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.
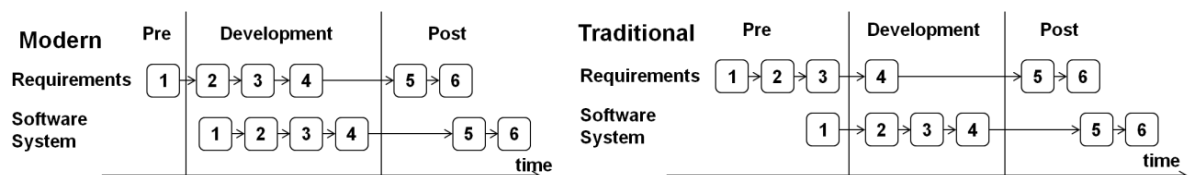

Figure 1. Difference between Traditional and Modern Development Approaches (Ng & Huang, 2013)

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.
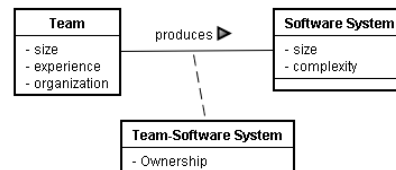

Figure 2. Augmenting Alphas with Structural Properties

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).
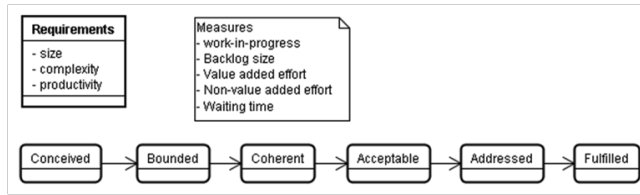
Figure 3. Augmenting flow Measures to Alpha States

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:

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

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

$$\text{alpha-property:value}$$

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:

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

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 short-hand 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.

$$P_A = \{\text{Team-size: small, Team-distribution: co-located,}$$

$$\text{Software System-technology: web}\}$$

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.
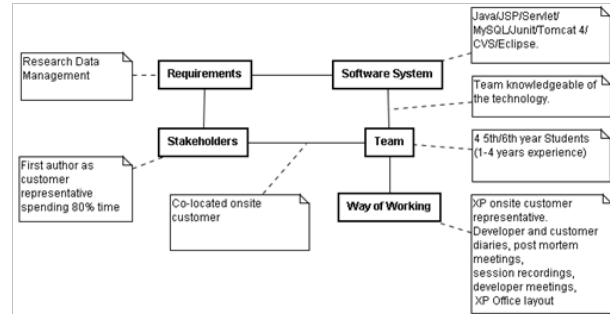

Figure 4. Using Essence structural properties to describe study context

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.
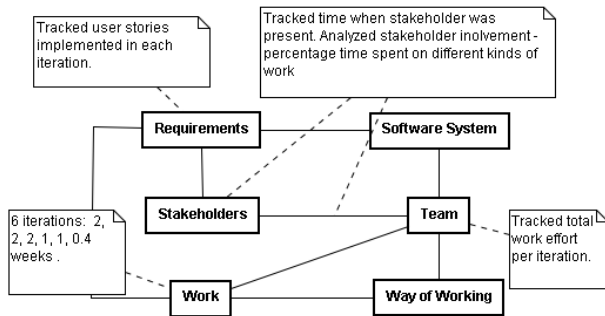

Figure 5. Using Essence to describe flow properties of observed behaviors

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 the 10th International Workshop Software Technology and Engineering Practice (STEP 2002)*, 74-78.