In Search of Software Engineering Foundations: A Theoretical and Trans-disciplinary Perspective

Murat Pasa Uysal

***Abstract*—Technical innovations and trends have been changing from time to time as a major driving force for Software Engineering (SE). Although it can be regarded as a relatively young discipline usually driven by industrial needs or practices, fundamental problems of SE still continue to exist. It is thought that the problem may be not only in adopting a domain specific technology or method, but also in understanding the foundations and use of theories in SE. Therefore, investigating the trans-disciplinary aspects of SE may pave the way of some solutions while it may shed light on building the theoretical background of possible empirical studies. However, the review of SE literature shows the little effort given to this research gap, and thus, this paper attempts to offer a conceptual framework and brings a different perspective for understanding the theoretical and trans-disciplinary foundations of SE as a discipline.**

***Index Terms*—Software engineering, trans-disciplinary foundation, theory use.**

I. Introduction

Most of the constraints in Software Engineering (SE) stem from its intangibility, complexity, human dependency, and the main problem in SE, therefore, can be seen as making the connection between the abstract world and the physical world. Historically, language-centered computer programming has been dominant in SE. Technical innovations have been changing from time to time as a major driving force for SE trends and practices. However, the fundamental problems still continue to exist, and it is claimed that some of the issues may be assumed to be theoretical rather than only practical ones. Thus, the main arguments of this paper are:

1) With an evolutionary point of view, SE can be regarded as a relatively young discipline usually driven by industrial needs and practices. However, it already integrates theoretical and methodological perspectives drawn from other disciplines.
2) Theory is the most important mean to improve SE discipline while facilitating communication of ideas between different research communities.

Therefore, this manuscript attempts to offer a conceptual framework for understanding theoretical and trans-disciplinary foundations of SE.

II. Software Engineering as a Discipline

The Institute of Electrical and Electronics Engineers (IEEE) defines SE as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software". It is an applied discipline and it encompasses processes, methods, tools, standards, and principles in order to build reliable, maintainable and large-scale software systems with high productivity and quality.

In SE, software may play dual role and can be a product or a means to deliver another product acting as the basis for the control and creation of tools and environments. While a software product is the combination of programs, data and other types of computer software, it may also be generic or customized. The former is the stand-alone system developed by an organization and sold on the open market whereas the latter is appointed by a particular customer to a software contractor, and then it is developed specifically for that customer.

In this context, engineering approaches and common engineering principles form the integral part of theoretical and practical fundamentals of SE. In conventional engineering, the common approach is moving from abstract to concrete, and the final product is usually the realization of an abstract design physically. However, in SE, this approach is reversed [1]. Moving from real world to abstract world, the final product is the virtualization and coding of a software design that expresses and represents a real world problem.

As an engineering discipline applying theories, processes, methods, and tools to build high quality software, SE is concerned with all aspects of software production, which are from the phases of requirements and system specification through to the maintenance of the system after delivery. To that aim, SE may incorporate multi-facet principles in other disciplines while allowing the use of SE-domain specific theories. Engineering, Computer Engineering, Computer Science, Mathematics, Systems Engineering, Management, Economics, Cognitive Science, Information Science, Project Management, Quality Management are amongst the major related disciplines, and thus, it may be suggested that software engineers have some knowledge of materials from these disciplines as well as software process.

III. Software Process

Software process, which is the combination of a set of interrelated activities functionally coherent and reusable for SE, transforms inputs into outputs by using resources, tools and techniques. SE processes and activities (planning, requirement, design, construction, testing, configuration management, and maintenance) occur both at the

Manuscript received December 20, 2014; revised May 2, 2015.

Murat Pasa Uysal is with the Department of Computer Technologies, Ufuk University, Ankara, Turkey (e-mail: murat.uysal@ufuk.edu.tr).

organizational and project levels to ensure that the software product is delivered effectively and efficiently for the benefits of all stakeholders. Therefore, SE methods and the use of information, behavioral, structured modeling techniques provide a systematic approach to both problem solving and software development. Software processes that specify and transform requirements into a deliverable software product are included in Software Development Life Cycle (SDLC), and it can be linear or iterative. On the other hand, Software Product Life Cycle includes SDLC plus the additional processes of maintenance, support and evolution.

SDLC begins with the software requirement process that includes elicitation, analysis, specification, validation and management of the needs and constraints placed on a software product and its development. Software design process consists of architectural and detailed design, which in turn describes how software is organized into components and desired behaviors in sufficient detail. As one of the life cycle stages, the design process produces a description of the software's structure with a set of models and artifacts serving as the basis for its construction. Software design and architecture methods, such as Object-Oriented Design and Structured Design, provide a common framework for software engineers. Software construction process refers to the creation of working software through a combination of coding, testing, debugging, verification and integration activities. This stage produces source files, test cases, documentation and configuration items that have to be managed in a software project. Software testing is the verification process whether a program or module meets different types of expectations and provides the behaviors on a set of test cases. Configuration management includes systematical control of the changes to software as set forth in technical documentation, which also aims the maintenance of integrity and traceability of the software. Finally, software maintenance phase is the modification of existing software while preserving its integrity. It is performed in order to correct faults, implement enhancements, and adapt software to different hardware, software, or environments.

It is possible to classify software process models into prescriptive or agile process models. The focus of the prescriptive models is the detailed identification, definition, and application of process activities and tasks while agile models emphasize a more informal and flexible approach to software processes. It is important to note that there is no ideal or single best software process or a set of software processes. Therefore, an adopted process for a software project might be considerably different from a process adopted for another software project. The best of SE processes and models have to be selected, adapted, combined and applied for each project in an organizational context.

Software engineers have also to understand quality related concepts, characteristics, values, and their application to SE processes. As closely interrelated, the activities pertaining to SE process quality, product quality and software quality management have a direct impact on the quality of the software process and final product. Therefore, one of the primary responsibilities of software engineers and project managers is to balance the conflicting demands for project scope, time, cost, risk and product quality.

IV. Theory and Software Engineering

Theories are commonly viewed as a coherent set of tested propositions, which are generally regarded as correct, and able to predict or explain facts or phenomena in SE. A theory should have empirical foundations and be grounded on systematic collection of empirical evidences. As having potential use to practitioners and researchers, a SE theory provides a conceptual framework for explaining observed phenomena as well as it helps understanding the basic concepts and underlying mechanisms of software systems and their behaviors. Since SE is an applied discipline and each SE case is unique, a theory may need local adaptations, and it is expected to be relevant, formalized, and ultimately useful for the software industry.

A. Elements of a SE Theory

For the structure of a SE theory, it is suggested that the four main parts, such as (a) Constructs, (b) Propositions, (c) Explanations, and (d) Scope, may comprise the theory [2]. The constructs are the basic components in which a SE theory is expressed, and to which this theory provides a prediction or description. Propositions are formed by the relationships indicating how the constructs interact. The explanations, which are also experimental observations of propositions, are logical reasoning showing why the propositions are as specified. The circumstances, under which the theory is assumed to be applicable, define the scope of a SE theory. Thus, constructs and the relationships between constructs form the building blocks of SE theories, and they should be derived from or associated with four archetype classes: (1) Actor, (2) Technology, (3) Activity, and (4) Software System. An actor applies technologies to perform activities on an existing or planned software system in a typical SE situation [2]. The subclasses: industry, organization, team, individual and project, may extend from the parent class Actor. A programming language, tool, technique, method, model, and process can be the subclasses of class Technology. Planning, analyzing, designing, developing and maintaining a software system are the subclasses of class Activity. Finally, application domain, type of software and project subclasses may belong to the parent class Software System. These classes and subclasses, together with condition statements, define the scope of a SE theory, and for whom, where and when the theory applies.

B. How to Use a SE Theory

When supporting research studies, a SE theory helps to develop and combine research efforts, and it facilitates communication of knowledge and ideas. As to the industry, it can provide software decision-makers with required input regarding the selection of a method, tool or technology for a software project. There may be three modes of theory use in a SE research: (1st) using theories from other disciplines as they are, (2nd) adapting theories generated in other disciplines to SE, and (3rd) generating theories from scratch in SE discipline.

According to Gregor [3]; Hannay, Sjøberg and Dybå [4], there are also five types of theory that may be adapted to a SE context: (1st) the "Analysis" type of theories includes classifications, taxonomies, ontologies, and it describes object of a study and "what is"; (2nd) the "Explanation" type of theories explains why something happens or why a phenomenon occurs; (3rd) the "Prediction" type theories are used to predict what would happen by using probabilistic and mathematical models; (4th) the "Explanation and Prediction" type of theories is empirically-based, and they combine the basic features of explanation and prediction theories; (5th) Finally, the "Design and Action" type of theories are usually prescriptive and describe how to do things in a SE process as presented in Fig. 1.

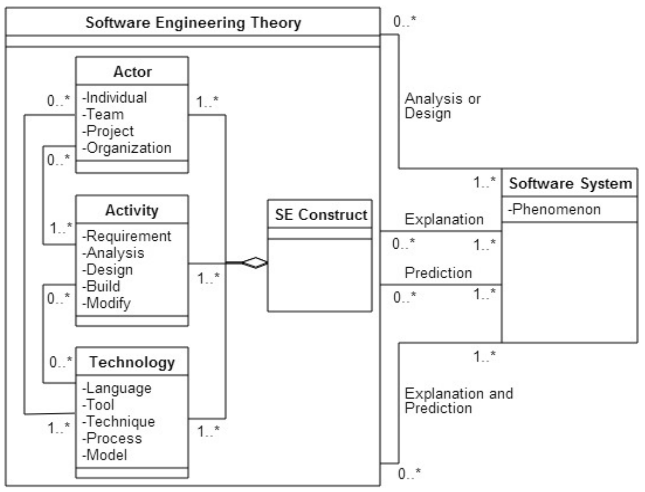

Fig. 1. Conceptual diagram for theory use in SE.

SE requires both empirical and theoretical research. Empirical SE studies explore, predict and try to explain the investigated cause-effect relationships between constructs of a theory, and find out what types of SE constructs, which are also derived from the archetype classes, should be used in what situations and circumstances. A SE experiment is, therefore, the primary research method directly to make comparisons and to observe the effects of measures taken to improve a SE process.

The research approaches for empirical studies should be based on quantitative, qualitative, or mixed research paradigms, which may also use experiment, case study, benchmarking, and standardization methodologies with statistical analysis techniques. To that aim, the roles that a theory would play in SE experiments may be: (1st) the design of an experiment for the hypotheses that may be justified by a theory; (2nd) a theory can be used for the explanation of observations on the cause-effect relationships after an experiment,; (3rd) a theory may be tested directly with an experiment when an additional justification is necessary; (4rd) an existing theory can be refined and enhanced, and then it needs to be tested; (5th) a theory may involve and provide structural elements as a basis for the use of another theory [4].

Finally, a SE theory provides a conceptual framework for organizing facts and knowledge, and it helps understanding the underlying mechanisms of software systems. It, at the same time, facilitates communication of ideas between research communities, which in turn, enables establishing the trans-disciplinary foundations of SE. Therefore, there have been also attempts to indicate that SE may need more general, discipline-specific, or unified SE theories as in some other disciplines [5].

V. Trans-disciplinary Foundations

SE has been usually perceived as a part of Computer Science providing basic computing theories and programming methodologies. However, the trans-disciplinary foundations of SE indicate that it requires not only the domain knowledge, but also understanding the theoretical essences that have close relationships with other disciplines (Fig. 2). Therefore, theoretical and empirical SE explores models, architectures and methodologies for large-scale software development as well as the nature and mechanisms of software behaviors and the laws behind them. Therefore, SE theories and methodologies are developed and adopted primarily for dealing with these challenges.

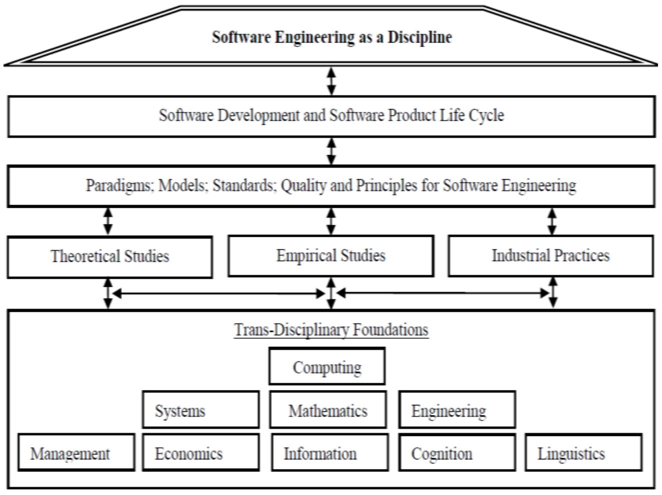
Software Engineering as a Discipline
Software Development and Software Product Life Cycle
Paradigms; Models; Standards; Quality and Principles for Software Engineering
Theoretical Studies
Empirical Studies
Industrial Practices
Trans-Disciplinary Foundations
Computing
Systems
Mathematics
Engineering
Management
Economics
Information
Cognition
Linguistics

Fig. 2. Trans-disciplinary foundations of software engineering.

The theoretical foundations of SE incorporate multi-facet principles, trans-disciplinary theories, and the knowledge acquired both from theoretical and empirical studies, industrial practices. For example, SE grew out of Computing Theory, and therefore, it is one of the most important foundations [1]. The fundamental models in computing can be classified into Program Modeling, Data and Object Modeling, Operational Modeling, Process and Resource Modeling. Computer Science provides the computational methods, computer architectures and implementations, computing objects and their abstract representations, and programing methodologies. It is possible to state that Mathematics is one of the top level abstraction means, and thus, it is the most general human knowledge in Science and Engineering. For this reason, SE uses it to express basic notions to form conceptual models, to design and treatment of architectures, to define abstract objects, relations, and software behaviors at the highest level of abstraction. The topics, such as Mathematical Logic, Set Theory, Functions and Relations, are the essential means to model software architectures and behaviors. Regarding the System Science Foundations, it is also seen that system concept is widely used in many disciplines of science and society, and may be classified into different categories depending on its key characteristics and components. Being highly complex abstract systems, software systems mostly adopt system theory in their life cycle. Thus, ISO/IEC 15288 standard [6] provides a process model that accepts the large-scale software development as part of software systems to help dealing with various problems and complexities in SE. While System Engineering is concerned with all aspects of systems development that may include process, hardware and software, SE may be part of this more general and abstract process.

Information is the product of natural or machine intelligence, and the Science of Information studies the nature of information and ways of its processing and transformation. Thus, the main function of languages is to communicate information and express abstract human thoughts and behaviors. In the same context, the Linguistics discipline studies natural languages, which are oral, written or symbolic system for communication, thought and self-expression. Therefore, software can be seen as a type of behavioral and instructive information about architectural and behavioral aspects of a software system, which mainly describes a solution for the design and implementation of a system. Programming languages describe and specify this computing and instructive information about architectural and behavioral aspects of a software system.

One discipline that has a strong tie with SE is the Cognitive Science, which also consists of multiple research areas, studies mind and its processes, explores how information is represented, processed, and transformed within brain and computing systems, and addresses the constraints on cognitive processing of information. Software is initially generated, represented and cognitively processed in long term and working memory of the brain before it can be transferred into a computer. Indeed, both problem-solving and SE have much in common such that they require high order cognitive skills and engage a variety of cognitive components. Therefore, acquiring computer programming skills and development of large-scale software systems are based on fundamental cognitive processes. The Information Processing Theory and Cognitive Load Theory can be given as the sample theories for this knowledge domain, which are also extensively used in many research studies of various disciplines.

Another example for the foundational disciplines is the Management Science, which primarily studies how an organization may be operated effectively and efficiently on given internal and/or external constraints and/or in different environments. It involves organizational theories, decision theories, operational theories, quality theories and strategic planning. Management is a process with the main functions i.e. planning, organizing and controlling, to achieve the goals that may not be possible by individuals. When tracing the history of SE, it can be been seen that many of the important concepts, such as requirement analysis-specification, design, testing and quality, were borrowed or adopted from the methods and practices developed in Management Science and other engineering disciplines. Therefore, in addition to computer programming and technical aspects of software development, SE also deals with the issues of organization and management of software related infrastructures.

VI. Conclusion

Experienced software engineers effectively use the fundamental knowledge of SE and perform problem-solving activities to look for solutions within limited resources, such as time, scope, and budget. As aforementioned, current SE industry still faces the major problems despite technological trends and innovations. Although it is claimed that the causes would be not only attributed to technology or practice, the studies reviewing the theory use in SE report the lack of explicit and/or relevant theory use in SE. Indeed, it is possible to state that the awareness for the importance of theory use in SE research community exists. However, the majority of the studies appear to provide post-hoc explanations of the results, or they justify research questions rather than being a theory-driven research study.

In this study, therefore, it is pointed out that SE already integrates theoretical and methodological perspectives drawn from other disciplines, and the theory use is an important mean to improve SE. Thus, software engineers and researchers should be not only concerned with technical issues, but also, conceptual and theoretical background of SE discipline. As a result, this paper attempted to offer a conceptual framework and brought a different perspective for understanding the theoretical and trans-disciplinary foundations of SE as a discipline.

References

[1] Y. Wang, *Software Engineering Foundations: A Software Science Perspective*, 1st ed. New York, USA: Auerbach Publications, Taylor & Francis Group, 2008.

[2] D. I. K Sjøberg, T. Dybå, B. C. D. Anda, and J. E Hannay, "Building theories in software engineering," *Guide to Advanced Empirical Software Engineering*, 1st ed., 2008, pp. 312-336.

[3] S. Gregor, "The nature of theory in information systems," *MIS Quarterly*, vol. 30, no. 3, pp. 491-506, 2006.

[4] J. E. Hannay, D. I. K. Sjøberg, and T. Dybå, "A systematic review of theory use in software engineering experiments," *IEEE Transactions on Software Engineering*, vol. 33, no. 2, pp. 87-107, 2007.

[5] SEMAT. Software engineering method and theory. [Online]. Available: http://semat.org

[6] ISO/IEC-15288. The standard for systems and software engineering-system life cycle processes. [Online]. Available: http://www.iso.org/iso/ catalogue_detail?csnumber=43564

Murat Pasa Uysal is an Assoc. Prof. Dr. at the Department of Computer Technologies in Ufuk University. He holds a B.S degree in electrical & electronic engineering from Turkish Military Academy, a M.S degree in computer engineering from Cankaya University, a Ph.D. degree in technology of education from Gazi University. He completed his post-doctoral studies at Rochester Institute of Technology in New York, which was on both software re-engineering and IT governance. He directed or served as an advisor and engineer for IT projects in Turkish Army (TA) for many years, and also conducted studies addressing the problems of TA in the research areas of IT. He has been teaching IT, computer and software engineering related courses. His research interest is in the areas of IT, software engineering, instructional methods and tools for computer programming.