CAPTURING THE ESSENCE OF SOFTWARE ENGINEERING—AN ACADEMIA PERSPECTIVE OF SEMAT

Shihong Huang, Associate Professor, Dept. of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, USA, shihong@fau.edu

1. INTRODUCTION

Semat initiative was launched by Ivar Jacobson, Bertrand Meyer, and Richard Soley at the end of 2009. Semat addresses the many challenges we face today in software engineering field. In essence, the biggest challenge is to understand how to build better software, and why we need a theory for software engineering.

This chapter reflects the Semat vision and briefly discusses the differences between science and engineering. Then, the chapter proposes the three-layer hierarchical structure of universals in the context of engineering from general to specific, highlights some of the uniqueness of software products, and finally, it describes how academia could utilize and benefit from the Semat initiative.

2. A REFLECTION ON SEMAT VISION STATEMENT

The Software Engineering Method and Theory (Semat) initiative (Semat, 2011) aims at addressing some of the prevalent problems of software engineering theory and practices, and to revitalize this discipline. Some of the problems mentioned in the Purpose and Scope of the Vision Statement (Jacobson *et al.*, 2009) include prevalence of fads, lack of a sound and widely accepted theoretical basis, large number of methods and their variants with differences little understood and artificially magnified, the need of credible empirical evaluation and validation, and the gap between industry and academia.

In order to address these problems, Semat sets five goals: defining the basic definition of software engineering, providing a strong mathematical basis to the discipline, identifying the truly universal elements to be integrated into the kernel, and providing assessment techniques to evaluate software practice and theories, including the results of Semat. The kernel and its elements will be precisely defined using the domain-specific language (the domain being practices for software development). Practices are defined in terms of the kernel elements and can be considered as extensions of the kernel. Additionally, the language will also be used to define practices and entire methods Figure 1.

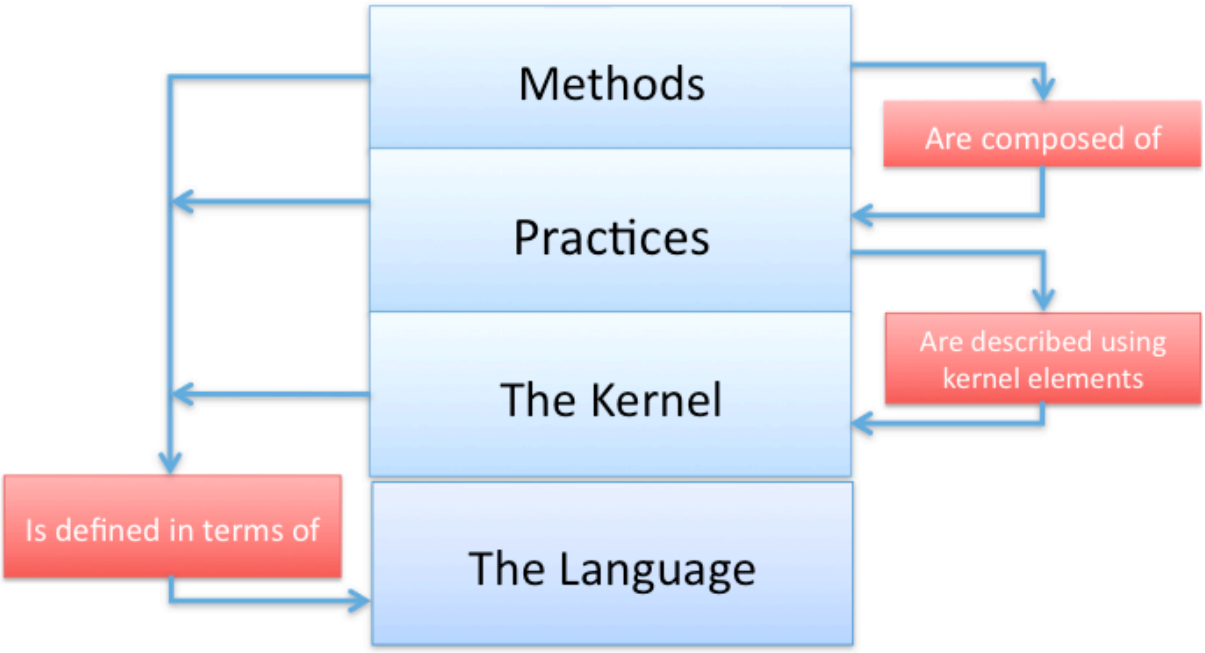

Figure 1. The method architecture (Jacobson *et al.*, 2011).

The one-line goal of Semat is to "create a kernel and a language that are scalable, extensible, and easy to use, and that allow people to describe the essentials of their existing and future methods and practices so that they can be composed, compared, evaluated, tailored, used, adapted, simulated and measured by practitioners as well as taught and researched by academics and researchers." (Jacobson *et al.*, 2009)

According to the Vision Statement, the two immediate tasks are identifying the Universals and defining the Language; these two tasks will lay the foundation for the other three tasks.

This section discusses some of the candidate aspects and attributes of the Universals to complement what was discussed in Vision Statement Appendix 3 (Jacobson *et al.*, 2009). The goal of the Universal track is to identify "the universal elements of software engineering, which must be integrated into the Semat kernel," and in the meantime, to "keep the kernel concrete, focused and small". Identifying Universals should be based on the definition of software engineering and their essential concepts of the discipline (Jacobson *et al.*, 2009, Task 1). To some extent, Universals and Definitions are mutually tightly coupled and restrained—Definition establishes the scope of Universals, and Universals codify Definition. So, our first task is to have a basic understanding of what "software engineering" is and what the uniqueness of software engineering is from other engineering disciplines.

3. THE DIFFERENCE BETWEEN SCIENCE AND ENGINEERING

By simply parsing the term "software engineering", we understand there are two parts which constitute this discipline—one is "software", the other is "engineering"; hence the obvious definition: "Software engineering is the application of engineering methods and discipline to the field of software" in Vision statement [A1.1] (Jacobson *et al.*, 2009). Whether this definition is sufficient or precise (Petroski, 2010) we leave it to the community to discuss. However, one would agree that software engineering is indeed an engineering discipline. Therefore we should treat software engineering the engineering way. However, there is significant difference between science and engineering. In Henry Petroski's recent book (Petroski, 2010) he points out the difference of the two endeavors as:

- Science seeks to understand what is, whereas
- Engineering seeks to create what never was

It is not appropriate to describe engineering as mere applied science. There are some extra-scientific components to engineering, something often referred to as the creative nature, situated culture and particularity to a specific application domain. In engineering, we often do the design first, and then the hypothesized structure can be given a scientific or mathematical litmus test (Koskela & Howell, 2002). It is essential to keep in mind the similarities and differences between science and engineering, when we give the definition of software engineering and define the Universals. In engineering, analysis follows synthesis and observation—not the other way around. Unlike science that deals with the universal laws, which are context and time independent and true everywhere, engineering deals with situated culture, needs to have constant learning, refinement and adaptation to meet the environmental requirements.

4. THE UNIVERSALS HIERARCHY

Given the nature of software engineering discipline, the Universals should be a hierarchical structure which consists of three layers of components from general to specific: Engineering Universals, Software Engineering Universals, and Domain Specific Software Engineering Universals. They are shown in Figure 2.

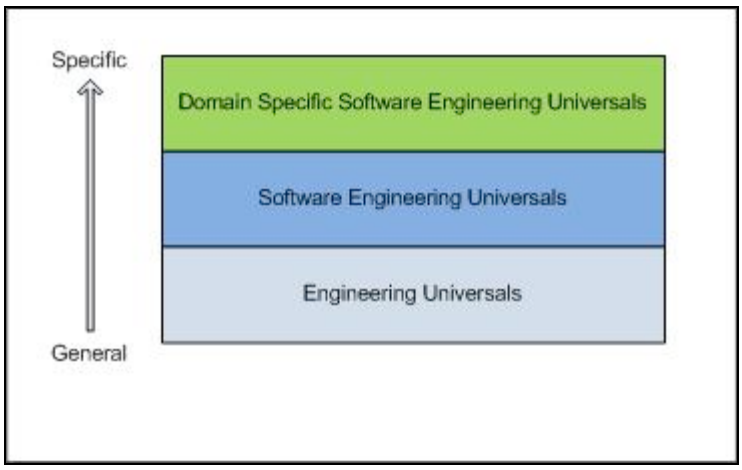

Figure 2. Hierarchy of Universals.

(1) Engineering Universals: this layer includes the best practices of engineering disciplines, such as civil engineering, mechanical engineering, electrical engineering, that apply to software engineering. This is the foundation of the "engineering" aspect of the Universals (*e.g.*, project, team, system, quality *etc.* as mentioned in A3.4, Jacobson *et al.*, 2009).

(2) Software Engineering Universals: this layer includes the unique best practices to software engineering, such as extensibility, interoperability, evolveability, reusability, maintainability, *etc.* This is the "software" aspect of the Universals.

(3) Domain Specific Software Engineering Universals: this layer addresses domain specific universals, such as for real-time systems, self-adaptive systems, self-management systems, Web application systems *etc.* This is the "variability" equivalent to software product line approach.

The following section further elaborates on this hierarchy.

5. THE UNIQUENESS OF SOFTWARE PRODUCTS

Since software engineering is an engineering discipline, it possesses some of the engineering fundamentals. For example, it should follow some of the ingredients of engineering project management (Koskella & Howell, 2002) for a project, *i.e.*, transformation, flow, and value generation. From a management perspective, it should have the stages of planning, execution and controlling.

While software engineering follows the engineering fundamentals, there are some unique features of software engineering and software products that are different from general engineering and engineering products, and must be addressed in order to reflect the nature of software. For example, for the engineering model, full specification is followed by design, manufacture, testing, installing, and maintaining; whereas for the software model, we may not need, or couldn't specify everything upfront, we use systematic, iterative, and incremental development. We build as we learn more.

One prominent feature that a software product is different from general engineering products is that software needs to be consistently maintained and evolved to meet new business requirements (Bennett & Rajlich, 2000). The evolution is more important in software than in other engineering disciplines. Software engineering rarely involves "green field" development, because most organizations have substantial legacy

systems. The legacy systems represent significant assets containing valuable components that can be reused (through reverse engineering) (Müller *et al.*, 2000; Canfora *et al.*, 2001) as the system evolves over time to meet changing requirements and new business challenges. The cost incurred in changing the software after its deployment is likely to exceed the development cost by a factor of 3 to 10 (Jacobson *et al.*, 1997).

Given the malleable nature of software, a good collection of Universals should include not only the general engineering universals, that capture the core practices of engineering disciplines, but also some of the unique features of software, such as changeability, extensibility, interoperability, evolveability, and software reuse (Jacobson *et al.*, 1997).

The uniqueness of software determines some of the Universal attributes we must have; these attributes differentiate software engineering from other engineering disciplines.

Besides following the common practices of engineering disciplines and uniqueness features of software engineering, the last layer would be domain specific software engineering Universals that reflect and address the knowledge of different more situated application domains.

6. ACADEMIA IN THE CONTEXT OF SEMAT COMMUNITY

The software engineering community and its most important user groups are described in Semat Three Year Vision paper (Jacobson *et al.*, 2011), “In simple and not exhaustive words: the practitioners and the industry drive what needs to be taught and researched. The academics teach and formalize what needs to be better understood. The practitioners are the doers in the industry and the industry leads the practitioners in creating business value.” (Figure 3.)

One of Semat vision is to “create a platform (the kernel) allowing people to describe their current and future practices, patterns and methods so that they can be composed, simulated, applied, compared, evaluated, measured, taught and researched.” (Jacobson *et al.*, 2009).

By introducing Semat concepts (methods, practices, and the kernel) into the regular software engineering curriculum, we can focus on the fundamental, common ground, widely accepted practices instead of the particular ideas that form every method, process, or methodology. We could teach students the basis of software engineering, followed by training them a set of good practices using that basis.

In this context, academia plays three roles that are interrelated:

- Education (*e.g.*, curriculum development, theses, and dissertation supervision),
- Research (*e.g.*, basic research and applied research), and
- Service to the broader software engineering community.

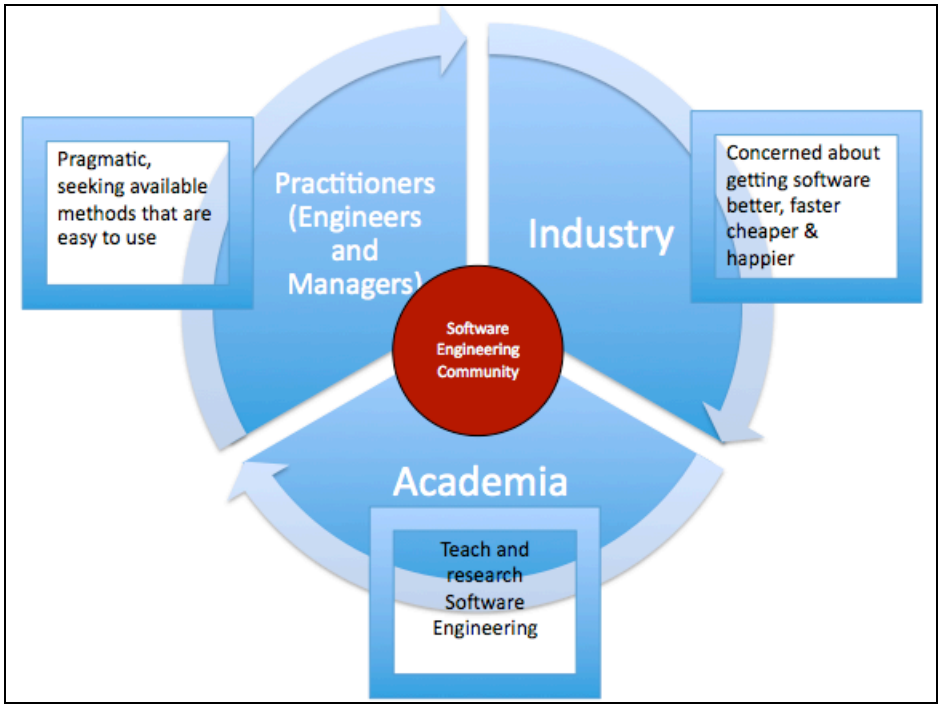

Figure 3. Software engineering community.

As shown in Figure 3, working together with software practitioners and industry, academics can help to:

- Understand what industry needs, and conduct research on these needs by leveraging academia resources, and in turn,
- Provide feedback to industry of Semat practices,
- Standardize teaching and research software engineering principles, artifacts, practices and methods,

- Foster shared understanding, conducting empirical assessment and creating new concepts and related topics in software engineering. The sharing would include educators, students, researchers, practitioners, and other disciplines.

6.1. Teaching the fundamentals

First, we need to identify the skill sets needed for students to understand and use Semat methods, practices, and the kernel and kernel language. Then, we shall develop curriculum that will integrate Semat results into the core courses of computer science, computer engineering, and software engineering degrees. The curriculum design can use Semat as a way to teach basic skills according to and based on available Semat related building blocks. Semat can be used in a way to compare and show students the pros and cons of different methods and practices they encounter during their course study.

Semat practices and language can also be used to design software engineering curricula, based on which provide basic concepts of software engineering in terms of concepts such as universals, practices, and methods.

The research-oriented teaching is related to students' senior projects, thesis and dissertations, when students will spend an intensive time conducting research on a particular topic. Students can focus on their original ideas and can build on existing results more easily due to easier composition of new and existing methods and practices. It is here where we can provide feedback to industry, validate and assess of

Semat practices, and construct new methods, based on universals and existing practices and methods.

6.2. Semat for academic research

Essentially, academia can play many roles in utilizing Semat results and getting feedback to Semat community and industry. Academics can develop the empirical assessment methods to compare software engineering artifacts, practices, and methods; foster the advancement of formal foundations of software engineering, which sharpens our understanding of fundamentals of software engineering concepts. Academics can also provide a research infrastructure that serves as a test-bed and fast deployment of new ideas in the community.

These issues are related to basic and applied research in different ways, which will be discussed in the next two sections.

6.2.1. Basic Research

Semat helps to identify missing formal foundations aspects of software engineering (*e.g.*, theoretical computing, semantics, and so on). Basic research can provide new ways to perform empirical studies in software engineering. Semat itself can also serve as the starting point for new versions of Semat, thus providing new insights into the construction and development of complex technical solutions in heterogeneous and distributed organizational and technical contexts and environments.

6.2.2. Applied Research

Semat provides the basis for better-shared understanding of academia researcher and practitioner from industry. Academia can conduct large and distributed experiments to assess the efficacy of certain Semat methods, practices, and universals, providing empirical evaluation of comparing practices from different methods/ways of working, such as pair programming, Scrum, iterative development, *etc*. Several universities could also utilize their specific expertise to provide better and more comprehensive tool integration for research collaboration.

For example, academics can empirically assess Semat practices and methods in a controlled experimental environment, *i.e.*, classroom setting in universities. Although there are some advantages of using industrial projects, there are also many limitations set by the industrial context and beyond the control of the assessment procedure. In contrast, academic experimenters have much fine control on the experimental parameters, and it is possible to achieve some degree of reproducibility and statistical significance (Semat, 2011).

7. CONCLUSIONS AND FUTURE WORK

Semat initiative addresses the many concerns and issues that challenge the field of software engineering. The greatest challenge is to understand how to build great software, better, faster, and happier. At its heart is to identify a set of widely accepted

kernel elements that provide the common ground references, and the language to describe practices and methods. One of the success factors of Semat is its broad adoption by the primary user groups of the software engineering community—practitioners, industry, and academia. This chapter reflects Semat vision from the academic perspective. It proposes a three-layer, from general to specific, hierarchical structure of universals in the context of engineering. It highlights some of the uniqueness of software products, and discusses Semat impact on academia from education to research. Semat is a multi-year project, a long-term endeavor that involves participants from different disciplines. Academia is definitely one of the main sectors that will benefit and provide feedback to Semat results. Semat provides a solid foundation and abundant topics of education and research. There will have certainly many challenges ahead of us. It takes a big effort from community to reach the goals, and in turn, serves the community.

REFERENCES

Bennett, K. & Rajlich, V. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of International Conference on Software Engineering Proceedings of the Conference on The Future of Software Engineering*. IEEE CS Press, 73-87.

Canfora, G., Di Penta, M., & Cerulo, L. (2001) Achievements and Challenges in Software Reverse Engineering. *Communication of the ACM 54*(4), 142-151. DOI:10.1145/1924421.1924451

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software Reuse: Architecture, Process and Organization for Business Success*. Addison Wesley Professional

Jacobson, I., Huang, S., Kajko-Mattsson, M., McMahon, P., & Seymour, E. (2011). Semat —Three Year Vision. In *Proceedings of Spring/Summer Young Researchers' Colloqiuium on Software Engineering*.

Koskela, L., & Howell, G. (2002). The underlying theory of project management is obsolete. Available online at http://www.leanconstruction.org/pdf/ObsoleteTheory.pdf

Müller, H., Jahnke, J., Smith, D., Storey, M-A., Tilley, S., & Wong, K. (2000). Reverse engineering a roadmap. *In Proceedings of International Conference on Software Engineering Proceedings of the Conference on The Future of Software Engineering*. IEEE CS Press. pp. 47-60.

Petroski, H. (2010) *The Essential Engineer: Why Science Alone Will Not Solve Our Global Problems*. New York: Alfred. A. Knopf.

Jacobson, I., Meyer, B., & Soley, R. (2009). Software Engineering Method and Theory—A Vision Statement. Available online at http://www.semat.org/pub/Main/WebHome/SEMAT-vision.pdf

Software Engineering Method and Theory (Semat). (2011). Available online at www.semat.org