Gurdeep S Hura. (2017). Software Development using Object-First Approach: A New Learning Strategy. *Journal of Education and Learning*. Vol. 11 (3) pp. 229-234.

Software Development Using Object-First Approach: A New Learning Strategy

Gurdeep S Hura [*]
University of Maryland Eastern Shore

Abstract

Software Engineering approach deals with the Software Development (SD) that is aligned with design and development of software applications. The Software Development may be implemented in a variety of techniques but its implementation using a procedural paradigm and an imperative language seem to be more effective and efficient for the design and implementation of software applications. The procedural approach for Software Development offers advantages as this it may be used to teach some basic features of programming languages. The object of this paper is to introduce the software development and associated object-first approach for the design of software project application using top-down method. This approach defines functions and modules as basic units for the design and implementation and also for offering hands-on experiences with the basics of programming languages of sequences, selections, iterations structures. These structures will be used to define various modules with programming language constructs for of software development process. The software Development process is one of the very crucial processes of software engineering.

Keywords*: software engineering, software development, programming, object-first approach*

[*] Gurdeep S Hura, Department of mathematics and Computer Science, University of Maryland Eastern Shore, Princess Anne, MD 21853
E-mail: *gshura@umes.edu*

Received April 1, 2017; Revised June 2, 2017; Accepted June 17, 2017

Introduction

Software Engineering deals with Software Development (SD) that is aligned with design and development of software applications. It includes the requirements specifications meeting the needs of clients [1, 2]. It may also involve formal specifications and formal methods if the application is large and complex. The software engineering, course is one of the core courses of any Computer Science or Information Systems or Computer Engineering, discusses various phases of software engineering life cycle. The Development within software engineering process is aligned with the design and implementation of software project and it also deal with understanding of basic features of programming language. Further it also helps in learning the design and testing methodologies of Software Engineering. The teaching team provides the program specifications, so, there is normally not much emphasis on requirements elicitation or systems analysis.

The course on software engineering offers needed skills and necessary knowledge for software development and as such students taking the course acquire the appropriate experience. The offering of the course teaches the required theory that is generally not too difficult to understand and it ensures that students gain the necessary experience in a very simple and easy way. The course can become very interesting and learning provided if this can be designed with relevant outcomes that can make them more proficient and confident. However, being proficient in a skill is a function of time and the time available on a degree program is simply not enough. Thus, it is essential that the course team design the practical elements of a software engineering and other programming-based courses of study with great care and ensure an effective delivery.

In Computer Science or Information systems, the first CS/IS course provides the basics of programming language. This course helps the students to use their knowledge and skills of programming into other programming-based courses at educational institutions. This paper focuses on the teaching of software engineering via software development process for design and implementation of software applications.

In this paper, we will present the traditional method of offering a course on software engineering. We will introduce the concept of procedural paradigm and highlight the limitations of this on the software development. Finally, we will introduce object- oriented approach for the software development. We will also introduce how this approach supports the first programming language. A part of this work has been presented in conference [11].

Software Development: A Brief Introduction

There are different methods of implementing software development and also a number of paradigms for representing functional composition. Some of the needed frameworks include: imperative, object-oriented, functional and logic. All these frameworks are very important to understand the basics of programming language at different levels. It has been observed that procedural or imperative frameworks are being extensively for providing training on object-oriented or object-based approach for software development. This approach offers the following features:

1) This approach is highly compatible with today's computers.
2) The third (and fourth) generation languages are highly suitable for implementing procedural programming.
3) The programming based projects consist of set of projects or objects, but it looks that set of procedures seems to be better and more natural than set of objects.

There are some situations where the software development can be designed using traditional functional and logic programming methods. In the following section, we will present both the frameworks procedural programming and object-oriented [3].

Procedural Programming

Although the procedural approach may be considered as suitable for programming-in-the-small, it is not entirely appropriate. It is a bottom-up approach highly dependent on the chosen language. The emphasis is on learning a language and not on the modelling of realistic computational problems. In most cases, teaching of the language follows a scheme that requires the teaching of the following in, more or less, the order, as presented here:

1) General program structure
2) Declarations and Variables
3) Input/output and Assignments
4) Iterations and Selections
5) Arrays and Records

6) Functions and Procedures
7) Other features of the language

In addition to learning the syntax of a computer language, we need to address the language environment, often called *interactive* development *environment* (IDE) where we discuss the use of the IDE, use of the editor to input and edit source code, and use of the compiler system for compilation, linking, building and executing the programs. Furthermore, we need to get a clear understanding of the system and other diagnostics including syntax and runtime errors, all of which adds another layer of difficulty to the learning process [4].

In summary the *following* features for designing any software applications need to be understood:

1) Detailed design steps for software development
2) Syntax of necessary declarative and other functional statements of a language to convert design into program code
3) Procedures for entering, editing, compiling, linking and executing the program
4) All other procedures such as starting the IDE, getting into the language system and closing the project.

The Design of software development may use any techniques like function-oriented methodology, functional decomposition or stepwise refinement and each of these techniques also provide easy learning of language and design and its environment.

Thus, the steps required for the design of software development are well understood and are being used properly. It is important to note that the right use of these steps may create confidence and provide useful results, failing which the developers may get disappointed.

Not with standing the difficulties mentioned above, a careful observation reveals that the process, that is usually followed, does not support each construction of software as an engineering activity. Although, applications are being designed and developed, the emphasis is not on building software based on accepted engineering principles, but make sure that we understand the need to employ when we are engaged in the development of complex software. Also, we find that the project team are so busy learning the essentials that the approach does not leave much time for learning documentation, quality, professionalism and elements of good practice.

In order to address some of the issue as discussed above, we introduce an objects-first approach for the software development.

An Objects-First Approach

An objects-first approach requires that modules and functions are regarded as basic building blocks, that software applications consist of interacting modules and that new modules are built using existing ones, whenever possible. This forces a structured approach to modular programming where use of modules and functions establishes the principles of code *reuse* and functional independence. Clearly, the emphasis is on modularisation, encapsulation, recursion and reuse. This contrasts sharply from the traditional procedural approach where modularity, functions and recursion do form part of the curriculum but they are taught much later in the module.

Object-First-Based Software Development

This section presents the use of object-first approach to procedural paradigm for the implementation of an engineering approach to software development. The proposed model for the design is a loosely based method and is based on the modelling concept [5]. In the model, the design method regards the construction of software as an engineering activity where modules and functions are the fundamental building blocks. The method helps to produce properly structured and good quality modular software. It is a top-down approach where the important concepts of object technology and principles of engineering are introduced right at the beginning of project development.

Our method requires the establishment of a library of functions on a suitable topic (e.g. graphics) prior to the delivery of the software development (SD) module. When students begin to learn SD, their programs will be written as sequences of given functions where we will consider only the external behaviour of these functions. Building programs in terms of functions will help us to understand modularisation, reuse and encapsulation mechanisms, without knowing the intricacies of the computer language.

It is important that we are able to successfully execute their programs early on in the course. This provides a sense of achievement and increases our confidence. After successfully running a number of simple programs and understanding the basics of the language environment, programs are written as selections and repetitions of the same functions. It is at this point that we learn the general syntax of

selection and iteration statements of the language. Now, the importance of modularity and code reuse can be re-emphasized and mechanisms for reducing software complexity, incremental development, polymorphism and overloading of functions explained which can be practiced in later sessions when we produce our own libraries of functions.

We have gained enough experience and understanding in learning the syntax of input, output, assignment and other basic statements of the language for producing 'real' programs. Now, we have to practice functional independence, quality, code readability, maintainability and other elements of good programming style. Further, we have the necessary practice and knowledge of the language, we can begin to 'problem solve', design and build our own programs from given specifications.

During the development of software design module, our focus should be on problem description and problem solving strategies. We suggest that the design technique be a simple one so that we do not feel that we are learning an additional method - Stepwise Refinement [7] is a perfectly acceptable approach. Although, we will learn by producing our own programs, use of good quality, well structured and properly documented worked examples will greatly advance the learning process.

Learning Strategy

We now outline a learning plan based on the above model. We assume the module will be discussed for the duration of at least one month during which lecture notes, homework, practice on programming and understanding the software development process within the context of software engineering for the design of software project.

During the first week, we expect the team members to create a library of functions on an appropriate topic e.g. graphics with simple example of two simple routines such as LINE and CIRCLE. It will demonstrate how to draw a line of a given length in a given direction starting at a given point and the other to draw a circle of a given radius at a given point. After the basics, we introduce the IDE and get the team members to familiarize with the editor. We will give a simple working program, which they enter and execute to understand the compiling and execution process. Further, we introduce the program structure in the chosen language. It is important that examples chosen are well constructed and properly commented to illustrate good practice.

During the second week, we introduce the concept of functions and modules as components and explain the purpose and use of some of the library routines e.g. LINE and CIRCLE. Next, we can explain the significance and meaning of required parameters and use of argument lists. Now the team members will get experience in writing simple programs calling functions in a sequence, drawing a chair (consisting on a number of straight lines) or a table (consisting of a circle and a number of straight lines) and drawing a row of chairs or a number of tables using only the functions they are already familiar with e.g. LINE and CIRCLE. Now we can introduce the basic design process e.g. stepwise refinement that includes selection and iteration statements and get experience with these in the program by drawing a row of chairs or a table surrounded by a number of chairs.

During the third week, we explain the significance and importance of good programming style, modularity, code reuse, functional independence and other engineering principles, the advantages of incremental development and benefits of producing proper designs and test plans, benefits of using appropriate standards, developing quality software, keeping accurate records and producing proper documentation. Next, we encourage them to use modules they used in earlier programs (e.g. LINE and CIRCLE) represented objects and classes and to create new objects (chairs and tables) in their programs. Now, we introduce the basic concepts of object technology, the essential terminology and teach the syntax and use of input, output and assignment statements of the language. Finally, we will expect the team members to write their own functions and objects and produce modules say CHAIR to draw a chair (invoking perhaps just a single function LINE) and TABLE to draw a table (invoking perhaps just one CIRCLE and a number of LINE functions).

During the fourth week, we expect the team to produce more complicated programs and practice what they have learnt. Here, we may offer windows-oriented programming environment where they will be exposed to programming language such as VBasic, C# and Java [8-10]. Further, advanced features of the language and other engineering concepts can be taught. Make them aware of the difference between programming-in-the-small and programming-in-the-large. Re-emphasise the advantages of incremental development and benefits of producing proper designs and test plans.

Learning Plan

The learning strategy and plan for training for the development of Software Development of Software engineering presented in this paper suggests the sequence of following concepts in the sequence:

1) Program structure and program layout
2) Use of libraries and sequences of statements
3) Functions, procedures and parameters
4) Selection and repetition statements
5) Input, output and assignment statements
6) Data structures: arrays and records
7) Advanced features of the language.

Programming Languages

The primary objective of Software Development module should be to train the team members with the principles of programming. In this respect the choice of a language becomes irrelevant [6]. However, the teaching team need a language to illustrate the principles and provide practice of SD. Choice of the language, then, depends on the programming paradigm employed. Since, procedural programming is the most favoured approach; first languages tend to be mainly procedural. However, object-oriented, object-based and visual languages (e.g. C++ and VBasic) can also be used for procedural programming for console applications. In some cases, we can also use declarative languages that are based on logic and functional programming paradigms [3].

It is often suggested that a first language should be well structured, available in the sense of staff expertise and easy to teach, learn and use [6]. Whereas, this may be acceptable for programming-in-the-small, when training principles of engineering and elements of good practice with a view to producing complex software, the criteria is not sufficient. Since choice of a language depends, also, on the programming and design methods used, the above criteria need to be extended. We suggest that a first language should possess at least the following characteristics:

1) Small, simple but powerful
2) Strongly typed and block structured
3) Procedural with capabilities of offering extensions to implement object technology
4) Features allowing implementation of engineering principles and concepts
5) Adoption by industries

Simplicity and smallness imply ease of use as well as ease of learning and debugging. Power of a language is its ability to deal with complex problems as well as simple ones. Strong typing reduces debugging problems and block structuring helps to produce structured and modular software. Features to implement object technology and engineering principles are essential when implementing object-oriented approach or a model similar to the one suggested in this paper. Industrial relevance is important for the reasons of collaborating with Software industries.

Fischer [3] suggests the following criteria for the programming language choice:

1) Powerful enough to demonstrate the programming concepts
2) Easy to learn
3) Not error-prone i.e. get running fast
4) Easy to use development tools
5) Well supported by ways of availability of library functions.

Currently available languages such as VBasic, C# and Java [8-10] are all highly suitable first languages for the development of Software Development in Software Engineering.

Conclusions

Procedural paradigm is the traditional and most favoured approach for training Software Development. It is a bottom-up and syntax driven approach highly dependent on an imperative language. With this, we can learn not only the syntax of a language but its environment as well. If a formal design method is also included at the same time as the language then we can get so overwhelmed by the amount of learning that some may loose their confidence and get disappointed with the learning experience. Also, the traditional approach teaches programming in the sense of producing code and does not teach SD as an engineering activity.

To resolve the inherent issues in the traditional approach to teaching SD, this paper suggested an objects-first approach to procedural paradigm. This is a top-down approach, which regards functions and modules as the fundamental building elements for the construction of software. The emphasis is on modularity, code reuse, practice of engineering principles as well as quality, standards and professionalism right from the start.

A learning strategy and plan scheme was also presented which can be used as a basis to construct a well-defined training and understanding of Software Development.

References

Sommerville I, "Software Engineering", Addison Wesley, 8th Ed, 2006

Bell D, "Software Engineering for students", 4th Ed., Addison Wesley, 2005

Fischer P, "Teaching programming to beginners", IMM, DTU, Retrieved 20 June 15, from www2.imm.dtu.dk/~tb/fischer.pdf

Dehnadi S and Bornat R, 2006, "The camel has two humps", Retrieved 20 June 15, from www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

Bornat R. "Programming from first principles", Prentice Hall.

Busschots B, "Teaching programming – why the choice of first language is irrelevant", Bloggs dated 8 Jan 2015, Retrieved 20 June 08, from http://www.bartbusschots.ie/blog/?p=634

Wirth N, "Program development by stepwise refinement", Comm. ACM, Vol 14, No 4, pp 221-227, 1971

Balena F, "Programming microsoft Visual Basic", 2nd Ed, Microsoft Press, 2006

Murach J, "C# 2008", by dissection, M Murach & Associates, March 2008

Deitel and Deitel, "Java: How to program", 7th ed, Prentice Hall, 2007.

Gurdeep Hura," An object-first approach for the Software Development: a new learning paradigm, MTMI conf. Virginia Beach, VA, Sept 22-23, 2014