Software Engineering Education for Significant Learning Experience

Eun Man Choi

Abstract—**Software engineering is not just a document or design, but a habit of the developers and a culture of the software industry. New teaching methods to prompt creative and practical thinking in software design have been introduced. However, an integrated approach to put a course goals, components together and formulating significant learning in software engineering education was not provided. This paper presents the taxonomy of significant learning in software engineering course and proposes curricula goals, with major components such as teaching and learning activities and feedback assessment. Evaluation, lessons and impact of the change indicate a significant learning achieved in software engineering education.**

Index Terms—**Significant learning, software engineering education, course design, software design studio.**

I. Introduction

Software is closely related to our lives and its influence on society is getting bigger. The way to grow the high added value economy is based on an important task of software. As applications grow and the need for software in everyday life grows, so does the need for more software. Direction of development of industry and the change of our life more depends on the development direction of computer and software. The development of AI and robot technology is also changing the framework of the employment society.

The demand for software is getting bigger, but the supply is not keeping up. The performance level of software engineering is not very satisfactory in delivery rate, cost compliance rate, defect density, productivity, quality cost.

One of the infra structure for the growth of the software industry is education. The outcomes of the software industry are the result of partial intellectual labor and determine most of the performance of human resources which are the main factor. Therefore, the key factor to success in software industry is education. We need to think deeply about what and how teach in software engineering education for significant learning.

Software engineering is not just a document or design, but a habit of the developer and a culture of the software industry. Therefore, it cannot be achieved by only lecture training or simple assignments. Various educational methods such as design studio [1], problem-based learning [2], flipped learning [3], [4], case study development [5], wiki introduction education [6], combined research and training [7], [8] and athletic approach [9] were tried in software engineering education.

Manuscript received March 25, 2019; revised September 12, 2019.

Eun Man Choi is with Dongguk University, Seoul, South Korea (e-mail: emchoi@dongguk.edu).

This paper propose the learning experience in software engineering education resulted in something that is truly significant in terms of the students' lives. Process and outcome dimension of significant learning are studied for software engineering education. We recognized taxonomy of significant learning, defined curricular goals, built components of course, and designed course structure and teaching activities.

Chapter II describes significant learning experience and related research. Chapter III presents the taxonomy of significant learning in software engineering course and learning activities. In Chapter IV, the proposed curricula goals, built components of course, and designed course structure and teaching activities are covered. Chapter V includes evaluation and lessons. Chapter VI includes conclusions and future research challenges.

II. Research Background

A. Software Engineering Education

The main purpose of software engineering education is to raise engineers who can develop good software in a word. To develop good software, students have to start with programming skills, understand the computer system, learn the development process, manage the actual project and people, and have business sense.

A well-documented core knowledge of software engineering is the software engineering curriculum based on IEEE SWEBOK [10]. SWEBOK is divided into 11 areas of knowledge and suggests topics and curriculum time in each area.

Carnegie Mellon University implemented software engineering education in graduate education based on SWEBOK Core Body of Knowledge (CBOK) [11]. The CBOK is divided into 11 topics: ethics and professional conduct, system engineering, software requirements, design, construction, testing, maintenance, configuration management, software engineering management, software engineering process, and software quality. The level of knowledge on each topic is specified using Bloom's proposed level of consensus [12].

- Knowledge of content - level of knowledge
- Understanding concepts - levels of understanding
- Applying what you have learned - the level you can apply what you have learned to a specific situation
- Analysis - the level of understanding the whole contents

of the learning contents

- Convergence - the level of understanding you use to create new ideas
- Assessment - the level at which the value of the learning content can be judged

TABLE I: KNOWLEDGE AREA OF SOFTWARE ENGINEERING

Knowledge Ares	Topics
CMP	Computing essentials: computer science foundations, construction tools,
FND	Mathematics, Engineering Fundamentals: logic, engineering fundamentals for software, engineering economy for software
PRF	Professional training: group dynamics, psychology, communication skills, professionalism
MAA	Model and analysis: modeling basics, types of models, basics of analysis
REQ	Requirement Analysis and Specification: requirements base, requirement elicitation, requirement specification and documentation, requirement verification
DES	Software design: concept, strategy, architecture and HCI design, detailed design, design evaluation
VAV	Software review and verification: V & V basics, reviews and static analysis, testing, problem analysis and reporting
PRO	Software processes: process concepts, process implementation, project planning and tracking, software configuration management, evolution processes and tasks
QUA	Software quality: quality concept and culture, process assurance, product assurance
SEC	Security: security foundation, computer and network security, security software development

Engineering is not a science that understands simple principles. In other words, it is the task of designing and applying software by applying the principle, so it is necessary to have the ability to integrate and synthesize to apply the understanding concept.

It is pointed out that only contents taught in computer science are difficult to handle the software development practice because of the lack of educational integrated approach required for significant learning. Computer science deals with natural phenomena and principles, but software engineering is the application of principles to the construction of software and institutional change of work process.

B. Significant Learning Experience

Learning experience in software engineering education needed to be resulted in something that is truly significant in terms of the student life. Not only will students be learning throughout the course, by the end of the course they will clearly have changed in some important way.

Significant learning [13] offers enhancing student work, enabling students to contribute to the many communities of which a student will be a part, and preparing students for the world of work. Significant learning should not be a content-centered course but a learning-centered approach to teaching.

Good courses for training software engineers are courses that challenge students to significant kinds of learning rather than something relatively insignificant. Significant learning course needs to use active forms of learning rather than passive learning provided by only lecture. Teachers in significant learning course care about the subject, students, teaching and learning and interact well with students. Significant learning courses have a good system of feedback, assessment, and grading.

Fink introduced major categories in the taxonomy of significant learning which is framework for formulating course objectives and a basis for testing student learning as Fig. 1.

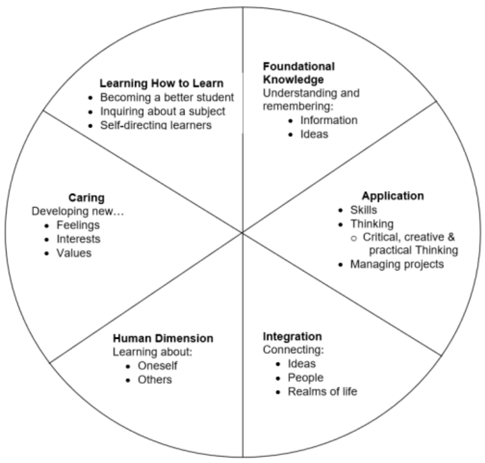

Fig. 1. Taxonomy of Significant learning [13].

Each category of significant learning contains several more specific kinds of learning that are related in some way and have a distinct value for the learner. This taxonomy of significant of learning is not hierarchical but rather relational and even interactive. Each kind of learning is related to the other kinds of learning and achieving any one kind of learning simultaneously enhances the possibility of achieving the other kinds of learning as well. For example, if a teacher finds a way to help students learn how to use the information and concepts in a course to solve certain of problem effectively(application) this makes it easier for them to get excited about the value of the subject(caring).

III. DEVELOPING LEARNING GOALS

Formulating course learning goals around significant learning for software engineering education has two implications. Learning goals for a course should include but also go beyond content mastery to make learning experience inherently more worthwhile and at the same time make more interesting for learners. Secondly significant learning goals will can create some interaction effects and synergy that greatly enhance the achievement of significant learning by students.

1) Foundational knowledge

- Have a mental map of software development process and be able to correctly understand design model and software artifacts.
- Understand major knowledge area – process, requirements analysis, modeling, design principle, UI/UX, design patterns, coding, testing, maintenance.

2) Application

- Be able to find information on and analyze problems in real world to build software system.
- Be able to use software tools and method in order to develop software effectively and efficiently.

3) Integration
- Identify the integration between software and other realms of application such as hardware. Business, economics, biology, administration, and so on.
- Identify the integration between requirements analysis, design, coding, testing, and maintenance.

4) Human dimension
- Be able to identify ways in which engineer's personal life affects and is affected by interactions with user, customer, and other stake holders in software development.
- Be able to intelligently discuss software development contracts with other people and the impact of software on business or application domain.

5) Caring
- Be interested in software development and maintenance and want to continue learning about software design, coding, testing, deploying and maintenance.

6) Learning how to learn
- Be interpret technical significance of new technology and ideas acquired in the future.
- Be familiar with a number of popular programming languages and design methods with tools.
- Have some specific idea about what else it would be desirable to know about software development.

One of the attractive features of a taxonomy is that it encompasses and integrates a wide range of methods and tools in software engineering on desirable kinds of learning.

IV. Designing Significant Learning Experiences

When teachers face the task of putting together a course, they use subject-centered approach to create a list of topics on it, and then proceeds to work up lectures on each topic. The alternative to this traditional, subject-centered approach is to take a learning-centered approach and put together systematically, in a process integrated course design.

The basic features of integrated course design are shown in Fig. 2. The box at the bottom, situational factors, refers to information that is gathered from research report by Samsung Economy Research Institute [14]. The gap between the technology required to develop software and the education provided by the university education has been pointed out several times in Korea as the curriculum reorganization project or the Seoul Accord project. Software engineering education is inappropriate to develop software development ability, business understanding and practical problem solving ability required by industry.

A. *Teaching and Learning Activities*

University education has traditionally relied on lectures. Most of the lectures are student-centered, content-oriented, and knowledge transfer-oriented. Naturally, learning depends on memory and stays in the relationship of the communicator rather than the personal relationship of the student and the teacher. Most of all, lecture education relies on evaluations that stay on individual learning and thus stimulate competition without interacting between instructors and students.

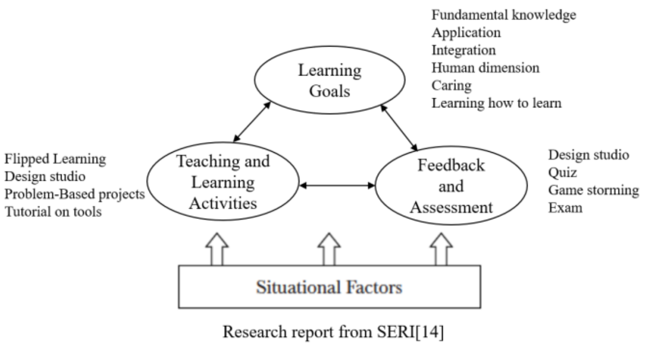

Fig. 2. Key components of integrated course design.

However, software engineering education is not sufficient for instructional lectures. A new educational paradigm is needed to complement lectures. Students and instructors must maintain deep relationships through interaction and exchange knowledge and experience. Cooperative learning should be done rather than individual learning. We need to build, inquire, and discover rather than facts, principles, and memory.

To promote significant learning, a new paradigm education method like design studio, game storming, flipped learning were accepted as teaching and learning activities.

- *Design studio* – a method of software design studio education which is utilized in the master program of software engineering of Carnegie Mellon University. It is a small design studio style education method widely used in design education such as architecture and fashion [15]. Educational features include professional, mentoring, and rigorous project management, with hands-on, medium-sized projects for training purposes. In other words, it is composed of team(customer, team member, technical writer, mentor, studio course staff) similar to the business and strictly manage the result. Results include development documentation(SOW, SRS, SDD, test plans), walkthroughs, demos, and more. Studio education also includes training on non-technical skills such as communication methods.
- *Game storming* – One good example of in-class activities in game storming [16] is post-it game brainstorming to find feature list. Post-it cards are given to all members of the team attended. They think about what features can provide. Then, each member of project team writes a feature to be included in new system onto post-it. After a certain amount of time to write system features on post-in are shared to colleagues by explaining their ideas. Good idea from another team during explanation can be added to their post-it description. Features after explaining the entire team gathered are pasted on board and organized by categories.
- *Flipped learning* – Traditional education is taught mainly in lecture rooms. Activities and exercises are conducted outside the classroom. The lectures are passive education focusing on listening, but active learning is done on

activities and practice. Now, using the Internet, lectures can be done outside the classroom [17]. Therefore, passive learning is a flipped-learning concept that draws active learning activities out of the classroom into the classroom, and is introduced as an inverted classroom. Most of the educational activities in flipped learning train creative activities to make software. For example, if you are teaching skills related to developing mobile apps, simply learning an iOS or Android programming language and a simple example cannot be an educational activity. These contents are provided by VOD in advance and learned, and in flipped learning, creativity is demonstrated, and new mobile app is cooperated and produced. Flipped learning can be done at low cost by making good use of MOOC (Massive Open Online Course), a video which is recently released at home and abroad.

TABLE II: LEARNING ACTIVITIES IN SOFTWARE ENGINEERING EDUCATION

Activities	Getting information and idea	Doing	Observing	Reflecting
Act01: Team building - MBTI - Case study for process	• Questionnaire • Process models	• Presenting MBTI • Discussion	• MBTI of team members	• Organizing project teams • In-depth reflective dialogue and writing on the process
Act02: Conceptualization - Project concept - Mission statements	• Lecture and textbooks • Brainstorming	• Case study • Writing SOW	• Success story	• Reflecting project planning
Act03: Post-it gamestorming - Brainstorming for feature list	• Survey related domains	• Writing post-it and circulating	• Priority of functions	• Reflecting SRS(Software Requirements Specs)
Act04: Use-case analysis - Write use cases	• Sample Use Cases and template	• Drawing use case diagram • Writing use cases	• User interactions	• In-depth reflective dialogue and writing on modeling
Act05: Class diagram design studio - Pick up class candidates - Identify relationships - Draw class diagram	• Use cases • Domain-driven analysis	• Drawing class diagram with UML tools	• System structure and components	• In-depth discussion about good design and improvement
Act06: Sequence diagram design studio	• Use cases, class diagram	• Drawing sequence diagram with tools	• System interaction	• In-depth discussion about good design and improvement
Act07: Finding quality characters - Applying design principles	• Performance requirements	• Analysis of tradeoff in architecture	• System bottle-neck with simulation	• Reflecting architecture design
Act08: UI/UX Design studio - UI design	• Use cases	• Drawing menus and controls in UI	• User's perspectives to system	• In-depth reflective dialogue for user friendly UI
Act09: Applying design patterns - Detail design	• Lecture and text book	• Modifying class diagram with patterns	• Design quality for easy extension and modification	• Reflecting detail designs
Act10: Writing coding standards	• Samples of coding style	• Defining coding style and applying coding	• Code readability	• Reflecting source code
Act11: Writing test cases for equivalence partitioning	• SRS	• Writing test cases • Testing implemented system	• Defects of source code	• Assessment of software quality
Act12: Presentation of project results	• System designs • User experience	• Prepare demo and slides	• Usability • Process quality	• Assessment of project results

We create the kinds of learning activities capable of achieving significant learning experience. Table 2 illustrates new conceptualization of active learning that makes all three modes of learning such as experiences, information and ideas, reflecting in software engineering education.

B. Feedbacks and Assessment

The third component of a significant learning course is feedback and assessment. Traditional midterms and final are not enough to measure the performance of various learning activities in software engineering education. Those serve only one function to audit student learning as a basis for the grade turned in. We added more educative assessment such as forward-looking assessment, self-assessment with criteria and standards(CMMi) and feedback in design studio. During the learning activities such as design studio, learners engage in software design activities in an effort to learn how to do

UML modeling, apply design patterns, learners should each be getting feedback to help them understand what are the faults in their design and implementation and how to improve the quality of their work and software artifacts.

As indicated in Fig. 3, educational assessment four components: tutoring, self-assessment, feedback, criteria and standard. In design studio students provide presentation after self-assessment. Lecturer, peers, tutors give feedback with tutoring and finally give grades.

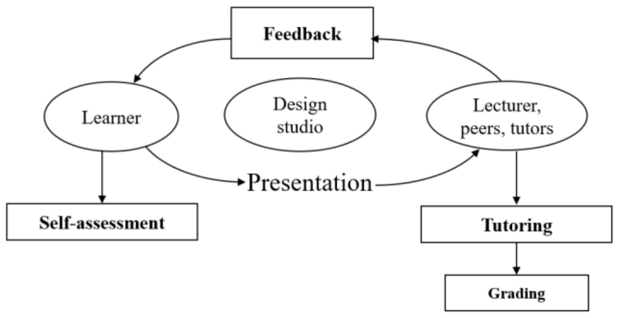

Fig. 3. Educative assessment model.

C. Integrating

The final step in the course design process is to make sure the main components are properly integrated. This means we need to check the four components to be sure that they support and reflect each other. Figure 4 shows that three major set of course decisions consistent with the information gathered about the situational factors.

TABLE III: INTEGRATE THE PRIMARY COMPONENTS

Learning goals	Learning Activities	Assessment
1. Understand and remember key concept, terms, relationship	Act03, Act04, Act05, Act06, Act08, Act11	Self-assessment, tutoring in design studio, Grading
2. Know how to use the content	Act02, Act04, Act09	Exam
3. Be able to relate software engineering to other subjects	Act02, Act07	Gamestorming
4. Understand personal and social implications of knowing about the software engineering	Act01, Act03	Peer review
5. Care about the software development	Act10,	Feedback Software quality tools
6. Know how to keep on learning about software engineering after the course over	Act12	Questionnaire

V. EVALUATION OF COURSE AND TEACHING

Based on course operation for significant learning, lessons can be summarized as follows.

- *The value of dialogue* – Being able to talk with someone else when working on software development is very powerful. The value of dialogue about gamestorming, design studio comes part from the opportunity to engage in dialogue with someone who will use software and varied expertise.
- *The value of focusing on the significant components of software design* – It is good chance for students to prompt non only critical thinking, but also creative and practical thinking in software development.
- *The value of training on effective development process* – Flipped learning classroom has advantage of more opportunity for students to interact with. Even the most interactive lectures are likely to actively involve only a software development process. Most of the class time in flipped learning classroom model can be spent with these students that are struggling, as opposed to the traditional lecture where most of the questions posed during software process come from real world problems.
- *The need for monitoring learning and tutoring* – The big breakthrough in this case, came from clear monitoring of active learning and educative assessment. A significant improvement in software engineering education allowed a synergistic improvement to take place in software process with leading-edge practices.

VI. CONCLUSION

Good teaching cane be used to foster better learning [18]. The taxonomy of significant learning offers a special kinds of course that students might learn in software engineering in a way that is capable of software development skills required from software industry. The model of integrated course for software engineering is designed and applied in university education. Significant learning metaphor is a tool that enables instructors to support and promote active learning. The case study shows the positive outcomes because it incorporates and organizes several existing and potent ideas about teaching in software engineering, for example, flipped learning, problem-based learning, gamestorming, software design studio.

CONFLICT OF INTEREST

The author declares no conflict of interest.

AUTHOR CONTRIBUTIONS

Conceived and designed the analysis; Collected the data; Performed the analysis; Wrote the paper are all the author(Eun Man Choi)'s contribution.

REFERENCES

[1] J. Tomayko, "Teaching software development in a studio environment," *ACM SIGCSE Bulletin*, vol. 23, no. 1, pp. 300–303, 1991.

[2] I. Richardson and Y. Delaney. "Problem based learning in the software engineering classroom," in *Proc. 22nd Conference on Software Engineering Education and Training*, pp. 174-181, Feb. 2009.

[3] G. Gannod, J. Burge, and M. Helmick, "Using inverted classroom to teach software engineering," in *Proc. ICSE 2008*, pp.777-786, 2008.

[4] E. M. Choi, "Applying inverted classroom to software engineering education," *International Journal of e-Education, e-Business, e-Management and e-Learning*, vol. 3, no. 2, pp. 121-125, 2013.

[5] D. Dahiya, "Teaching software engineering: A practical approach," *ACM SIGSOFT Software Engineering Notes*, no. 2, pp. 1-5, 2010.

[6] K. Parker and J. Chao, "Wiki as a teaching tool," *Interdisciplinary Journal of Knowledge and Learning Objects*, vol. 3, pp. 58-72, 2007.

[7] L. Johns-Boast and S. Flint, "Providing students with 'real-world' experience through university group projects," in *Proc. Australian*

Association for Engineering Education Conference, pp. 299-304, 2009.
[8] M. Gehrke, H. Giese, E. Kindler, and J. Niere, *Software Engineering Education: The Synergy of Combined Research and Teaching*, Paderborn, Germany, Jan. 2003.
[9] E. Hill, P. Johnson, and D. Port, "Is an athletic approach the future of software engineering education?" *IEEE Software*, January/February, pp. 97-100, 2016.
[10] *Software Engineering 2014, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*, IEEE ACM, 2014.
[11] M. Bass, "Software engineering education in the new world: What needs to change," in *Proc. IEEE 29th International Conference on Software Engineering Education and Training*, pp. 213-221, 2016.
[12] B. Bloom, "Handbook 1: Cognitive domain," *Taxonomy of Educational Objectives: The Classification of Education Goals*, New York: Longman, 1956.
[13] L. Dee Fink, *Creating Significant Learning Experiences*, Jossey-Bass, 2003.
[14] Samsung Economic Research Institute, *Challenges of Korean University Education for IT Human Resource Development*, 2011.
[15] R. Bareiss and M. Rosso-Llopart, "Software engineering education at Carnegie Mellon University: one University; Programs taught in two places," *Systemics, Cybernetics and Informatics*, vol. 7, no. 5, pp. 72-77, 2009.
[16] D. Gray, S. Brown, and J. Macanufo, *Game Storming: A Playbook for Innovators, Rulebreakers, and Changemakers*, O'Reilly, *2010.*
[17] E. M. Choi, "Applying inverted classroom to software engineering education," *International Journal of e-Education, e-Business, e-Management and e-Learning*, vol. 3, no. 2, pp. 121-125, April 2013.
[18] P. Palmer, *The Courage to Teach: Exploring the Inner Landscape of Teacher's Life,* San Francisco, Jossey-Bass, 1998.

Eun Man Choi was born in Seoul, S. Korea in 1958. He received the B. S degree in computer science from Dongguk University, Seoul in 1982 and the M. S. degree from KAIST in 1985. He has got the Ph. D. degree in computer science from Illinois Institute of Technology, Chicago in 1993. Currently he is a full professor in Department of Computer Science and Engineering in Dongguk University. His major field of study is software engineering, specially software testing. He was on the technical staff of computer research center working on the computer code standards for Korean characters and software standards for Korea Institute of Standards and Science like NIST in U.S.A. He developed an experimental Ada maintenance environment based on a syntax-directed editor for supporting Program Understanding. He constructed a number of CASE tools with an emphasis on integration mechanisms of tools and Object-Oriented Software Metrics. Prof. Choi is senior member of Korea Information Science Society from 1985 and a member of governing board in SIG Software Engineering, Korea Information Science Society since 1998. He was a member of editorial board, Journal of Korea Information Processing Society since 1997. He was a visiting scholar in department of computer science of Colorado State University in 2000, 2007 and Baylor University in 2014.