and secondly, develop a software project using agile practices and the kernel. The first and second authors of this chapter were the leaders of these two teams.

The main goal of both projects was to create a functional and usable version of two web applications. Figure 1 shows the skinny desirable system on which the initial planning of the projects would be based.

4.1 *Hi-Q SEMAT Experience Report*

4.1.1 *Description of the project*

Hi-Q or Peg Solitaire is a board game for one player involving movement of pegs on a board. In its standard version, the board has 33 holes arranged in the shape of a Greek cross. The left board in Figure 2 shows the initial state of the game in which there are 32 pegs occupying all holes, except the one at the center of the board. The only allowed move is jumping a peg X over any one of its immediate neighbors, let's say Y, as long as the target is an unoccupied hole. At the same time the peg Y is removed from the board. The board on the right side of Figure 2 is the result of an opening movement. The pegs can be moved only horizontally or vertically, and the goal is to find a sequence of moves (jumps) that leaves only one peg at the center of the board.

The initial requirements for an online version of this game include: allow for the user to play Hi-Q with the standard board of 32 pegs (see Figure 2), count movements, undo and redo actions, change the initial configuration of pegs (see Figure 3), and the use of a timer.

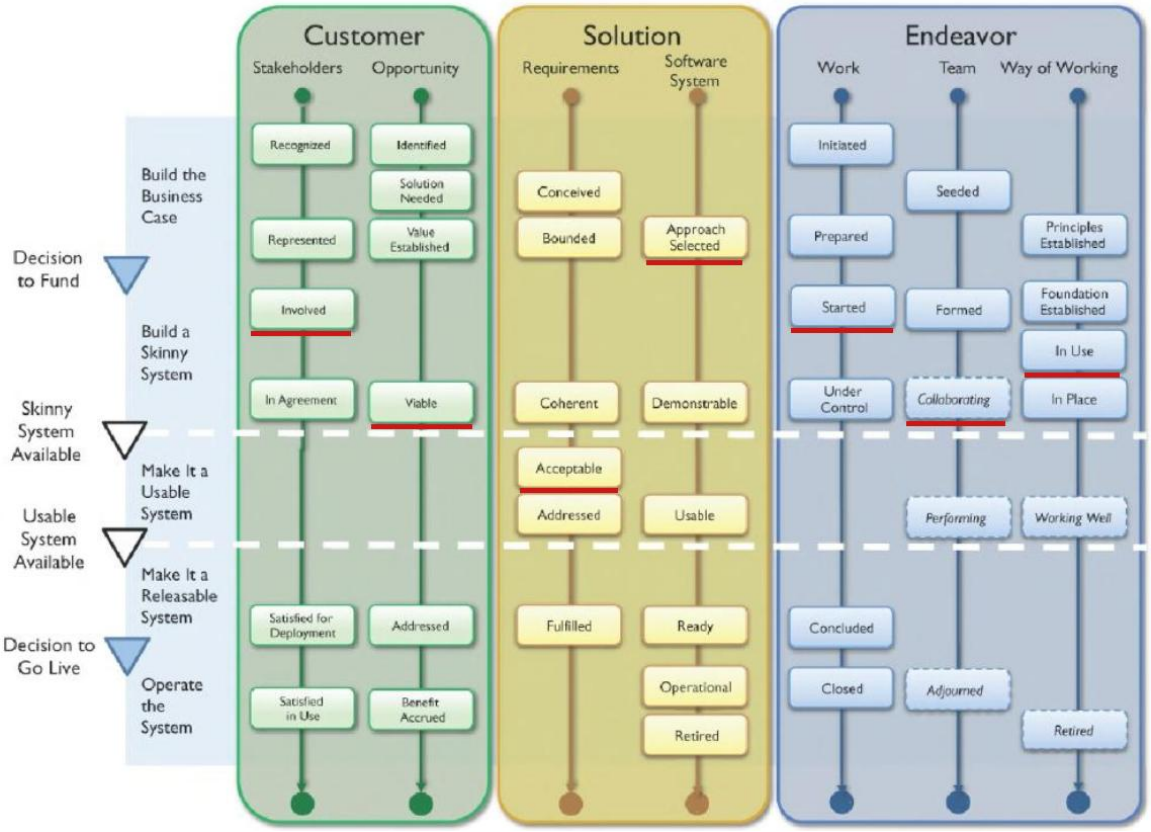


Figure 1. The skinny system

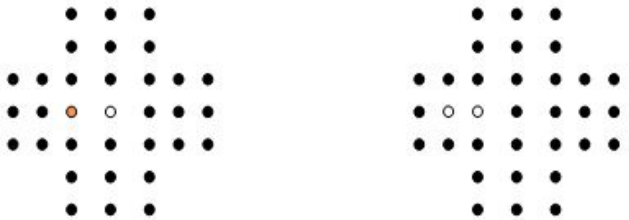


Figure 2. An opening movement

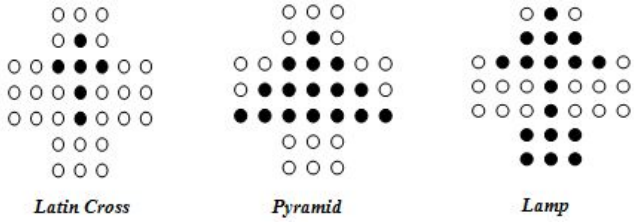


Figure 3. Other initial configurations of pegs

4.1.2 *Tools*

The team decided to use several tools for supporting some usual technical and managerial tasks of a software project. First, a Facebook group and short meetings every two weeks were the communication media used by the team. Second, for controlling each member tasks, the team used the virtual canvas provided by Mural.ly, where each card can be posted, in order to represent every alpha state (see Figure 4). Lastly, the team hired Google services to track the software evolution, including Google Docs for storing documents, Google Code project hosting service, and also, Google App Engine was used to deploy the application.

For software development they used Java and JavaScript, with the Netbeans IDE. Java was used for the business entities logic and JavaScript for controlling the web user interface and consuming the methods exposed by the business logic.

4.1.3 *Agile Practices adopted*

While the team was learning what SEMAT is and how to use it, as a group they decided which practices would be used, based on what they found on the literature. Among the main practices adopted they highlight the use of short iteration cycles, incremental development, pair programming, and working software as measure of progress from XP; prototype development from the prototype development model; the use of a product backlog, having a couch, talking to stakeholders for continuous feedback, having a short meeting every class day, doing a sprint planning each two weeks from Scrum; and finally, tracking the progress of the team by measuring an estimated time and spent time for each specific task done.

Figure 4. Mural.ly preview for state cards

4.1.4 *Iterations*

Since the available time was not enough to build a complete system and the main goal was to experiment the Agile/SEMAT combination, they decided to perform only three iterations. In this Section we summarize the three iterations and we point out their salient points.

4.1.4.1 Iteration 1

In this initial iteration the group had a hard work mainly because, at that moment, there was no previous experience working as a team and using the kernel. Thus, the first task was to know each other. Two of team the members are system and computing engineering students, focused on software engineering, who had worked together in several software projects. The other member is a computer science bachelor who came from Germany and has no too much experience with the development of this kind of projects.

As a second task, the team focused on the project status to determine where they are, and where they want to go. Thus, the team used the kernel to agree on the current state of the project. Using the planning poker approach, each alpha and each state were reviewed by the team members, and every member gave arguments about why a state were reached or not, until achieving an agreement.

As a result, the team got the target states for this iteration: *Way of working: Foundation Established*, *Team: Performing*, *Work: Started*, and *Software system: Demonstrable*. For reaching those states, the team created several assignments for each team member. Unfortunately the team could not finish all the tasks proposed for this iteration, due to the lack of experience for estimating the time required by each task. Figure 6 shows the final state for this iteration.

The artifacts produced were a document that describes the way of working of the team, a group of classes such as *PlayField* as main entity class and *PlayFieldSolver* for business logic of the game, and also, an HTML page with the first version of the user interface.

4.1.4.2 Iteration 2

With the first iteration as experience using the kernel, the team had a better idea of how to guide its effort and estimate the tasks required for reaching a state. Accordingly, the target states proposed for the second iteration were: *Stakeholder: Involved, Way of working: In use, Work: Under control*, and *Software system: Demonstrable*.

This working cycle is particular since it includes a target state that should be reached in more than one iteration. The state mentioned is *Software System: Demonstrable*. Having an unreachable state, did not mean the tasks for the iteration would not be finished. For that reason, there are only a few tasks listed that could be finished within the iteration.

After identifying the target states and their characteristics, the team began their work. The team noticed that they need to measure and track the progress, especially who the owner of a task is and how much time is going to spend in order to fulfill the labor. For that reason, the team decided to add new information at the end of the task description which includes the estimated time, the task owner, and the real spent time. (see Figure 5).

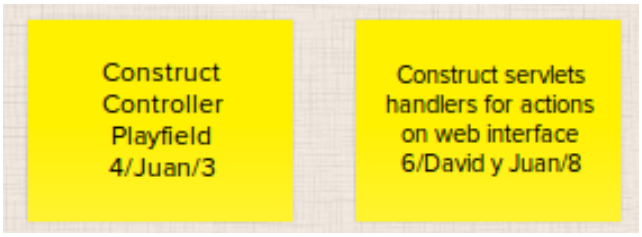


Figure 5. Tasks with tracking information

Finally, the group closed the iteration without any kind of problem. At the end of this work cycle, a new version of the software system was released which contains an enhanced user interface and a lot of new features, including playing the game and the undo/redo actions.

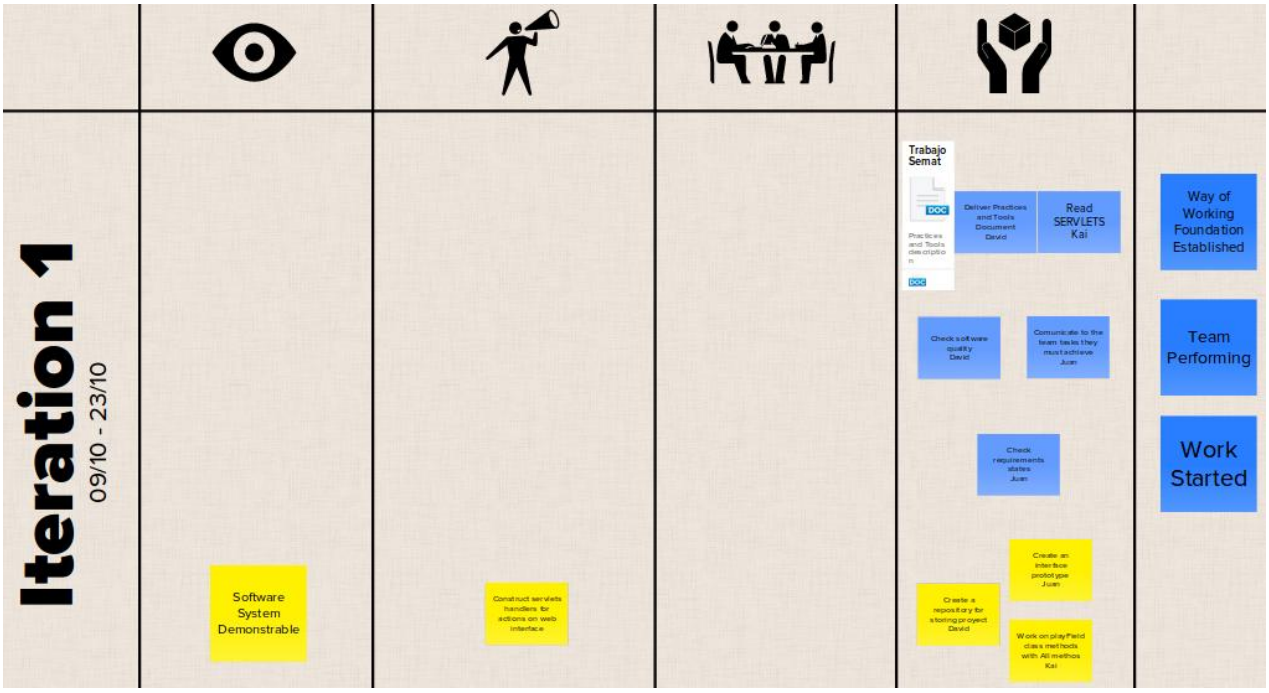


Figure 6. Iteration 1 at its final state

4.1.4.3 Iteration 3

By a suggestion of the lecturer, and after assessing the current state of the project, the team agreed to reach the *skinny system* milestone, by the end of the third iteration. Hence, the states the project must reach are: *Software System: Demonstrable, Way of working: In Place, and Stakeholder: In Agreement*. This goal resulted in a bunch of tasks to be done, so that this iteration was the most challenging for the team.

The tasks assignment was in such a way that each member was in charge of the task most suitable for their knowledge and skills. Thereby, one of the members focused on business logic, an automatic solver, and several entity attributes with its methods; the second member focused on exposing the

methods created by the first one, as services for being consumed from the interface; and the third member focused on enhancing the user interface and adding features based on the methods already exposed. At the end of this iteration, the target was reached and a usable version of the game was deployed (see Figures 7 and 8).

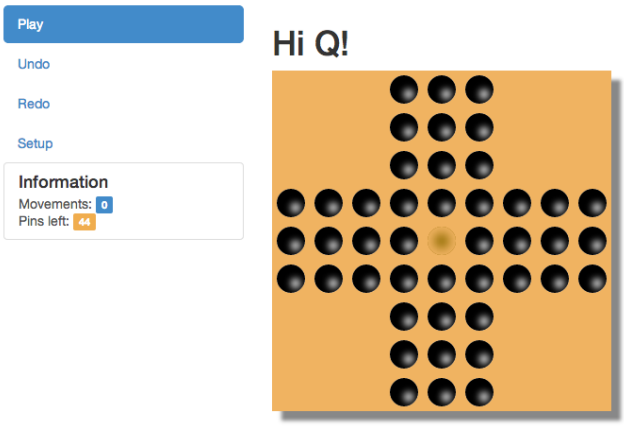


Figure 7. Final interface with new features

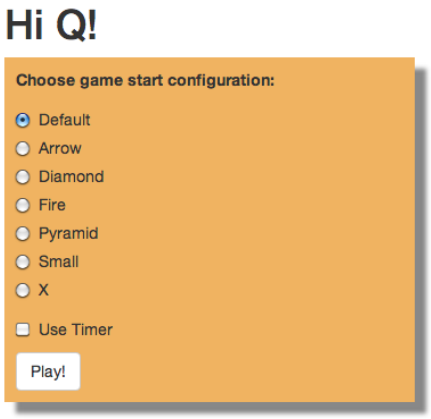


Figure 8. Setup menu

4.1.5 *Difficulties*

During this project, the team had two drawbacks which have a considerable impact on each iteration. Firstly, when planning an iteration, it was often difficult to decide whether an state would be reached by doing the corresponding tasks. The reason was that some items of the checklists were ambiguous for some participants of the meeting. Secondly, although the canvas with the committed tasks is very useful, it loses its value as project management tool when a task is created and there is no automatic way to notify the member who was assigned to that task.

4.1.6 *Previous experiences*

The team members had very different experiences with software development processes and the way of working. For this reason, the way of working experimented several changes from the beginning of the project.

Two of the members had developed several web applications, so that they were already familiar with JavaScript, Java, and the MVC pattern. Additionally, they had worked with agile methodologies, namely, Scrum and XP. The third member just had a strong experience creating java components, and he did not know how to expose the methods for a web based application. Besides, he did not have any experience with agile process, and therefore, he learned this kind of ways of working during the project lifecycle.

4.2 *SEMAT assistant experience report*

4.2.1 *Description of the project*

The SEMAT assistant would be a web application based on project management for software development with the SEMAT kernel. The tool will support the software teams for the development of their projects, and will allow for them to manage their requirements, tasks, and project status over time.

The idea of creating this tool arises due to the lack of applications on the market for supporting the SEMAT initiative. The tool will be implemented based on the SEMAT kernel, and will manage the Dashboard, the Task Board. Besides, it should allow users to define iterations and establish practices for the team. Its main goal is to facilitate and streamline the development of and agile software projects that uses the SEMAT kernel.

4.2.2 *Tools*

The team selected the following tools for developing the SEMAT assistant:

- Java: Java was chosen because the visual part can be easily achieved with a framework.
- Eclipse IDE: As integrated development environment Eclipse was chosen since it is lightweight and versatile.
- SQLite: A portable database.
- Maven: for packaging the project and managing the libraries.
- MercurialHg: For managing versions.
- Tomcat: As web server.
- PrimeFaces: For the managing the components of the web interface.
- JPA: For communication with the database.
- Software for work coordination:
    - Google docs to manage and share documents.
    - Skype or Google Hangout to coordinate meetings.
    - Mura.ly to manipulate the set of SEMAT cards, and also, to control the progress of each iteration

4.2.3 *Agile Practices adopted*

Based on the previous experiences of some members of the team the following practices were adopted:

- The development methodology is based on the kernel and adding some of the Scrum practices.
- Unit testing. The framework chosen for unit testing is JUnit. The unit tests will be performed by the developer when new functionality is developed.
- Continuous integration by using Jenkins.
- Iterative development. Iterations will last between two and three weeks.

4.2.4 *Iterations*

In this Section describes the iterations of the project in which we applied concepts and practices of the SEMAT initiative for the development of this web application. The development of the tool was carried out on one initial phase and four iterative cycles. For each iteration, they present its goals and describe the tasks, as well as an overview of the progress of the project and the team organization for achieving the proposed goals.

4.2.4.1 Definition initial state

First of all, the four team members determined the initial state of the project based on the planning poker approach. After discussing about each alpha, they agreed on the initial state of the project as: *Opportunity: Identified, Stakeholders: Recognized, Requirements: Conceived,* and *Team: Seeded*.

4.2.4.2 Iteration 1

The timebox was set for 3 weeks and the targets selected were: *Opportunity: Solution needed, Stakeholders: Represented, Requirements: Bounded, Software system: Architecture selected, Team: Formed, Work: Initiated,* and *Way of working: Principles established*.

They agreed on having a weekly meeting to review project progress and feedback. The main tasks performed were the definition of the architecture, setting up development environment, definition of the responsibilities of each team member, and training the team members on the tools to be used.

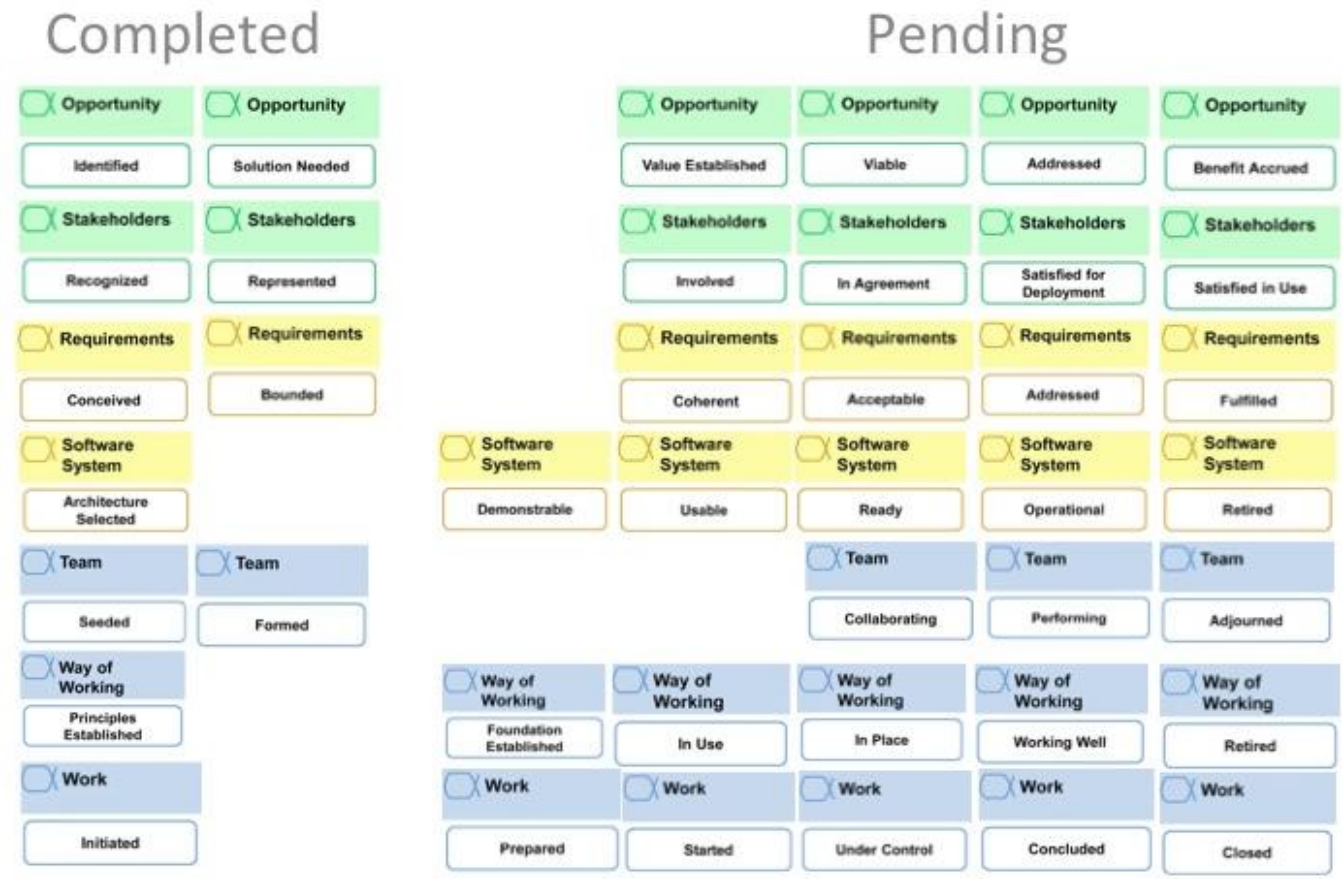


Figure 9. Final state of the iteration 1

Iteration was successfully completed and all planned tasks were performed. As retrospective outcomes, it was suggested to change the version control system (Mercurial) and reinforce skills in Java for the members that are still not familiar

with the language. Figure 9 shows the final state of the iteration.

At this stage, it was very important to gain the knowledge required to balance the development skills of the team members. The main purpose of choosing the targets was to clarify what is going to be developed and the scope of the project.

As for gathering information, constant communication was maintained during those two initial weeks of the timebox with the stakeholder. At the end of the iteration it was considered that the goals were reached.

4.2.4.3 Iteration 2

The timebox was set for 2 weeks and the targets selected were: *Opportunity: Value Established, Stakeholders: Involved, Requirements: Coherent, Software system: Architecture selected, Team: Collaborating,* and *Way of working: Foundation Established.*

The main tasks performed in this iteration were establishing and estimating the expected impact with the use of the tool in academic and business environments, the definition of the advantages provided by the SEMAT kernel compared to other agile development methodologies, the definition of the communication channels with the stakeholder, establishing the stages when using the application (e.g., setting the initial state of a project, checking the lists of alphas and others), the definition of standards for the development of software modules, the definition of the communication channels between team members (tools, schedule, repositories, etc.), and the integration of tools in the development environment.

We completed most of the tasks planned. Although certain dependencies between tasks prevented a complete success, the results of this stage were evaluated as positive. It was achieved the definition of the skinny system and much of the way of working was coordinated. Additionally, it was successfully configured the development environment of the initial version. The defined scenarios gave an idea of the deliverables to be built, as well as the possible inputs, outputs, and conditions that must be taken into account when using the application.

Some of the activities performed in this iteration were not specifically identified at the beginning but were discussed through team meetings and stakeholder interventions.

4.2.4.4 Iteration 3

The targets selected for third iteration were: *Opportunity: Viable, Requirements: Acceptable,* and *Way of working: In use*. The main tasks performed in this iteration were the validation of the defined requirements, the definition of the contingency plans, assigning priorities to the proposed scenarios and use cases already defined, making sure that the proposed solution provides real value for the customer, and gathering feedback about the tools used and the way of working, in order to be used by the team members.

Other tasks related to the presentation, documentation, design and development were designing an interface for the dashboard with the *Pencil* tool, designing and creating the DB model, populating the database with static information (alphas, states, and checklists), and integrating queries for the database with the *primeFaces* project.

The tasks were performed satisfactorily, although there were a lot of dependencies among them. The results of this stage were difficult to assess since the tasks were focused primarily on the design and development stages. Thus, there was no stable version of the system that allows for the team to show concrete progress.

4.2.4.5 Iteration 4

The timebox set was 3 weeks and the targets selected were: *Software system: Demonstrable* and *Work: Started*. The main goal was to complete functionality of the application to deliver a skinny system. The functionalities to develop were the integration of the dashboard, managing the alphas and their different states, the display of the status of a checklist when selected an alpha, including graphs of alphas, saving the alpha status, moving the alpha when all their checklist are checked, saving the current project status, retrieving the status of the project (alphas, states, and checklists), and loading the positions of the alphas on the dashboard.

Figure 10 shows one of the checklists viewed from the dashboard.

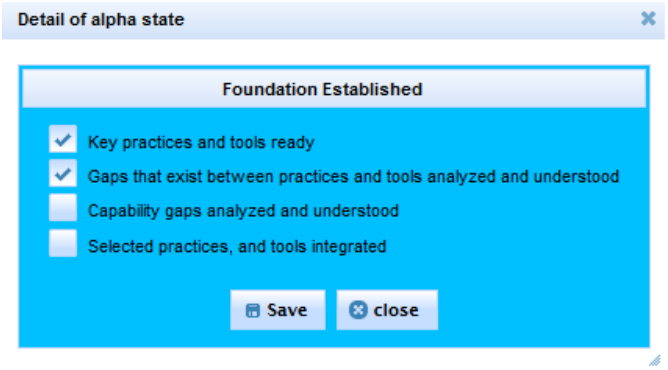


Figure 10. Checklist for an alpha state

It was considered that the goals of the iteration were reached, despite of the difficulties encountered. These difficulties were caused mainly by the accumulation of tasks. The main goal of achieving the skinny system was successfully completed according to the definition of that system.

Figure 11 shows the status board of the tool and Figure 12 shows the final state of the fourth iteration.

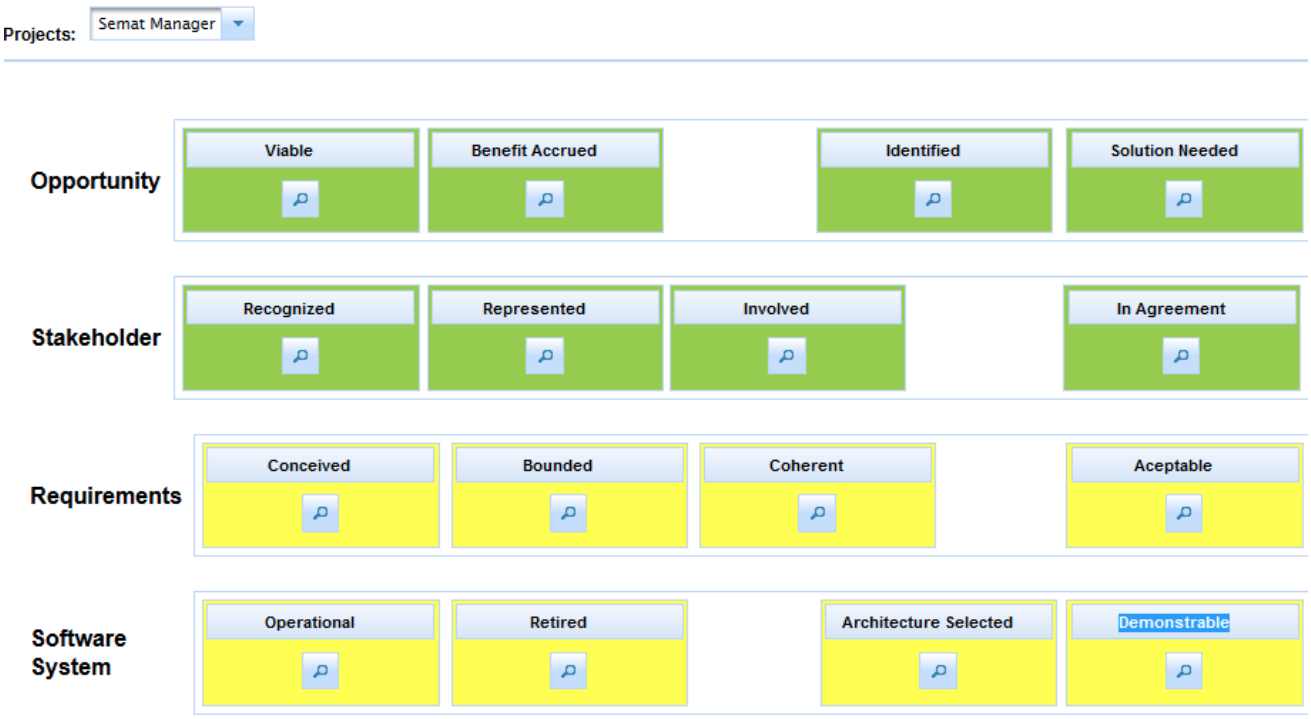


Figure 11. Status board of the tool

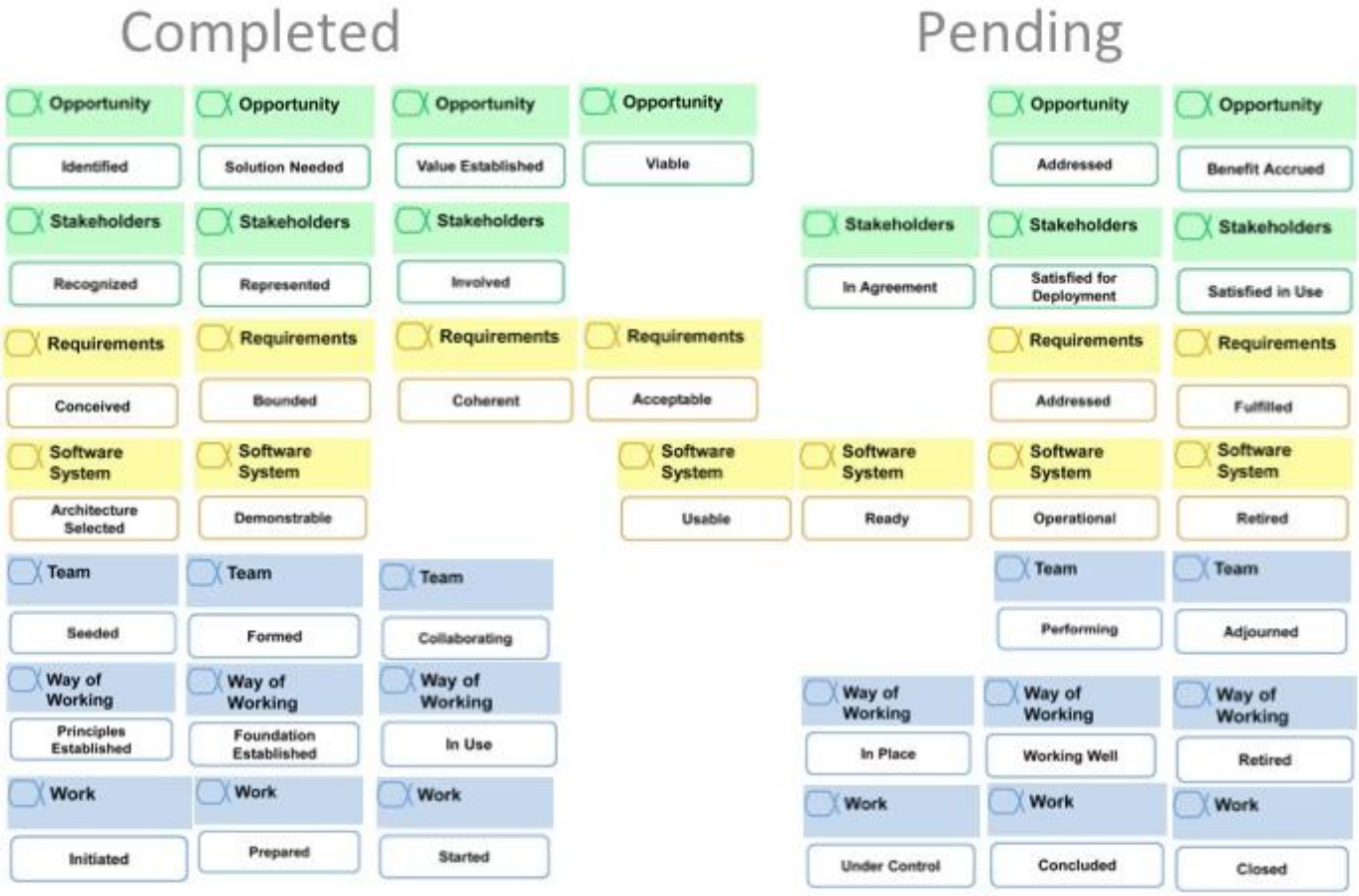


Figure 12. Final state of the fourth iteration

4.2.5 *Difficulties*

The adoption of practices and artifacts was initially difficult because not all members had experience in agile methodologies. For this reason, the initial planning of the project lasted more than expected. Having some kind of default set of artifacts and practices within the SEMAT framework would be useful to perform this initial stage of the project.

At the beginning of the project the team had problems deciding when a particular state was reached due to the lack of full checklists for all the alpha states.

Finally, it is important to emphasize in the gathering of requirements, that you should not always associate a new artifact for each of the requirements of the checklist of alpha states. This was one of the main difficulties in deciding whether an alpha state was fulfilled.

4.2.6 *Previous experiences*

Two team members had extensive knowledge of Scrum and so they were the guide for iteration planning and the definition of the way of working. The other two members only had theoretical knowledge of the agile methodologies. None of them had previous experience with SEMAT.

As soon as the team learned the fundamentals of the SEMAT kernel, the members who knew more thoroughly agile methods realized that SEMAT serves to manage the project, manage their resources, and to identify the status of the project at any time. Despite this, it allows for the team to choose the specific practices and deliverable artifacts.

5 THE STUDENT STANDPOINT

At the end of the projects, in order to gather the opinions and thoughts about SEMAT, the students were asked to answer the three questions below.

- From your own viewpoint, what are the main advantages of SEMAT?
- Would you use SEMAT for a real software endeavor?
- What are the main drawbacks or deficiencies, if any, of the SEMAT proposal? Any ideas for improving the SEMAT approach?

The following sections summarize the responses collected.

5.1 *Main advantages of SEMAT*

According to the student point of view, there are two salient features of SEMAT. Firstly, SEMAT allows developers to identify and show the current state of a software project. The reason is that the kernel induces the team to think about different perspectives or dimensions of the project by always measuring project process in alpha states. Simultaneously, the set of cards provides the team and stakeholders with a novel and understandable way to represent the status of the project that visually conveys a lot of information about the software endeavor. Secondly, the alphas and associated checklists guide the team throughout the life cycle of the project regardless of their nature, and being compatible with any agile methodology used by the team. Thus, the SEMAT kernel offers developers a framework that allows them to understand and control both static and dynamic aspects of the software project.

Another couple of characteristics highlighted by students are flexibility and extensibility. SEMAT is flexible in the sense that it is adaptable to any way of working of the team and the features and technological setting of the project. Extensibility means that the team can add alphas and their states, practices, and activities according to specific needs of a software endeavor. In this regard, one student noted: "*I would definitely use the SEMAT initiative in developing real software projects because it offers advantages not offered by any of the current and traditional methodologies or fads; also, it offers flexibility in adapting ways of working and practices depending on software teams and projects features.*"

One of the students quoted "*I think because of the simplicity of the kernel it will help computer scientists and stakeholders to build a common understanding of the challenges in the project in early stages, which might help to prevent misconceptions and get the programmers to work on the things the stakeholders really want.*" Another one wrote, "*… the kernel is easy to understand, and people, even non-technical staff, will have a rapid coupling with SEMAT.*" These reviews highlight the simplicity of the SEMAT kernel and foresee a high degree of acceptance in the business world.

Within the list of specific advantages mentioned by students it is worth emphasizing the support for post-delivery stages, the ease of use of the set of cards, the support for team organization and its relationships with the stakeholders, and also, the way it shows to the team that a software project involves several dimensions, apart from the code itself.

5.2 *Drawbacks and ideas for improving SEMAT*

After this initial experience using the SEMAT kernel, the students suggested that the initiative could be improved in two ways. In the first place, they would like to see a predefined set of practices that could serve as a basis for inexperienced teams or when the team members have highly dissimilar backgrounds. Second, the checklists should be initially provided to inexperienced teams in full format as the short-form of these checklists might not be self-explanatory. This short format is more suitable for teams that are already familiar with SEMAT. This pair of drawbacks reflects difficulties experienced by students in their first contact with SEMAT and agile principles, and therefore, is an issue that might suggest

that, at least in learning contexts, it could be appropriate to advise teams unfamiliar with SEMAT and agile methods about the selection and use of agile practices and checklists.

Regarding tool support for SEMAT, one student noted: "*It would be nice to have some software support in order to minimize the work overload … A web-based platform that provides something similar to mural.ly but especially built for the kernel. I think that if such software were built very well, it might support the use of the kernel in software projects, even more by giving hints, warnings and references to a database of similar projects conducted previously, such that the users can learn from those as examples.*" This suggestion is related to the fact that the use of cards is very useful as a teaching tool, but it is not well suited for representing and tracking the successive states of the project.

As a general remark, the students believe that although it is unlikely that SEMAT solves all the problems that promise to solve, it is feasible that this initiative will substantially improve the teaching and practice of software engineering in the near future.

5.3 *Will they use SEMAT in their future projects?*

One student pointed out: "*I can definitely see myself using the kernel in the future. But I suppose I will not be in a leading position in software projects in the near future since I'm still quite young. But once I have some experience or see that things are going wrong in a project, I might suggest using the alpha state cards to establish terminology and a common understanding of the problems in the project.*" Another one noted: "*Of course I will use it. During my short experience using SEMAT I felt that defining the iterations is fairly easy, since it controls the scope of each and avoid future inconveniences and distractions by focusing on current priorities. On the other hand, I think that due to the novelty of this initiative, people involved in the software industry will receive it in a very good way, because it is new knowledge for them, easy to understand, fairly visual, and thus, developers will have rapid coupling with SEMAT.*" As a last one example, a third student noted: "*Personally, I really want to apply SEMAT within projects in which I can participate in the future; I find the kernel very useful and I think its proper application ensures the successful completion of many projects.*"

The opinions of the students regarding the future use of SEMAT predict high adoption rates of the Initiative by software developers. Similarly, this feedback also indicates that within the students the SEMAT will provoke great enthusiasm.

6 CONCLUSIONS AND FUTURE WORK

In this Chapter we presented a preliminary experience for introducing SEMAT and agile development in a software engineering course. The pedagogical approach aims at providing the students with practical know-how and skills to successfully cope with technical challenges and non-technical issues that typically include learning domain specific subjects, working in a team, organizing the work in short time-boxed iterations with adaptive, evolutionary refinements of plans and goals.

We also discussed the special circumstances surrounding software projects developed in an academic context, and proposed alternatives for adapting agile principles to this environment. Apart from that, we described a concrete strategy for adding the SEMAT kernel and language to a project-based course that has been using agile principles for a long time. The proposal includes the use of the kernel alphas as a roadmap that guides teams from the definition of the project, during the development, until its final presentation at the end of the course. Similarly, the proposal points out that the alphas provide the lecturers with a single conceptual framework that allows them to evaluate and compare projects, as well as to monitor their progress and health throughout the course.

We also mentioned a set of online tools suitable for assisting and helping the team to correctly apply the agile principles for organizing and managing iteration tasks. This includes a tool for manipulating the deck of cards so that the current state of the project and the sequence of transitions are easily recorded and are also accessible for everybody; a tool for tracking and managing the committed tasks within each iteration of the project or sprint, and a project hosting service that provides the team with a collaborative development environment. At the same time, these tools allow for the lecturer to constantly monitor the progress and health of the project, and evaluate the teamwork, as well as the individual contributions of each member.

Finally, we reported the experiences of two software projects developed as a preliminary pedagogical experiment. The students used practices and artifacts from the Scrum method and SEMAT kernel and language as a tool for measuring progress and planning the work. The experience was described as successful by all participating students, who further highlighted the main advantages of the SEMAT kernel, pointed out some drawbacks when using SEMAT in combination with agile practices, and proposed various ideas for improving the SEMAT proposal and its usage.

Currently, we are applying this pedagogical approach in a complete software engineering course in which there are 35 students who are working on 7 projects. We are tracking the tasks, activities, and meetings of each student team with the aim of analyzing and reporting our findings in a future publication. Simultaneously, we are working on a web-based plat-

form that is going to help software teams to apply the SEMAT kernel and track all their iterations, states and transitions. This tool would be suitable for instructors not only to track and evaluate the team work, but also to have a more accurate view of the impact of the SEMAT framework on academic projects.

7 ACKNOWLEDGEMENTS

We express our deepest gratitude to the students who actively participated in the development of the two projects described. They allowed us to enrich our knowledge about SEMAT and contribute to the academic world through this work.

8 REFERENCES

Alfonso, M. I., & Botia, A. 2005. An Iterative and Agile Process Model for Teaching Software Engineering. In *18th Conference on Software Engineering Education*, 9–16.

Bavota, G., De Lucia, A., Fasano, F., Oliveto, R., & Zottoli, C. 2012. Teaching software engineering and software project management: An integrated and practical approach. In *2012 34th International Conference on Software Engineering (ICSE)*, 1155–1164.

Budd, A. J., & Ellis, H. J. 2008. Spanning the gap between software engineering instructor and student. In *Proceedings of the 38th Annual Conference Frontiers in Education, 2008. FIE 2008.*

Gnatz, M., Kof, L., Prilmeier, F., & Seifert, T. 2003. A practical approach of teaching Software Engineering. In *Proceedings of the 16th Conference on Software Engineering Education and Training, 2003. (CSEET 2003)*, 120–128.

Jacobson, I., Spence, I., & Ng, P.-W. 2013. Agile and SEMAT: perfect partners. *Communications of the ACM*, 56(11): 53–59.

Kajko-Mattsson, M., Jacobson, I., Spence, I., McMahon, P., Elvesater, B., Berre, A. J., & MacIsaac, M. 2012. Refounding software engineering: The Semat initiative. In Proceedings of the 34th International Conference on Software Engineering (ICSE), 1649–1650Larman, C. 2003. *Agile & Iterative development: A Manager's Guide*. Addison Wesley.

Muller, M. M. & Tichy, W. F. 2001. Case study: extreme programming in a university environment. In *Proceedings of the 23rd International Conference on Software Engineering, 2001. ICSE 2001*, 537–544.

Ng, P.-W., & Huang, S. 2013. Essence: A framework to help bridge the gap between software engineering education and industry needs. In *Proceedings of the 2013 IEEE 26th Conference on Software Engineering Education and Training (CSEE T)*, 304–308.

Pierce, K. R. 1992. The benefits of maintenance exercises in project-based courses in software engineering. In *Proceedings of the Conference on Software Maintenance*, 324–325.

Rajlich, V. 2013. Teaching developer skills in the first software engineering course. In *Proceedings of the 2013 International Conference on Software Engineering*, 1109–1116. Piscataway, NJ, USA: IEEE Press.

Razmov, V. 2007. Effective pedagogical principles and practices in teaching software engineering through projects. In *Proceedings of the 37th Annual Education Conference Frontiers of the Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07*, 21–26.

Shukla, A., & Williams, L. 2002. Adapting extreme programming for a core software engineering course. In *Proceedings of the 15th Conference on Software Engineering Education and Training, 2002. (CSEE T 2002)*, 184–191.

Zapata, C., & Jacobson, I. 2014. A First Course in Software Engineering Methods and Theory. *Dyna*, 81(183): 231–241.

Chapter #12

SoftRace—the software development race under the SEMAT kernel

W.A. Arévalo Camacho,
Corporación CIDENET, Bogotá, Colombia

J.A. Jiménez Builes & J. Gaviria Giraldo
Universidad Nacional de Colombia, Medellin

1 INTRODUCTION

Learning is an important element that defines the way a person faces a topic. It can make the object of study easier or more difficult for someone (Morales *et al*. 2013). According to Morales *et al*. (2013), the most common learning styles are the traditional theoretical and pragmatic, followed by active learning. The latter is appealing to the students, since it allows for them to experience and live their learning.

Software development is still considered as a high risk activity and the most arduous task for industry. Several methods have been developed and adapted to the specific needs found by those who define the method, with the aim of decreasing risks. This fact has made possible to create and provide the community with many initiatives intended to improve the software development process (Jacobson *et al*. 2012).

On the other hand, the path people follow to create mechanisms to allow for better resource management and to obtain profits is also a critical activity that leads to the creation of many strategies for achieving this goal. One of them is the proposal of Kiyosaki (2001), with the CashFlow game, in which daily life difficulties are laid out and the weaknesses of a poorly financially organized person are shown.

After comparing both tendencies, it was found that different methods are created which try to guide the process in order to improve the results. Nevertheless, it was concluded that—from any perspective—the basis is always the organization of processes and that of the stages of each initiative. The aim is what always changes, but the basis of the model is always the organization that a person, a team, or a company possesses to carry out the process.

In this Chapter we introduce the game SoftRace, a strategy for teaching the player how to establish his/her own processes inside the SEMAT[1] kernel framework (Jacobson *et al.* 2012) by using a CashFlow game (Kiyosaki 2001) adaptation. The game rules and parts are described here. With this game, the theoretical and active styles are mixed in order to help the students master the fundamentals, teaching them to structure their own software development process.

This Chapter is structured as follows: in Section 2 we provide a general scope about active learning and CashFlow; in Section 3 we propose SoftRace as a CashFlow adaptation; finally, in Section 4 we conclude and state the future work.

2 GENERAL SCOPE

2.1 Active learning

Active learning involves the students in the topic of study and favors their understanding of concepts. Games are one way to get students involved in the topic of study. Games produce a less formal environment and allow for the participant to be an active member in the learning process (Morales *et al.* 2013).

Taking this approach as the basis, the game SoftRace is introduced here, with which the participant experiences the importance of having a method and applying good practices when developing software applications. In addition, participants should start using their knowledge to create a strategy leading them to reduce costs and to help their projects advance, have good quality and satisfy the stakeholder needs.

2.2 CashFlow

CashFlow game by Kiyosaki (2001) was taken as a basis. Kiyosaki conceived this game with the aim of improving the financial education of the players. It allows them to understand day-to-day financial activity and demonstrates how lack of planning keeps the player in the *Rat Race*. Also, the game shows that if players use their financial resources properly, they can increase their finances. The Rat Race makes the player always spend their incomes in cravings which do not add value to their money.

[1] The theoretical framework related to SEMAT is completely described in the Preface of this book.

CashFlow sets out two tracks: one teaches how to take advantage of financial resources and the other one shows the profits of that exploitation (see Figure 1). This game helps players to identify businesses that generate better dividends in order to enjoy cravings present in the *Rat Race*, but based on financial organization (Kiyosaki 2001).

Every track has a format in which incomes and expenses are recorded and serves as a support in case the players require a loan either for an opportunity or a craving. Actions in each track are given by events that take place in a normal life of a person (Kiyosaki 2001).

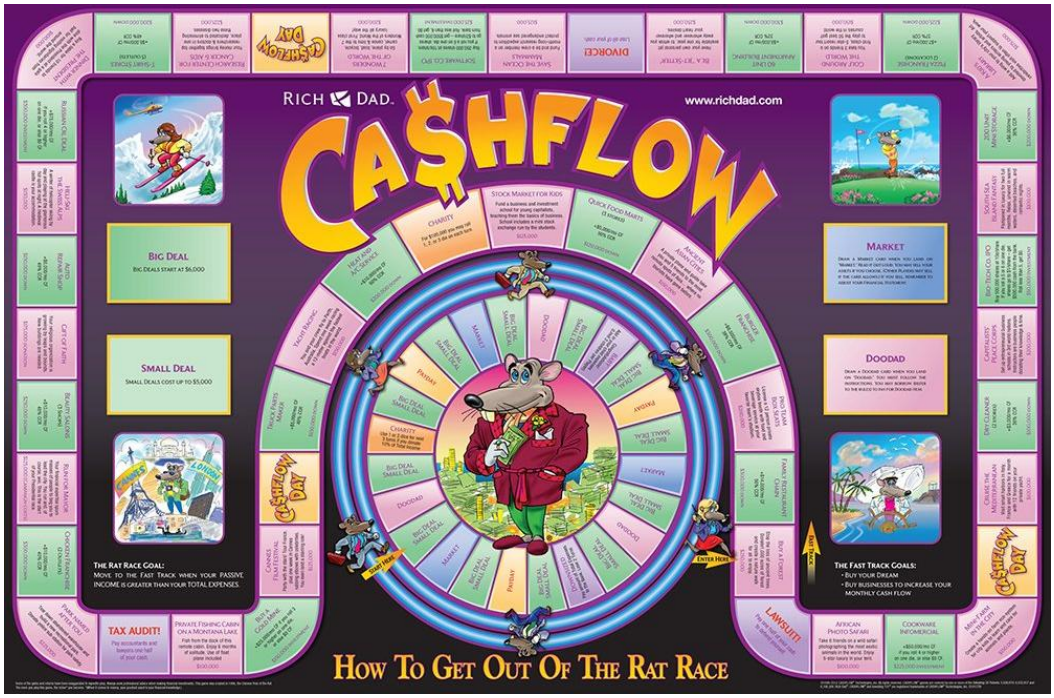


Figure 1. Cash flow Board (Kiyosaki 2001)

3 SOFTRACE

As an analogy to CashFlow, when in a company or a software development team the best practices are not implemented, processes are not defined, or metrics do not exist, then risks increase and quality decreases. The aim of SoftRace is that the player examines how to define and use methods, by clearly defining process and by executing them. Thus, the player realizes that being strict with those conditions substantially reduces the risks in a project and increases the quality of the product.

3.1 General Rules

The game should be played by a minimum of two and a maximum of four teams, each made up of minimum one player. Each team must select one of the projects proposed in the game framework. There will be a person acting as moderator, who will also manage the financial resources.

The teams will make an economic proposal, identifying the tasks to perform and the time they will spend in each task, the resources they will need, and the total duration of the project according to the previous items. Those features will be recorded in a specific format (see Figure 2). The game moderator will define the market cost per hour and the monthly payment due for each resource.

There are several cards named opportunity, risk, market fluctuation, and alpha (see Figure 3), which must be separated and available face down on the board so that the players can draw them. The SoftRace board was adapted from the original CashFlow board as depicted in Figure 4. We kept two rides: the normal race—we named *maturity race*—and the *rat race*.

Every team receives a project description, which indicates restrictions for the project and for the team. Some examples of project descriptions are depicted in Figure 5, including resources and activities with costs and prices.

Once these conditions are defined, the game can start. To do so, the players will roll a dice and will move forward according to the number on the dice. In the *maturity board*, the teams will be faced with the possibilities a development team is faced with every day and in the *rat race* the game dynamics is increased.

3.2 Game Rules

The project profile card is used for describing the features of the project, which was assigned to the team, and monitoring its progress. Every activity and task that will be carried out should be recorded, with a clear description of what each consists of and the resources that will be used for the project.

Finally, the project profile indicates the order in which tasks will be performed and the chronological time each of them will require in order to finish the activity. The total estimation of all the tasks in one activity is the estimated endeavor needed for such activity.

SoftRace

Project Profile

Project Name:					Timeline	
Activity/Task	Description	Resources	Estimated		Start	End
Team						
Resource	Abilities	Education	Experience	Quantity	Unit Cost	
Financial						
Activity	Project cost	Paid	Progress	Fulfil	Status	Profit

Figure 2. Project Profile

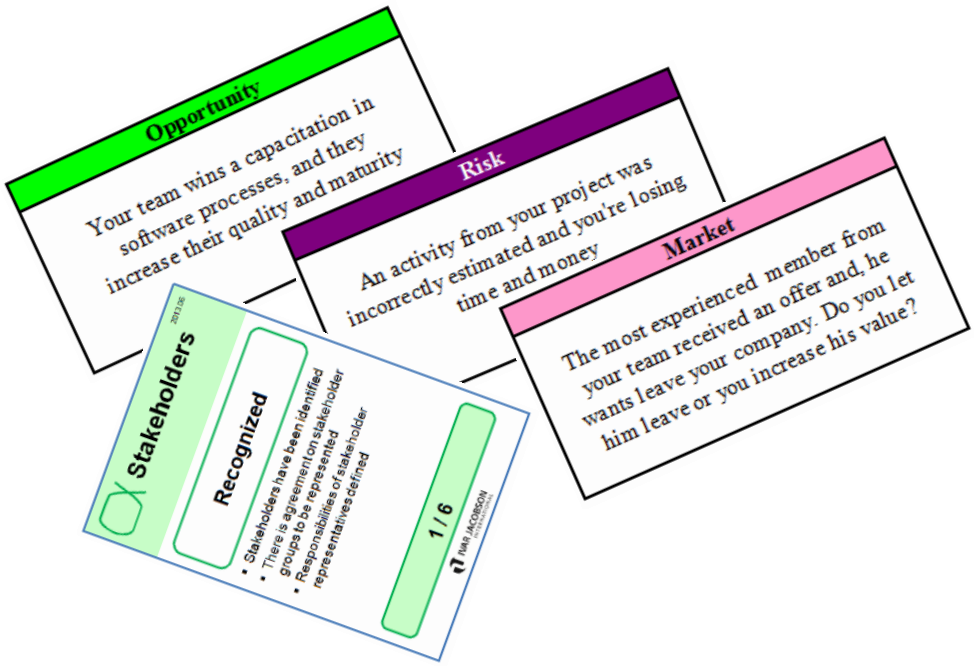


Figure 3. SoftRace cards

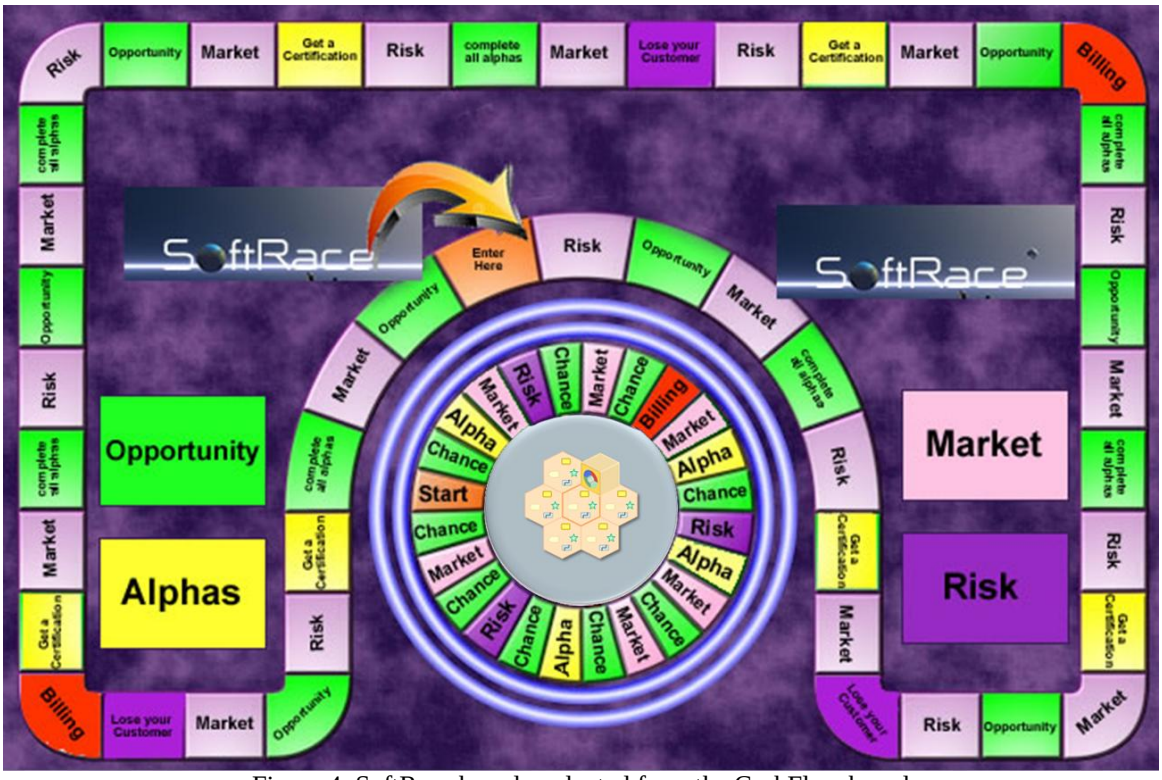


Figure 4. SoftRace board—adapted from the CashFlow board

Your Project

GAME

Your customer wants you to develop a 3D-animated game, managing a ranking, so all players can increase their points looking for being the game King

Resources	Quantity	Cost
Gamer	1	800
Developer	4	8.000
Designer	3	7.000
Tester	1	1.000

Activities	Prices	
Back-end development	40.000	
Front-end development	30.000	
Strategy planning	5.000	Per Month
Testing	10.000	Total 85.000

Your Project

GEOREFERENCING

Your customer needs an App to record the location where users are asking for some services. With those data, he wants to make a plan to increase his sales.

Resources	Quantity	Cost
Architect	1	1.500
Developer	2	2.000
Designer	1	1.200
Tester	1	1.000

Activities	Prices	
Back-end development	30.000	
Front-end development	15.000	
Data Modeling	3.000	Per Month
Testing	10.000	Total 58.000

Your Project

INVOICING

Your customer needs a software application for controlling sales and stock, expenses, and employees. So, he can make better business decisions

Resources	Quantity	Cost
Architect	1	1.500
Developer	3	6.000
Designer	2	2.500
Tester	2	2.000

Activities	Prices	
Back-end development	60.000	
Front-end development	20.000	
Data Modeling	5.000	Per Month
Testing	20.000	Total 105.000

Your Project

SUPPORT

A big bank requires a company for supporting a software system devoted to control financial information. New features are needed.

Resources	Quantity	Cost
Manager	1	2.000
Requirements	1	1.500
Developer	2	4.000
Tester	1	1.500

Activities	Prices	
Requirements	4.000	
Development	10.000	
Testing	4.000	Per Month
		Total 18.000

Figure 5. Project descriptions

Each team sets up the resources it counts on as a team, identifying the qualities of each resource according to the project profile card. This means that if the team has a software developer, the programming tools it uses and the kind of training and experience required are defined. Those data should be recorded in the project profile card (see Figure 2).

Besides, the value that will be obtained for the project and the total time that will be spent by adding up all the resources are also recorded.

Defining the team does not mean that there must be one player per each defined resource. However, the roles should

be clearly understood in order for the team to be able to carry out a software development project. Once the initial profile card recording is fulfilled, the team will receive all the features of the alphas, and will be allowed the selection of three alpha cards per each resource, in order to categorize the team.

Each rolling of the dice represents the number of spaces the player moves forward, as well as the progress in the time given to the assigned task in the selected project. Once the time assigned by the team to each activity is over, the team should deliver the activity and the moderator will evaluate whether it was delivered within the agreed deadline and will give the agreed value for that object to the team. The team will pay the resources and will record the profit from that delivery.

Once a team has completed all the alphas from the endeavor category, it goes to the Fast Track, in which the development speed is doubled. This means that twice the number indicated by the dice must be recorded on the project profile card.

The game is over when one of the teams completes the project that was assigned to it. In case a team loses liquidity and does not have enough money to pay for the project resources when it is due, that team loses one turn and its project is delayed if it has more projects. In case the team only has the main project, it loses and is out of the game.

3.3 The Game Card

The board has four different types of cells, which match the types of cards. The team should take one card that matches the cell type:

Opportunity: it offers the chance to improve the team, product, or relationship with the customer.

Risk: it shows the common disadvantages in a project that cause the team to be delayed or to lose quality of the product.

Market fluctuation: it could be prize or penalty; the opportunity to take an additional project, obtain or lose resources of the project or change the requirements or the stakeholder.

Alphas: they present the ability that satisfies one feature of an alpha; the team should match it with its corresponding alpha. The aim of the game is to complete the alphas to be able to finish the project.

3.4 Resources

The resources are the staff the team should have to carry out the project; this is explained to the user, but he/she is supposed to have an idea about which would be a suitable team according to the project and the conditions each resource must have. Nevertheless, in case its liquidity and estimation allow for the team to obtain another resource, it may do it.

4 CONCLUSIONS AND FUTURE WORK

This game was tested with some students from the National University of Colombia and with the employees from a software development company. Participants found remarkable the fact that they can realize how the financial part behaves, and see both the relationship between how capacity increases when processes are defined and standardized, and the accuracy of their estimations. This directly improves their incomes and the profits. In addition, estimations are carried out based on metrics created from experience.

If the player does not know the kernel, he/she understands how the abilities on the cards help to meet the alpha checklist. Besides, abilities in the game are associated with those a software engineer needs to have in order to create a product suitable to the needs of the stakeholder.

The experience behind SoftRace give us the motivation to promote some other games related to the software engineering and, particularly, to the SEMAT ideas. We also need to design an experiment to prove the results we observed in the game. Finally, the automation of SoftRace will provide the possibility to share this game with other SEMAT practitioners and methodologists in the world.

5 REFERENCES

Jacobson, I., Pan-Wei, N., McMahon, P., Spence, I. & Lidman, S. 2012. The Essence of Software Engineering: the SEMAT Kernel. *Communications of the ACM* (10): 42-49.

Jacobson, I., Spence, I & Pan-Wei, N. 2013. Agile and SEMAT - Perfect Partners. *Communications of the ACM* 9(11):54-61.

Kiyosaki, R. 2001. *Rich dad poor dad*. New York: Grand Central Publishing.

McMahon, P. 2013. *SEMAT and SWEBOK: A Perfect Marriage?* SEMAT blog.

Morales, A., Rojas, E., Hidalgo, C., García, R. & Molinar, J. 2013. Relación entre estilos de aprendizaje, rendimiento académico y otras variables relevantes de estudiantes universitarios. *Revista Estilos de Aprendizaje*, 11(12):151-166

This page intentionally left blank