On the Value of Essence to Software Engineering Research: A Preliminary Study

Pan-Wei Ng
Ivar Jacobson International
Singapore
panwei@ivarjacobson.com

Shihong Huang
Department of Computer & Electrical Engineering and Computer Science
Florida Atlantic University
USA
shihong@fau.edu

Yumei Wu
School of Reliability and Systems Engineering
Beihang University
China
wuyumei@buaa.edu.cn

***Abstract*—There has been growing interest in the use of empirical approaches in software engineering research. However, many researchers pointed out that a framework for reporting software engineering case study findings is lacking. As a consequence, it is difficult to compare results or evaluate their generality. In this paper, we use Essence as a foundation for such a framework. Essence is a software engineering kernel and language developed by SEMAT, whose aim is to find a common ground of software engineering. We chose Essence as a foundation because of its unique features such as being comprehensive, model-based, and extensible. We demonstrate the use of this framework by taking an existing case study research and analyzing how Essence could report its findings more systematically and comprehensively. The case study we investigate is about whether the customer representative role is too demanding in an extreme programming (XP) environment. Using the Essence framework, we found a strong validity threat to this case study. Although the validity threat could also be detected otherwise, the fact that it was not detected previously only highlight the importance of having greater rigor in case study reporting and evaluation, which is why a framework is necessary. Moreover, by using Essence as a foundation for such a framework, we show the value Essence brings to the software engineering research community.**

***Index Terms*—Software Engineering, Research, Essence, SEMAT, Kernel, Case Study, Empirical Validation**

I. Introduction

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, Vessey, and Ramesh [1] 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) [2] [3] was an example of such efforts. 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, Meyer and Soley [4] noted that software engineering today is gravely hampered by immature practices. They later started the SEMAT [5] initiative with the goal to bridge this gap [6]. As a first step, SEMAT working groups have established a language and kernel that define commonly used elements in software engineering, known as Essence [7]. Jacobson, Ng, McMahon, Spence, and Lidman [8] [9] 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 [8] [9].

A. Need for a Framework for Empirical Research

Surely, much needs to be done to bring research and industry together. Huang and Tilley [10] 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 paper. We recognize that there have been generally accepted approaches on conducting research and reporting case studies, such as those by Shaw [11] and Runeson and Höst [12]. These focus more on research methods. We on the other hand 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

978-1-4673-6273-3/13/$31.00 © 2013 IEEE

GTSE 2013, San Francisco, CA, USA

evaluate properties for their realism and the generality of the findings.

B. Essence as a Foundation for a 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:

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.

Model based – Essence expresses the relationships between these different dimensions, and hence provides a foundation to model the relationships between various properties of interest.

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 as practices.

In this paper, we use Essence as a foundation for a systematic framework to report case studies. This is achieved after augmenting Essence with different kinds of properties for describing case study research and analytical guidelines for evaluating research data and findings.

C. 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 [13]. 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 evaluated how its properties of interest could be better organized. We analyzed if it had included relevant properties of interest.

To our surprise, through our framework, we detected a strong threat to the validity of this case study. Understandably, this validity threat could 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.

II. Brief Review of Current State of Software Engineering Research

In this section, we briefly review the current state of software engineering research. We focus specifically on the area of empirical research because of its growing recognition [14] [15].

Dyba˚ and Dingsøyr [16] surveyed research for empirical evidence of agile software development and found that different reporting content hinders analysis. Jedlitschka, Ciolkowski, and Pfahl [17] 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 [18] 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 [19] 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 [20] also discussed the generalizeability of 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 readers' 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.

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

III. Key Concepts in Essence

In this section, we give a brief overview of Essence. We describe some of the key concepts in Essence [8] [9], namely *alphas* and *alpha states*.

Alphas – Essence uses an object oriented approach to identify typical dimensions of software engineering challenges. These objects are called *alphas*. Essence kernel identifies alphas that are common to software development such as *Opportunity*, *Stakeholders*, *Requirements*, *Software System*, *Work*, *Team*, and *Way-of-Working*.

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. For example, the progress of (a set of) Requirements goes through the following states: *Conceived*, *Bounded*, *Coherent, Acceptable, Addressed, Fulfilled*. Essence kernel provides a detailed checklist for each alpha and their states.

Extensible – Essence provides mechanisms to introduce additional alphas. For example, a team practice can introduce a *Team-Member* alpha and suggest team organization patterns; a component-based development practice can introduce a Component alpha and component patterns.

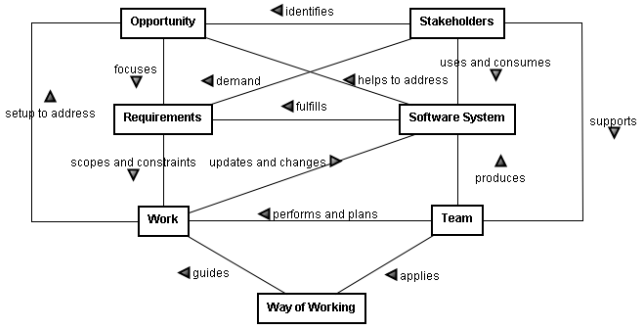

Fig. 1. Essence Alphas and Their Relationships [7]

IV. 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 [7] does not, at the time of writing of this paper, 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 paper.

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.

A. Structural Properties

Structural properties are those that describe and characterize the structure and complexity 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, Morasca, and Basili [21] provided 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. [22] [23] through 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.

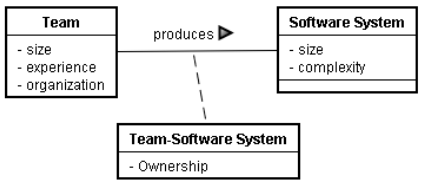

Fig. 2. Augmenting Alphas with Structural Properties

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

B. 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 paces. 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 (Fig 3).

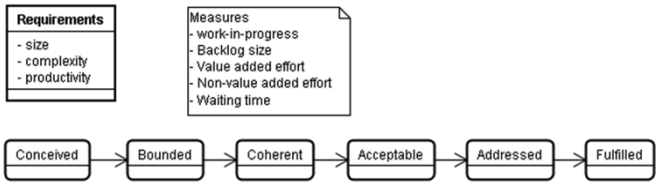

Fig. 3. Augmenting flow Measures to Alpha States

Such properties are used when looking for productivity gains and are heavily used in lean process improvements such as [24] [25]. We want to point out that the productivity properties have to be anchored to the items being worked on. In the case of Fig. 3. they are about Requirements, as opposed to architecture work, which should be pegged to the Software System alpha.

V. Providing Analytical Guidelines

As highlighted earlier, the software engineering research community lacks a common framework to report studies [16] [17] [18] [19] [20]. 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:

1. Comprehensiveness measures are about whether the study has adequate and relevant information to help the reader understand it.
2. Distance measures allow comparisons as to how far a study is to reality or to a reader's 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 in Section VI.

A. 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:

alpha-property:value

For example, consider the development endeavor E_{Web} of a simple web-based system performed by 3 students working together. Its description is a list of properties:

$P_{E_{Web}}$ = {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 4 pieces of information (i.e., 4 descriptions of 4 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 paper. For the purpose of this paper, 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.

B. 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} through 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 [27] 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 paper. In this paper, we will use intuitive terms like *weak* or *strong* for comprehensiveness and *near* or *far* for distance.

VI. An Example of Using Essence To Conduct Research

In Section I, 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 [13]. This case study was not arbitrarily chosen. Dybå et al. [16] 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 [13] was one of these 33 studies. Using the guidelines from Runeson and Höst [12], we organize our analysis in the following sub-sections:

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

In each section, 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 Section IV to describe the case study.
- Essence Analysis – We use the guidelines in Section V to analyze the comprehensiveness of the case study and its validity and highlight possible issues concerning the study.

A. 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:

1. The practice *P*, which is extreme programming here.
2. The experimental environment, which is the software engineering endeavor *E*, conducted in a university setting.
3. The real world environment *R*, which readers of the study are interested in.
4. 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's 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.

B. 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 storie 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 [28] [29] 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 Fig. 4. 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.

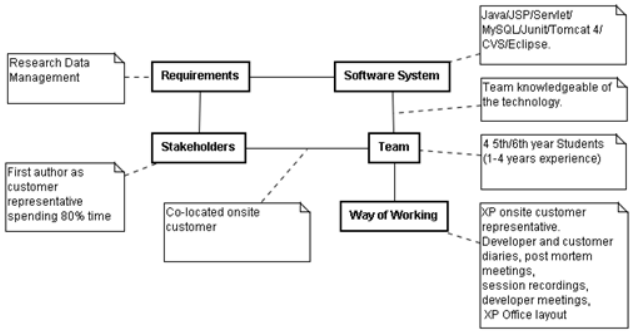

Fig. 4. Using Essence structural properties to describe study context

Essence Analysis – From the study description and the visual representation in Fig. 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 Fig. 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.

C. 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 Fig. 5.

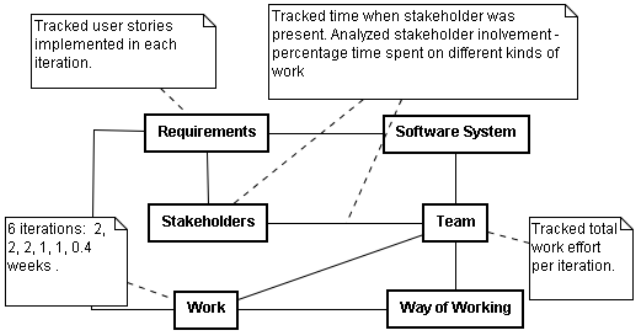

Fig. 5. Using Essence to describe flow properties of observed behaviors

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's 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.

D. Analyzing and Concluding the Research

Study Description – Koskela and Abrahamsson [13] 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:

1. In the experimental setup, the stakeholder environment is simplistic, i.e. $D_{Stakeholders}(R, E)$ is far.
2. 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, 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.

VII. 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 paper, 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 more stronger. More importantly, the results would make comparisons between studies and practices more accurate.

References

[1] Robert L. Glass, Iris Vessey, and Venkataraman Ramesh, "Research in software engineering: an analysis of the literature," *Information and Software technology* 44, no. 8, pp. 491-506, 2002.

[2] Scott Tilley, Hausi Müller, Liam O'Brien, and Ken Wong "Report from the Second International Workshop on Adoption-Centric Software Engineering (ACSE 2002)." *10th International Workshop Software Technology and Engineering Practice* (STEP 2002), pp. 74-78, 2002.

[3] Bob Balzer, Marin Litoiu, Hausi Müller, Dennis Smith, Margartet Anne Storey, Scott Tilley, and Kenny Wong. "4th International Workshop on Adoption-Centric Software Engineering," in *Proceedings of 26th International Conference Software Engineering* (ICSE 2004), pp. 748 – 749, 2004.

[4] Ivar Jacobson, Bertrand Meyer and Richard Soley, "The SEMAT initiative: A Call for Action", *Dr Dobb's Journal*, December 09, 2009.

[5] SEMAT (Software Engineering Method and Theory) online at www.semat.org

[6] Ivar Jacobson, Shihong Huang, Mira Kajko-Mattsson, Paul McMahon, and Ed Seymour. "Semat—Three year vision," *Programming and Computer Software*, Volume 38, Issue 1, pp 1-12, Springer January 2012.

[7] Essence - OMG Submission online at http://www.omg.org/cgi-bin/doc?ad/2012-11-01

[8] Ivar Jacobson, Pan-Wei Ng, Paul McMahon, Ian Spence, and Svante Lidman. "The essence of software engineering: the SEMAT kernel," in Communications of the ACM, no. 10, pp . 42-49, 2012.

[9] Ivar Jacobson, Pan-Wei Ng, McMahon, Spence Lidman, The essence of software Engineering: applying the SEMAT kernel. Addison-Wesley, 2013.

[10] Shihong Huang and Scott Tilley, "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). May 2004.

[11] Mary Shaw, "What makes good research in software engineering?" in International Journal on Software Tools for Technology Transfer (STTT) 4, no. 1 (2002): 1-7.

[12] Höst Runeson and Martin Höst, "Guidelines for conducting and reporting case study research in software engineering." in Empirical Software Engineering 14, no. 2 (2009): 131-164.

[13] Juha Koskela and Pekka Abrahamsson, "On-site customer in an XP project: empirical results from a case study." in: T. Dingsøyr (Ed.), *Software Process Improvement, Proceedings, Lecture Notes in Computer Science*, vol. 3281, Springer-Verlag, Berlin, 2004, pp. 1–11.

[14] Tore Dybå, Barbara A. Kitchenham, and Magne Jørgensen, "Evidence-based software engineering for practitioners." in IEEE Software 22, No. 1 (2005): 58-65.

[15] David Budgen, Gene Hoffnagle, Matthias Müller, Francois Robert, Asma Sellami, and Scott Tilley, "Empirical software engineering: a roadmap report from a workshop held at STEP 2002, Montreal, October 2002," in *Proceedings of 10th International Workshop on Software Technology and Engineering Practice*, (STEP 2002), IEEE, pp. 180-184, 2002.

[16] Tore Dyba˚ and Torgeir Dingsøyr, "Empirical studies of agile software development: A systematic review." *Information and software technology*, 50, no. 9, pp 833-859, 2008.

[17] Andreas Jedlitschka, Marcus Ciolkowski, and Dietmar Pfahl, "Reporting experiments in software engineering," in *Guide to Advanced Empirical Software Engineering* (2008), 201-228, 2008.

[18] Kai Petersen and Claes Wohlin, "Context in industrial software engineering research," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE Computer Society, pp. 401-404, 2009.

[19] Robert Feldt and Ana Magazinius, "Validity threats in empirical software engineering research - An initial survey," in *The 22nd International Conference on Software Engineering and Knowledge Engineering*, SEKE, 2010.

[20] Emerson Murphy-Hill and Laurie Williams. "How can research about software developers generalize?" in *5th International Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE), pp. 105-109, 2012.

[21] Lionel C. Briand, Sandro Morasca, and Victor R. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, 22, no. 1, pp. 68-86, 1996.

[22] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. "Putting it all together: Using socio-technical networks to predict failures," in *20th International Symposium on Software Reliability Engineering*, (ISSRE'09), pp. 109-119, 2009.

[23] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. "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 13th European Software Engineering Conference*, pp. 4-14. 2011.

[24] Shahid Mujtaba, Robert Feldt, and Kai Petersen. "Waste and lead time reduction in a software product customization process with value stream maps," in *21st Australian Software Engineering Conference* (ASWEC), IEEE, pp. 139-148, 2010.

[25] Kai Petersen and Claes Wohlin. "Software process improvement through the Lean Measurement (SPI-LEAM) method," *Journal of Systems and Software* 83, no. 7, pp. 1275-1287, 2010.

[26] Lionel C. Briand, Khaled El Emam, and Sandro Morasca, "On the application of measurement theory in software engineering," *Empirical Software Engineering* 1, no. 1, pp 61-88, 1996.

[27] Per Runeson. "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, pp. 95-102. 2003.

[28] Ron Jeffries, Ann Anderson, and Chet Hendrickson, *Extreme Programming Installed*, Addison-Wesley Professional, 2001.

[29] Kent Beck and Cynthia Andres, *Extreme Programming Explained: Embrace Change*, Addison Wesley Professional, 2004.