Improving Understandability In Teaching Of Software Engineering And Connectivity With The Industry

Izzat Alsmadi and Bilal Abul-Huda
Department of Computer Science and Information Technology
Yarmouk University
Irbid, Jordan
ialsmadi@yu.edu.jo , abul-huda@yu.edu.jo

ABSTRACT— **As a new field of study, software engineering teaching and subjects vary from one textbook to another. Despite the fact that most of the books cover similar subjects, however, students' view of the subject is mixed. Some students have problems understanding the entire picture. Other students have problems connecting concepts with each other. In this research, an overall view of software engineering knowledge is presented. The knowledge is presented from four perspectives: Process, Project, People and Product. Those four are usually referred to as the 4Ps in literature. The goal is to make a distinction between the progresses in each area and explore the opportunities in finding windows for more research in any of those four views.**

Researches in this field in many published articles appear to ignore the state of the studied field in the industry and focuses on its state in the academic arenas. Related work in research papers focus on those research papers published and do not look in company websites, web logs, discussion boards, etc.

Teaching software engineering should combine interactive methods of teaching besides the traditional class room teaching. Without such methods, the majority of the sought benefits and expected skills to learn may not be practical. Academic curriculums and research papers should give more attention to the industry and its current technologies. This can help students in their future jobs. It also helps the industry utilizing such researches once they become more realistic or relevant.

Keywords- Software engineering, ontology, separation of concerns, project, product, process and people, CASE tools, software development methodology.

I. INTRODUCTION

In teaching software engineering courses to students, it is noticed that some students complain from the lots of models that they need to know without having to

know the overall picture first. The same problem existed in another related field; formal methods. There are several formal method tools and sectors to learn without having an overall ontology that illustrates the connections between those tools or methods.

An ontological study simplifies the structure of understanding domain knowledge. It includes abstraction, representations, and assumptions. An ontology or a conceptual model facilitates communication or knowledge sharing on common grounds. In any ontology, abstractions are selected to focus on necessary components and eliminate irrelevant details. By large, software engineering field of study is lacking a comprehensive ontology that covers the overall knowledge body.

There are some other alternative views for the subject: software engineering development methodologies that usually get the main focus and concern. Other concerns can be divided into the dimensions; data; that focuses on entities, functional; which is concerned with functions or services, user and environmental views. This separation of views is important for many areas. For example, in software project planning and evaluation, managers need to plan and evaluate for each one of those dimensions separately. Similarly, in software design, designers draw different diagrams for the different views: class, activity, use case and sequence diagrams.

Out of the four views listed above, the software process and project views are the two that have the major focus, documentation and models in literature. Nevertheless, they are tightly coupled that makes it hard to distinguish whether this is a process or project attribute. Part of this confusion is understood since the software project management can be seen as a software process or activity. However, we should differentiate between software processes that are product oriented such as requirements, design and coding or development, and the processes that are people oriented such as the project management, personnel selections, and tasks distribution.

There are numerous software process models proposed to adopt while building a software. Those models can be classified according to the aspects and properties that distinguish each view. The software process is the set of activities and methods employed in the production of a software. The software project is how we manage and organize the process, the people and the product. This view makes the software project the main view that includes the three other views. The software product is the ultimate goal for the project, process and product. Software project success is measured through the product success. A successful project, people and process should result in a successful product. The people are the real and main resources in software projects. Without talented and dedicated people, we can't have a successful project and product. This shows that the four views are highly coupled and depending on each other and that we have to guarantee each ones success to guarantee the overall success. Figure 1 shows an overall ontology for software engineering concerns.

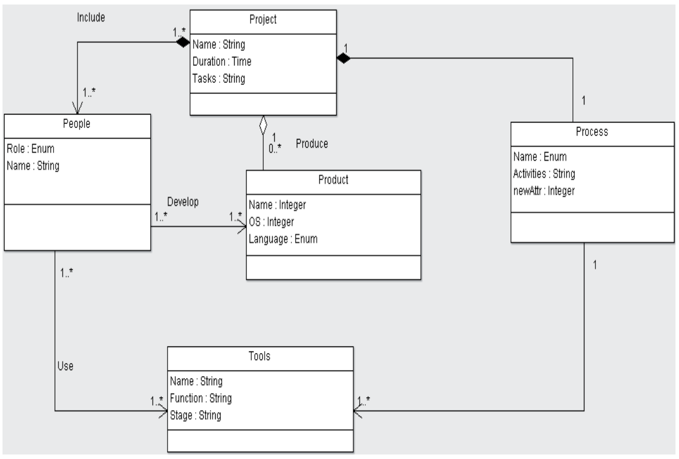

Figure 1. An ontology for software project concerns

In the following sections, the four dimensions are considered separately.

II. Literature Review

There are currently several known books that are usually used as textbooks in teaching software engineering [1,2, 4,5,6,7,8,9,10,11,12]. Those books vary in their presentation of the subject. There are some subjects such as formal methods or agile developments that get more focus on some books relative to others. Those two subjects usually divide teaching software engineering into two schools. A school that focuses more on traditional methods of teaching with presenting techniques that improve the correctness of the requirements and design in early stages of development. The other school following agile methodologies and their focus on the time and the flexibility factors relative to other factors such as correctness and quality.

The other part of the literature review are papers who discussed the issue of teaching software engineering and its different views [3,13,14,15,16,17,18]. Habra proposed two software engineering modules for an undergraduate education with focus on separation of concerns [3]. He defined the following software project dimensions to be considered by students: data, functional, user, reusing, and distribution. Hawker presented a model that combines: product, process and people elements [13]. He drew several UML diagrams presenting the different elements in the model along with their interaction with each other.

Several papers discussed problems, issues and difficulties in software engineering education [25,26,27,28,29,30, 31, 32, 33, and 34]. The majority of the subjects discussed in those papers will be investigated in this paper. To utilize effectively space in this paper, those issues will be discussed throughout the paper without further details in this section.

III. Software Processes; Activities and Models

As mentioned earlier, out of the 4P dimensions, the process is the one that has most of the existed literature or documentation. Software processes are the activities involved in producing and evolving the software. Examples of some of those major activities include; requirements gathering and specifications, software architectural and design, software implementation, testing and maintenance or evolution. We have to differentiate between software processes and software process models. Software process models are abstract representations for the models and their interaction. Similar to abstractions, models involve focus on particular concerns or perspectives. As a result, each model has certain scenarios that can be best used in. For example, the water fall process model is used when we have fixed and stable requirements. On the contrary, agile methods are better when we have uncertainties in the project.

The difference between the different models is largely depending on how the processes interact with each other (that's why they are process models). There are two possibilities on how processes can interact with each other.

A. **Straight forward processes**. In those models, each major software activity is completed first before moving to the next process. Once a process is completed, we can't go back and modify it. The largely known and used model in this type is the Waterfall model. Waterfall is used when we have stable and fixed requirements as it can hardly deal with or accommodate changes.

B. **Iterative or evolutionary processes.** In those models, major activities are completed partially in cycles and evolve to reach the final product. The goal is to deal with the instability of requirements and the need to accept and accommodate changes. With the exception of the waterfall model, all other software process models, such as the incremental model, spiral model, prototyping, and agile models, are examples of the Iterative models. Some models iterate through all process activities, others gather all requirements, then iterate through the rest of the activities. Spiral models make explicit risk assessments in every cycle, agile models combine iterations with project and people techniques to include abilities to deal with evolution and accept changes.

There are some software engineering books who consider some other software process models such as formal specification or Commercial Off-The shelf Software (COTS) as process models. The usage of formal specification or offshelf software can be in any of the previously mentioned models and need not to be a separate model. For example, formal methods are used to verify requirements formally before starting the construction process. This is a specific extra activity that is added to the requirement stage.

IV. PEOPLE, THE SOFTWARE PROJECT RESOURCES.

A software process model is an explicit description of the process through which software artifacts (or products) are implemented. Most of those process models focus on the products, time or goals as the factors that control the tasks for each stage of the model. In real environment the employees are the main resource and in each stage each employee, should have a specific task. Many of those current software process models do not consider some scenarios where a company may start a new project and each team or team member, whether from the business analysis's team, from the development, testers or from the document writers, is expected to do some work at any development stage.

Humans play an important role in the success of software projects. Communication is very important to ensure all team collaborations and contributions. Some methods, such as agile development models are customer oriented where the customer suggestions and concerns are always considered.

In most of the traditional software development models, the focus is in achieving the goals or the requirements that the application is expected to fulfill. In agile methodologies, the time is more sensitive giving the fact that requirements and/or many other factors may change with a relatively short time.

Although those models target most of business scenarios in developing projects, in some cases we may have all company employees required to work simultaneously. Following for example a traditional approach in that case, requires developers to wait or take a vacation if they don't have other projects to work on till business analysis's team finish colleting the requirements. Testers are also expected to wait longer time waiting for developers to design or implement the product or part of it. In a small business environment, this is expected to happen specially when there are no earlier versions of the application that those testers or document writers can work on through early stages.

Software project managers define tasks by roles, or individuals. It is better, however, to define project tasks by roles only. If the individual got busy in any other task, any other individual in the same class or role can be assigned the task. New software process methodologies such as Scrum, tries to mix software engineering roles to make all individuals capable of working in the different roles in different times. In reality, few people can perform all types of tasks professionally.

V. TOOLS, THE PEOPLE HELPERS

Tools play a major role in software engineering processes and development. In several cost estimation models such as COCOMO, the amount of assistance tools gave to developers is an important information needed to estimate the development time. They can noticeably improve the overall productivity of the team members.

Tools (also called Computer Aided Software Engineering, CASE) can be classified in several ways. They can be classified according to the software process stage they are working on (e.g. requirement tools, design tools, coding tools, etc). They can also be classified according to the number of stages they are working into tools, workbenches and environments that start from an individual stage to tools that support all software engineering activities.

VI. THE SOFTWARE PRODUCT, THE GOAL OF THE SOFTWARE PROJECT.

The software product is the ultimate deliverable or output that the team will produce. Any project's ultimate goal is to provide a product with the right functionalities and qualities. However, some projects may not have a deliverable product; instead, they will have objectives to fulfill. No matter how much successful the team was, or what tools, or techniques they used, if the product fails, all will be considered so. This means that logically successful project, process and people should produce a successful product. However, this is not always the case.

In the product teaching section, COTS can be introduced as a subject, quality assurance and software metrics are two other main subjects to be covered. This section will also focus on classifying products according to their business domain. Products that share same domain are expected to have several common characteristics.

VII. THE SOFTWARE PROJECT; THE UMBRELLA THAT COVERS ALL VIEWS

The software project is the umbrella that holds all the earlier concerns together. Project management takes care not only of the people, or resources of a project, but it also deals with the process and the product management. Project management major activities include: preparing the feasibility study, tasks' planning, allocating and scheduling, cost estimation, evaluation and measurements, risk assessment, and management, etc.

VIII. SOFTWARE ENGINEERING EDUCATION WITH INDUSTRY ORIENTATION

One of the early papers that discussed software engineering education is that of Mills in the early 80s [20]. He discussed some of those early challenges and requirements that faced education in the new field. Later on, in a paper he published in 1988 [21] he acknowledged that the software engineering education may vary depending on the degree and university requirements, industry needs and expectations, and many other factors that may eventually route the software engineering education to its ultimate destiny. He indicated that (even in the late 80s), the problem is not with the lack of technologies, but with the difficulties or problems of education management.

Software Engineering Body Of Knowledge (SWEBOK) is a general manual or guide from IEEE Software Engineering experts started in the late 90s on the general knowledge and information about this field [22]. In 2004, SWEBOK define the following knowledge areas in the software engineering field: Software requirements, Software design, Software construction, Software testing, Software maintenance, Software configuration management, Software engineering management, Software engineering process, Software engineering tools and methods, and Software quality. This work was an important document that contributed to the knowledge and education in this field, however, known researchers such as Cem Kaner and Grady Booch believed that such document needs a thorough reevaluation process.

The compromise between the theoretical and the practical teaching or education was always an issue of difference between curriculum or course contents. Another related issue is the ability to bridge the gap between the academia and the industry. Many progresses in this field are out of synch if we compare it between the academia and the industry. Universities are then in a challenge whether to produce good work force members or good researchers.

Sjoberg et al suggested several proposals to improve the coordination between the industry and the academia in the computer science field [23]. An example of those suggestions is the increasing research funding from industry and training software engineers in conducting more empirical studies with organization-specific goals and high quality.

Another related proposal to improve the connectivity between the academia and the industry is in the possibility of using the industry as a laboratory, instead of studying inexperienced students performing small programming tasks [24]. This is applied in many universities and proved to be beneficial for all parties; students, universities and the industry.

The major problem for computer and IT major graduates is always to find a job once graduated and to relate what they studied with the industry needs. Students understand that the degree itself is nothing but an initial requirement to a good job in the market.

In order for a computer student graduate to have a successful career, he/she needs to train themselves with different types of skills that they may not get them professionally depending solely on course contents. Examples of such skills include: databases, programming, technical support, networking, web design and graphic design.

IT fields are continuously evolving areas of knowledge. University curriculum in some cases lags behind the industry and the current technologies used by software companies. The research advancement in the industry is usually ignored by the people in academia. Related work in research papers focus on those research papers published and do not look in company websites, technical reports, web logs, discussion boards, etc.

A more proactive partnership is proposed between the Information Technology (IT) faculties and the IT industry. This is a summary of the sought benefits from having such partnership:

1. For students:

As students are the most important element in the proposal, and as the improvement of their skills and readiness to the industry is the ultimate goal of this project, the sought benefits for students from this partnership can be summarized by:

- Rather than working on hypothetical projects through their courses, students can work on actual projects from IT companies in the market. This give them more interest in the project they are working and encourage them to put more time and effort on those projects.
- As students will work on actual projects, they can relate the theoretical knowledge they learned through courses with the practical knowledge they learned through those projects.
- Students through those projects will be able to build professional relations with local IT companies. This is a major benefit fresh graduate students will need to be able to enter the highly qualification demanding industry. Through working hard on those projects, students can prove themselves and build their own relationships which will make it easier for them to enter the IT industry whether with this specific company or anyone else.
- Students can have professional experience and projects in their resume to include. An empty resume that has nothing but a degree and probably some courses, will be very hard to compete in a very high competitive market.

2. For IT faculties Universities:

- For instructors, an enthusiastic student will give them more interest on teaching and will allow them to show the students the benefits of the theoretical knowledge.
- Instructors don't needs to keep creating hypothetical projects. Projects will come from the industry partners. The selected partners will assist in the evaluating and grading of the project as their feedback on the student performance on the project.
- The IT faculty using this new trend is expected to raise its stake among other IT faculties and promote better position for it in comparison with its competitors. IT graduates like to have a mix knowledge in their university study and like a faculty or a university that can accelerate their ability to gain skills.
- Through partnership with industrial companies, the IT faculties will ensure its continuation to be at the edge of current trends and technologies in the industrial IT fields.

3. For the IT companies in the private industry:

- Usually, companies will not pay students for working on those projects. As such, the main benefit for companies is that they are getting work on their projects for free. The only overhead on this is the communication with students.

Companies usually assign a tech support coordinator to follow up with students regarding their enquiries about the project.

- Companies will have the chance to meet and know students for better future employment candidates. They can have a period and evaluation period without the need to hire those students for sometime and see how they will be doing. Usually companies lose money for incorrect selections of new employees. In those projects, companies can have the time to evaluate and select best students who performed well in their assigned projects to give them job offers in future.

Software engineering courses such as those of requirement, design, construction and testing should have practical or projects parts where students will be asked to select a project from a candidate company.

IX. Teaching Methodologies

Class traditional teaching methods may not be effective enough to be used in software engineering subjects' education. It should be combined with several other methods such as: interactive discussions, lab presentations and experiments, experimental projects or projects through actual business partners, etc. Several universities around the world due to several reasons , largely convenience, are focusing teaching methods on traditional class education.

The previously described proactive partnership with the IT industry can be an important contributor in improving software engineering teaching methodologies. Accreditation of a degree in software engineering should enforce the need to add methods other than those traditional ones. Through 3 years of teaching software engineering to undergraduate and graduate students, we noticed the positive effects in education and understanding as a result of injecting interactive lab sessions and experimental and practical projects. However, there are several types of obstacles and difficulties. Students usually have problems working in a team and learning how to evenly distribute tasks. On the other hand, some students may have problems accepting other team members' opinion. Students also tend to complain from working in projects in general due to the fact that they require extra efforts and usually learning new skills. Instructors may also tend to avoid projects' and labs' sessions especially in universities where they are not giving teaching assistants for supporting. On the other hand, grading lab and project homework assignments are not simple as they are unstructured. Those are all difficulties or obstacles that should be tackled subjectively when planning for software engineering courses.

X. Conclusion and future work

The goal of this separation of concerns is to organize the software engineering project into smaller manageable parts that can be easy to understand. It should reduce complexity and improve clarity. This concept is at the core of software engineering. The 4Ps concerns have some overlapping and distinct features. Concepts such as; ontology, abstraction, modeling and views or separation of concerns always include some sort of abstraction or focus. The goal is to draw a better image or understanding for the problem. The goal of the separation of the concerns in software engineering projects is to improve the understandability and consider only relevant properties for each perspective.

In another goal, we hope that the separation of concerns will help software engineering students better understand the large number of modeling and terminology concepts that may overlap and hence seem ambiguous.

This paper suggests a methodology to teach software engineering on the basis of the different perspectives or view. Such views are expected to develop and overall conceptual understanding that seems to be missing for many students who learn introductory software engineering courses. We described those different views in brief to proof the concept. As this is a suggestion for a book or a course, it should include more details and elaborations. We will introduce all software engineering terms, and concepts in terms of this view. In software project managements, managers need to separate their planning and evaluation among those four perspectives. Students should also differentiate between tools, concepts and standards used for each one of those views.

Academic curriculums and research papers should give more attention to the industry and its current technologies. This can help students in their future jobs by training them on the tools used by the industry to obtain the needed skills. It also helps the industry in utilizing such researches once they become more realistic or relevant.

XI. References

[1]. Ian Sommerville. Software Engineering (Seventh Edition). Addison-Wesley, 2004

[2]. Roger S. Pressman. Software Engineering: A Practioner's Approach (Sixth Edition, International Edition). McGraw-Hill, 2005.

[3]. Habra, Naji. Separation of concerns in software engineering education. ICSE 2001.

[4]. Jessica Keyes. Software Engineering Handbook. Auerbach Publications (CRC Press), 2003.

[5]. Hans van Vliet. Software Engineering: Principles and Practice (Second Edition). Wiley, 1999.

[6]. Timothy C. Lethbridge & Robert Laganière. Object-Oriented Software Engineering: Practical Software Development using UML and Java (Second Edition). McGraw-Hill, 2005.

[7]. Andrew Hunt, David Thomas. The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley Pub Co, 1999.

[8]. Barry Boehm. Software Engineering Economics. Prentice Hall, 1982.

[9]. J. Fairclough (Ed.). Software Engineering Guides. Prentice Hall, 1996.

[10]. C. Mazza. Software Engineering Standard. Prentice Hall, January 1995.

[11]. Alan M. Davis. 201 Principles of Software Development. McGraw-Hill, 1995.

[12]. I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Reading, MA, 1998.

[13]. J. Scott Hawker. Integrating Process, Product, and People Models to Improve Software Engineering Capability. <http://cs.ua.edu/research/-TechnicalReports/TR-2002-05.pdf>. 2002.2010.

[14]. C. Wille, A. Abran, J-M Desharnais, R. Dumke, The Quality concepts and sub concepts in SWEBOK: An ontology challenge, in International Workshop on Software Measurement (IWSM) , Montreal , 2003 , pp. 18,

[15]. C. Wille, R. Dumke, A. Abran, J-M, Desharnais, E-learning Infrastructure for Software Engineering Education: Steps in Ontology Modeling for

SWEBOK, in Ontology Modeling for SWEBOK , in Software Measurement European Forum , Rome, Italy , 2004

[16]. Software Engineering Research Laboratory. Institute of Electrical and Electronics Engineers, Inc. <www.swebok.org>. 2008. 2010.

[17]. Framework and Tool for Modelling and Assessing Software Development Processes. J. G. Doheny and I. M. Filby AIAI-TR-204. August 1996.

[18]. Goldberg, A., and Pope, S. T. 1989. Object-oriented is not enough! American Programmer: Ed Yourdon's Software Journal 2(7): 46-59.

[19]. Dongsun Kim, Suntae Kim, Seokhwan Kim, and Sooyong Park. Software Engineering Education Toolkit for Embedded Software Architecture Design Methodology Using Robotic Systems. 15th Asia-Pacific Software Engineering Conference. 2008.

[20]. Software Engineering Education. Harland D Mills. Proceedings of the IEEE, Vol. 68, NO. 9. 1980.

[21]. Strategic Imperatives in Software Engineering Education. Harland D Mills. Proceedings of the SEI Conference on Software Engineering Education. 1988.

[22]. Guide to SWEBOK, 2004 version. Alain Abran, James W. Moore. IEEE Computer Society. 2004.

[23]. Sjoberg, Dag, Tore Dyba, and Magne Jorgensen. The future of empirical methods in software engineering research. Future of Software Engineering, 2007. FOSE '07**.** 2007. Page(s): 358-378.

[24]. Sjøberg, D.I.K., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A. and Vokac M. Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments. In: Conradi and Wang (eds.) Empirical Methods and Studies in Software Engineering: Experiences. Springer-Verlag LNCS 2765, Pages: 24-38, 2003.

[25]. Anthony Finkelstein, Student Pmblems in Software Engineering Education, IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[26]. C J Theaker. D C Evans, Croup Projects in Software Engineering Degrees, IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[27]. P A Grubb, Follow up to Undergraduate Software Engineering Projects, IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[28]. Francis Tam, Fred Pran, Duncan Shonland, Engineering Software with Software Engineering: Formal Design to implementation, IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[29]. Juliet Brown and Martin Loomes, Humane Mathematics for Software Engineers, IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[30]. John Barrie Thompson, The use of a quality assurance tool in the teaching of software engineering principles. IEE Colloquium on Teaching of Software Engineering-Progress Reports, 1991.

[31]. Carlo Ghezzi, and Dino Mandrioli, The challenges of Software Engineering Education, ICSE05, 2005.

[32]. Melody Moore and Colin Potts, Learning by doing: Goals and experiences of two software engineering project courses, Software Engineering Education Lecture Notes in Computer Science, 1994, Volume 750/1994.

[33]. David Evans, Teaching Software Engineering Using Lightweight Analysis, CCLI01, 2001.

[34]. Pankaj Jalote, Teaching an Introductory Software Engineering Course in a Computer Science Program, ASEET09, 2009.