Engineering an Academic Program in Software Engineering

Dr. Akram I. Salah
Department of Computer Science
North Dakota State University

258 IACC,
Fargo, ND, 58105-5164
Email: akram.salah@nsu.nodak.edu

Abstract

Software industry is growing and maturing through the last years and expected to continue, even with a higher pace. Consequently, the need for software engineers is growing and the need for the industry to adopt and apply software engineering concepts within their development process is a must. The availability of qualified software engineers is limited. Reflecting on this problem, and adds to its requirements, are the official steps taken to define license for software engineers and restrict some practices to those licensed personnel. Such a license already started in Texas and it is expected that other states would do the same. The announcement of IEEE of offering a professional certificate exam is a step on the same path.

For many, there is a mix up between software engineering and programming. Many education and training programs mix those terms. It is the role of universities and colleges to provide educational programs, first, to graduate software engineers on various levels and for various aspects of the profession. Also, it is their role to provide information, within other courses, on software engineering to make computer, and non-computer, professionals be aware of the tasks and objectives of software engineering and on the software development industry and the role of various professionals in it.

The computer science department at NDSU realized this and started many efforts to provide educational programs in software engineering and to enhance the software engineering components in regular courses. Efforts are going in the undergraduate level, both breadth and depth, and on the graduate level. Among those is a program that provides a graduate certificate in software engineering and a master degree in software engineering. The certificate is directed to professionals in the software development industry to gain more knowledge and skills to become software engineers or for software engineers to be certified. The master degree program is an academic program of study to those computer graduates who want to study, on a deeper level, and contribute, through research and application, to the branch of software engineering.

A couple of challenges were faced for those programs. At NDSU, a graduate certificate has to cover parts of the master degree to give those who earn the graduate certificate the chance to continue, if they want, to earn a master degree. So both programs have to be designed to in coordination to satisfy this requirement. However, it is expected that some of those seeking to earn a certificate are already in the software business and they would not leave their jobs to study for the certificate. Thus, courses involved in both the master and the certificate are to be provided both on regular classes and online, in parallel.

This paper describes the design of those two programs. The Master degree courses were designed first, and then each course is divided into 5-7 modules. The modules are designed to be equivalent to the regular master courses; however, modules are designed in a way that allows students to have various choices, 3-4 modules, within each course to fit their line of work. The courses with their contents, assessment, assignments, lab work, research components, etc. have to be designed in parallel to satisfy both types of students and at the same time cover the standards information and skills defined by professional societies such as IEEE and ACM.

The paper states the problem and its dimensions, then overviews standards, based on IEEE and ACM, for such a certificate and degree. The paper describes the designed solution for the two programs, the regular and the online modules. A complete description of one course with its modular and regular description is given. Since this program is still under design, we discuss some of the expected results and thee future plan for giving the course.

This parallel design of a course turned to be very challenging and rich. On one hand, we want to balance a regular class and online modules. On the other hand, we want to balance between a completely structured, instructor directed, and module structured, student directed offerings. The design itself is intriguing and we are looking forward in the application for evaluating this parallelism. We found that the design of such a program is useful, particularly, the principle of class-online parallelism. We believe it is a good experience to be shared as a software engineering program for professionals and scholars, as well as a course design experience.

1. INTRODUCTION

For some, software engineering is just a glorified name for programming. If you are a programmer, you might put “software engineer” on your business card- never “programmer,” though. We could distinguish software engineering from programming by its industrial nature. Adopting a text book definition of software engineering: “the development of possibly large systems intended for use in production environment, over a possibly long period, worked on a possibly many people, and possibly undergoing many changes.” Development includes, in addition to design and construction, management, maintenance, validation, documentation, and so forth.

More people are learning how to do some programming, aided by the growing sophistication of development tools for the mass market. It is likely that many of the estimated six million people who are Visual Basic developers have not received formal computer science education. Despite the dot-com debacle and economic slowdown, salaries remain excellent Project leaders worry about headhunters hiring away some of their best developers, or ponder the latest offers they receive themselves. However, a serious economic downfall can make employers choosier. This competition will force the real professionals to stand out. Managers in the industry reveal that they are not just looking for employees- they are looking for excellent developers. This is the really scarce resource. The software engineering literature confirms [..] that ratios of 20 are not uncommon between the quality of the work of the best and worst developers in a project; managers and those who make hiring decisions soon learn to recognize where a candidate fits into this spectrum. The aim of the top educational program is to train people who will belong to the top tier.

On the product side, people grumble a lot about software, but they learn to get on with it. The demand on software is so high that people would compromise quality if they one way or another can get the job done in an acceptable time. In addition, the wide spread of personal computer owners who are not highly professional would sacrifice quality for the price. They can afford cheaper software and they in exchange would accept lower quality software. That situation however, would not last forever.

Since software industry is human intensive and involve intellectuals, the quality of the product depends on the quality of the developers and the development process. Those current problems with software have been serious enough to lead some government authorities, such as the state of Texas, to require licensing. It also has been announced that IEEE is providing an exam for licensing developers nationwide. Those initiatives reflect a general trend towards distinguishing the true software professional from the occasional programmer.

David Parnas, a pioneer in the field, emphasizes the “engineering” part. Thus, referring to the definition of engineering provided by the Accreditaation Board for Engineering and Technology, 1996, “*The profession in which a kanowledge of the mathematical and natural sciences gained by study, experience, and practice is applied* *with judgment to develop ways to utilize, economically, the materials and forces of nature for the benefit of mankind."* This definition applies to all engineering disciplines, civil, electric, mechanical, etc. For software, a text book definition of software might be: *Software is (1) instructions (computer programs) that when executed provide desired function and performance, (2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs."* Thus we can develop a definition of software engineering from combining those two.

2. REQUIREMENTS

Based on the IEEE Guide to the software Engineering Body of Knowledge (Version 0.95) which is built for the purpose of setting consistent view of software engineering worldwide and clarify and characterize the contents of software engineering discipline and to provide a foundation for topical access to the body of knowledge that should be covered for a software engineer to practice the profession. The software engineering body of knowledge falls into ten areas, each is called a knowledge area. The ten knowledge areas are:

1) Software Requirements
2) Software Design
3) Software Construction
4) Software Testing
5) Software Maintenance
6) Software Configuration Management
7) Software Engineering Management
8) Software Engineering Process
9) Software Engineering Tools and Methods
10) Software Quality

In the design of our curriculum, we viewed the topics in these knowledge areas not the sequential manner, i.e., it would be straightforward to have a course for each area. But, partly, based on our interpretation of the areas, and facilitated by our approach of modular design, we viewed the areas of software quality, software methods and tools, software configuration management as a lateral issues. Simply let us assume that we viewed topics in a vertical and horizontal dimension. The horizontal direction goes through the various software development activities. The vertical direction goes into levels of concepts, where it relates the various activities to the kernel concepts of software engineering.

Those concepts are not concentrated into one of the development activities but are contributed across the whole process of development. Based on that we viewed the software quality as a vertical concept. It is achieved throughout the whole process. It starts from requirement and proper development model to design and then during construction and testing. It also kept and maintained during the software maintenance. In a similar manner, we can view tools and methods as a vertical.

3. DESIGN METHODOLOGY:

The main theme of approaching this design problem was to apply a software design methodology to designing courses. In fact, that is what we see as a contribution for this paper. In software design there is architecture design, structural design, and detailed design. Generally, the design phase in software development is the process where decisions are made. Decisions about the general approach and major components is the architectural design. Then the breakdown of each components into smaller components in a hierarchical structure is the structural design. Finally the detailed design is where the details are added to each small components. The detailed design is the one that is passed to the implementation. In this phase each component is implemented separately, yet if the design is well established, the components would not have contradictions. Simply that is the main objective of the design.

It is note worthy to say that the same process of making decisions and implement them already exists both in course design and in software design. But they are mixed together in one big process, i.e., we make the decisions while we are making the implementation, we may run into problems. That is the main reason for constructing the course syllabus and plan the course before we start the teaching. This is part of the design. However, even when we do that, the detailed design is still left to the implementation time. Also, in this particular case here, it is our chance that we are designing the whole software engineering program, not just one course. This is a difference that allows the global design and making sure that the architecture of the whole program is handled separately. Also allows the interaction between courses to be seen in advance.

Design Constraints

In trying to have a general structure and an approach to whole program. The guidelines that we have are the different knowledge areas of the IEEE recommendations, general guidelines to designing courses and our experience both in course and software design. The whole process of course has to be defined within the constraints of NDSU which is the university that will offer the program.

The constraints were many. First, software engineering is an applied science. It is associated with computer science and the university, but it is mainly directed to those who are currently working in the software development field or those who are aiming to work in the software development field. It may also apply to some who are switching fields of work to software development. Some of those are students who will graduate and will join the job market, but the majority is already working. This means that the courses have to be designed in a way that makes it convenient for those who are at work to take the courses. This used to be solved by having evening classes. However, in the software development field the work is very demanding and sometimes people would work in irregular schedules based on the work plan.

Second, we have to build the courses with a reasonable number of electives. Again since software engineering is an applied field with various responsibilities and tasks, each student will be working in a particular range of activities. Thus he/she would like to elect what would fit the field of work, or the field that he/she wishes to be in. Adds to this factor that different companies adopt different methodologies of development of software and different measurements and characteristics of the software process and product. Thus electives have to also provide for students to apply the same concept in more than one way.

Third, objectives of various students are different. Some would like to tackle a narrow area of knowledge or application and apply it in their work. Others, particularly, those who work in research and training, or may be higher in the hierarchy of the management would like a broader view of the software engineering activities.

Fourth, though the interest in software engineering has been established for sometime, the formal approach and the call for designing university programs is emerging lately. This makes the job of software engineering may take the academic nature. This adds the interest in research to develop tools and techniques and to be used in the development companies to increase the quality and/or productivity of those companies or to improve their production process.

Those wide range of targets put wide range of requirements for the program. We need to design a set of courses that cover number of areas of knowledge for various people for various levels. That is each area has to be covered on various depths to satisfy those range of applications. The solution has to fit in the university program requirements.

Those challenges were the motivation to find a creative approach to find a solution. We use an approach that seems similar to the approach we use in designing software. We first think of architecture, then a structure, then we define the details.

4. PROGRAM DESIGN

We had to approach the problem as an integrated program. Look at all levels and all variety of courses as within one integrated plan. This produced the following rules to construct the program.

1) The core knowledge will be structured in six main course areas

The ten areas of knowledge that have been identified are divided in to six areas each of them is realized in one course. The six courses cover all ten areas in two dimensions. The six courses are:

a) Software Development Process and Lifecycles

This course covers a breadth view of the software development process and all its activities. It also covers people responsibilities and roles within teams in various team structures. For each of the development processes, analysis, design, implementation, construction, testing, and maintenance, it covers what are the objectives of each, the tasks achieved in each, the input and deliverables, the tools and techniques used, as well as its effect on the quality of the product and the maintenance activities.

b) Software project Planning and Estimation

This course is a second course after the process and covers the estimation of effort expected in each activity. It also covers various approaches in handling the development of software as a project, i.e., the development within limited time and budget. A major theme of this course is trade offs and how to handle problems of development. Scheduling techniques such as PERT and CPM together with software tools to support such techniques are covered. Also, it covers the effect of accurate scheduling on the productivity of the team, the project time, the project budget, and the quality of the product.

c) Software Requirements Definition and Analysis

This course reviews some of the systems analysis techniques of requirements gathering and elicitation. Then it covers the standards for requirements specifications and documentation. It covers the characteristics of good requirements, such as non-ambiguity, consistency, completeness, etc, and the review techniques that assures the non absence of those characteristics. Concepts such as tracability of requirements through lifecycle process of design and construction are covered. Also it covers techniques for building of test cases, integration strategies and test strategies.

d) Software Design

This course covers design concepts, design approaches, design techniques and assessment through reviews and walkthroughs. Various design levels of architecture, structural design, and detailed design are covered. Various paradigms of structured, top-down, and object design are compared and contrasted. CASE tools that are used in design checking and documentation are discussed. Configuration definition and documentation as part of the structure is emphasized as part of the process of configuration management. A major theme in this is the effect of a good design on the quality of the product. Also the association of design with the designing of testing techniques and the development of test cases.

e) Software Construction

This course covers languages of implementation, integration methodologies, and collaborative approach to building software, generic programming, and program libraries.

It also covers development environment and testing strategies on the unit levels. Issues such as performance and the effect of construction on performance are discussed. The concept of versioning and the associated configuration definition and management is also discussed.

f) Software Testing and Maintenance

This course overviews concepts of testing and various types and levels of testing. Quality issues and the effect of various tasks and activities on quality. The assessment of a software product and how to features can be shown. Also issues such as the difference between verification, validation and testing and proofs of correctness. The importance of developing a strategy as early in the life cycle as possible. Configuration management through the maintenance process and the issues of versioning is covered.

The design of the six courses covers all the ten areas of that are recommended by IEEE. The general philosophy is to create a breadth course first acts as an introduction and as a basis for the realization of the global view of the software development. Then One course on measurement of effort, estimation, and planning to add a managerial skills to the whole process. Then one course for each of the development processes of requirements, design, construction, testing and maintenance. The other areas of configuration management, quality, tools and methods are covered as part of each of the processes. That is what we referred to as a two dimensional design of the courses. On one Dimension we have the development process and on the second dimension we have depth of quality, tools and techniques. The process of configuration management is in our opinion across the whole life cycle and that is the reason for covering it across requirements, design, construction, testing and maintenance.

2) The program provides three degrees.

The program is designed to have three degrees on three various levels and for three different purposes. However, they are related to each other. The three degrees are a graduate certificate, a Master of Science, and a PhD. in Software Engineering.

The *Graduate Certificate* is a set of graduate courses from the six that are defied above, assuming an entry level of a BS or BA in Computer Science. The graduate certificate is targeted to workers in the software development industry or graduates who want to join the industry. Each certificate is a total of 10 credit hours

i) three courses from the above six, the software processes and other two;
ii) a seminar, for one credit, to discuss recent issues in the area; and
iii) an applied project of the choice of the student and the advisor.

For example, one who works in requirements will take a course a, B, and c from the above list. One who works in management takes a, b, and f. One who works in design takes a, d, and f. So we can build different set of certificates from those six courses. In fact a company can define its own requirements as far as it is the course “a” followed by two of the remaining five. The seminar is a discussion of papers from journals to update the recent issues in the area. The project is recommended to be in the area where the person works. This would guarantee the relation of the academic issues to the applied work.

The Mater of Science covers a breadth of the knowledge in software engineering. It is mainly directed to higher level of workers in the software development companies who want to have a global view of all the activities of the software engineering. Also, it help those who work on training and education or who supervise software engineering processes in a company. The requirements for this degree is 33 credit hours:

i) The core courses of computer science (9 credits);
ii) All the six courses of software engineering (18 credit);
iii) Three seminars (3 credits);
iv) A paper written on an applied topic in software engineering.

The master can be seen to cover comprehensive knowledge of software engineering with good depth. The seminars cover the recent issues in various and advanced areas f software engineering. The paper is a trial to apply or join more than one area of knowledge.

The PhD. is a higher-level depth and is directed mainly to those who seek jobs in teaching positions in schools of computer science. We expect that with the trend in formalizing degrees in software engineering, the need for professorial positions will increase. Candidates must show a comprehensive understanding of the knowledge areas in addition to the ability to conduct research and contribute to the body of knowledge of the software engineering by developing new techniques, new tools or new concepts. The requirements of the PhD. is the same requirements for the Master of Science in addition to:

i) Three advanced courses in software engineering,
    a. Formal Method in Software Engineering
    b. Development of distributed Systems
    c. Software Complexity Metrics
ii) Dissertation research.

In conclusion, we have defined the three levels of study in a way that covers various depth and breadth of the topics. However, it was taken into consideration that each degree contains completely the lower one. This facilitates two properties. First, if a person want to pursue a higher level of competence and knowledge, they would not lose any credits. They can proceed to continue accumulation of credits. Second, particularly, for those who are seeking jobs, if they want to discontinue a higher degree, they can stop at the lower level without losing the while credits. Also, the time span can be longer where graduates want to come back to school and pursue a higher degree.

3) Each course is further divided into a set of modules

Each course, as a structure, is built out of a number of modules. The current status is that each course is defined in 9-15 module. Each module is an independently studied unit. It has a given purpose within the area of knowledge. However, the total number of hours on the whole set of modules is higher than the required class hours in a course. This gives the choice of the instructor to design his/her class material as a number of modules (usually 5-7) out of those total number of modules. (This concept is similar to big text books with various implementation.)

However, the idea here if to give the student to elect the modules that suits his/her area of work. This creates a new level of elective. In addition to the elective courses, we have elective modules. A student who works in a company and they use certain standard or technique, they can chose to study what fits his area better. Meanwhile the instructor in the classroom have a fixed plan to cover as a pre-designed plan of work. (refer to the coming point of deign, online courses). Each module is deigned with an exam, example, case studies, etc. to make it an independent learning object. Each module also has a weight in number of points related to the equivalent class sessions they ought to be taught in the classroom. Thus a student can elect to gather various modules to sum up the course points.

This feature helps in many situations. First, if we are covering a certain tool for a particular technique. However, there are more than one tool available. A student may pick the module that covers the particular tool that is used in his company. Similarly, if there is more than one technique, say for design, an the student wants one that fits his/her area of application. Thus the course is design with various techniques with the instructor giving the choice to each student to study the one that fits his needs. Second, Since students are assumed to be working, it may be a situation that a student want a level of depth in one aspect more than other in class. In this case, he/she can go deeper in this area as a choice, given in the design, and would take less emphasis on another area.

Thus the course is design as several modules. Each module covers one area on various levels. Then the instructor gives the choice for students where he/she thinks that the student can have to elect either a level of depth or an technique. Meanwhile still the instructor can go with a fixed plan, if he/she chooses to.

4) On-line courses for the certificate level in parallel with the regular class

Since we assume the students can be working in software development companies as they are studying, we decided that the courses, particularly that are offered for the graduate certificate, will be available online. This would allow students to study parts of there studies while they are at work on at a time that is convenient with their work schedule.

We also, assume that he instructor decides as he/she offers the course on what modules that can be taken online and what other modules that has to be attended in class if he/she chooses to do so. This feature together with the previous, modules, gives a big flexibility to the student first to elect the modules that he/she chooses to take and the time and environment that they can study in.

5) **Each modular area has various levels and techniques to sum up the knowledge within this area.**

Each module is furtherly divided to smaller units, each of them is designed as an independent piece of knowledge. Each one piece can be studied independently, like a lesson or a chapter. However, it is more of a segmentation technique. The length of each unit is not necessary the same in all units. It depends on the topic covered. Thus the module itself is a structure, it is not one unit of text or illustration.

Each unit is independent in the sense that it contains examples, self test exams, and a final exam that mark the completion of the unit. As a unit is completed and the exam is passed, the unit is added to the credit of the student taking it. As the student competes the required number of credits, he/she completed the course.

This feature adds a level of flexibility of the studying style of each student, thus adds a level of convenience to the course taking. However, it is mainly added to make it easy for the instructor to change and add to the material without having to change the whole course design. So if a new technique or a new tool is defined, the instructor can add a new unit to the course and add it to the big mesh of information. We expect that this is a needed feature since software engineering is rather new area and changes are expected both in theory and practice.

It is noteworthy that there is a general structure of the whole program as it is built from modules. This structure reflects the relationships of the modules to each other and the levels of details within each module. Within this structure a module, or sometimes a unit, can be a sub-module from more than one module. So it is referenced from one piece of knowledge or it explains an item that is used in more than one area. This shows one other advantage of the modular structure. An instructor does not have to develop the same module twice even if they are taught in two different courses.

6) **Each module in a level has a given set of details.**

Each module contains number of units as explained above. Those units cover the different aspects of the knowledge area but also cover material on different levels of details. The first unit in a module is expected to overview the whole module. Then this unit is connected to more details. Each unit can be connected to even more details within the module. This gives the flexibility to the student to choose the level of details in one area that he/she should cover. Again since each unit is counted separately, a student can cover what fits in his/her interest.

6. DETAILED DESIGN

Detailed design is concerned with the specification of each unit and the relationships among various units in one integrated structure. We developed a form that is attached to each module. This form defines:

i) Module name
ii) The objective
iii) The equivalent Class hours
iv) The Super-Module (if any)
v) The names of sub-modules (if any)
vi) The level of the module
vii) The type of module (knowledge, application, tool, etc)
viii) The concurrent concepts used in the module
ix) Key words associated with the module

This will help the management of the structure of the modules in the future in updates and adding new modules or changing one module. The concept here is analogous to configuration management in software. We predict that the management of the modules will exert an effort. That is why we added those meta-level description for each module.

As for the structure of each module, it is built from the main content material. This is mainly text with illustrations or diagrams. Each module also has a section for examples, one for self-test problems, and one for exam.

Example Course Modules: Software Development Process & Lifecycle Models

Module 1: Business Systems Analysis
Module 2: Business Systems Design Strategies
Module 3: System Modeling
Module 4: System Design Components
Module 5: Software Quality & Quality Assurance
Module 6: Software Development Phases
- Module 6.1: Phased Lifecycle Concepts
- Module 6.2: Software Subsystem Requirements
- Module 6.3: Software Design
- Module 6.4: Software Construction
- Module 6.5: Software Testing
- Module 6.6: Software Administration
- Module 6.7: Software Maintenance

Module 7: Software Development Life Cycle Models
Module 8: Software Life cycle Tools
Module 9: Types of Software and Choosing the proper Model

7. CONCLUSION

The design that is discussed above is the one that is currently proposed at NDSU. This design is a result of applying the concepts of software design to the course design. The concept of modules, and units, is defined in software. A module is an independent unit of code that named, separately addressed and potentially usable in more than one application. The concepts of software engineering are organized and broken down in a structure that reflects in a way the software design. We have seen that the application of software design concepts added to the design of the courses and provided solution to some of the problems.

However, we have to add some of the concerns. As the course design may inherit some of the characteristics of software design, it may inherit some of its problems. First, The design of the whole program is seen as an integrated program. This as much as it is well design and gives flexibility to instructors and students, it may be seen as a limitation on the instructor in teaching. We do not see that personally. An instructor can add his/her own module or explanation of a concept and it would be attached with reference to the unit or module. Yet , it has to be part of the structure. Second, The number of modules and units that are provided are more than what is offered in one course. This is decided to allow for flexibility in teaching. This will add a load on the instructor to provide details that may not be covered in class.

The solution for this is to consider the consideration of the course design as a task different from the instruction. This will add tasks to universities and departments. However, may be the publishing houses would provide help in this are. It would be seen as providing a book that is in a digital form.

One of the future works that is considered is to design a tool that works on such a structure for construction and management. Such a tool would make the structure and management of the material easier for the instructor and the designer. However, it assumes that the whole program is seen as one integrated unit which means that it will deal with a large entity.

8. REFERENCES

[1] Meyer, B. “Software Engineering in the Academy,” IEEE Computer, May 2001, pp 28-35.

[2] Tsichritzis, D. “Reengineering the University,” CACM, Vol. 42, No. 6, June 1999.

[3] Parnas, D. “Software Engineering Programs are not Computer Science Programs,” IEEE Software November/December, 1999.

[3] IEEE Guide to the Software Engineering Body of Knowledge, Trial (Version 0.95) May 2001.

[4] Hamlet, D. and Maybee, J. “The Engineering of Software,” Addison Wesley, 2001.

[5] Sewell, M. Sewell, L. “The software Architect’s Profession, An introduction,” Prentice Hall 2002.