A Software Engineering Schema for Data Intensive Applications

By: Shan Suthaharan

S. Suthaharan. 2018. A Software Engineering Schema for Data Intensive Applications," in Proc. of The ACMSE 2018 Conference, Richmond, KY. March 29-31, 2018. (p. 23) 8-pages. ACM.

Abstract:

The features developed by a software engineer (system specification) for a software system may significantly differ from the features required by a user (user requirements) for their envisioned system. These discrepancies are generally resulted from the complexity of the system, the vagueness of the user requirements, or the lack of knowledge and experience of the software engineer. The principles of software engineering and the recommendations of the ACM's Software Engineering Education Knowledge (SEEK) document can provide solutions to minimize these discrepancies; in turn, improve the quality of a software system and increase user satisfaction. In this paper, a software development framework, called SET*h*, is presented. The SET*h* framework consists of a set of visual models that support software engineering education and practices in a systematic manner. It also enables backward tracking/tracing and forward tracking/tracing capabilities - two important concepts that can facilitate the greenfield and evolutionary type software engineering projects. The SET*h* framework connects every step of the development of a software system tightly; hence, the learners and the experienced software engineers can study, understand, and build efficient software systems for emerging data science applications.

Keywords: Software engineering | object-orientation | class diagrams | design patterns | data science

Article:

1 INTRODUCTION

Software Engineering is one of the important topics in computer science [11]; on the other hand, it is a difficult subject to teach, learn, and apply unless otherwise it is structured systematically. The basics of software engineering can be simply expressed as a game between a user and a system as illustrated by the proposed TBoSE diagram in Figure 1; however, the ultimate goal of the game is to achieve a win-win result between the user and the system. The TBoSE diagram describes the following: a “user requirements” document is created for a system, and a “system specifications” document is created for the user by a software engineer, and they are then continuously revised the software engineer to meet or exceed user’s expectations. This is the main loop of a software engineering development process and the framework.

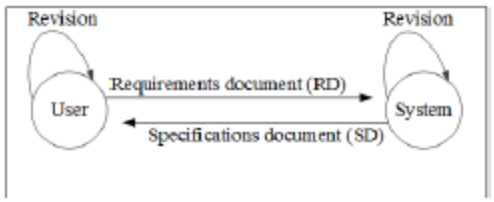

Figure 1. TBoSE: It illustrates the two main components - tuple (User, RD) and tuple (System, SD) - of a software engineering framework.

Two of the current focuses of computer science discipline are the education and practices of software engineering with global [2, 4], collaborative [7, 18], and big data [9, 13] perspectives. These are some of the emerging data-intensive applications that require new software engineering schema. It must include the design, development, and delivery of software engineering strategies and documents by defining suitable domain and models to address these emerging issues. Considering a large-scale and collaborative environment, a domain may be defined for software engineering education and practices using the fundamental definitions presented in [17]. Therefore, we can define the domain of software engineering by using the TDoSE diagram presented in Figure 2, where multiple TBoSE models are used for covering a large-scale collaborative environment (or multiple systems).

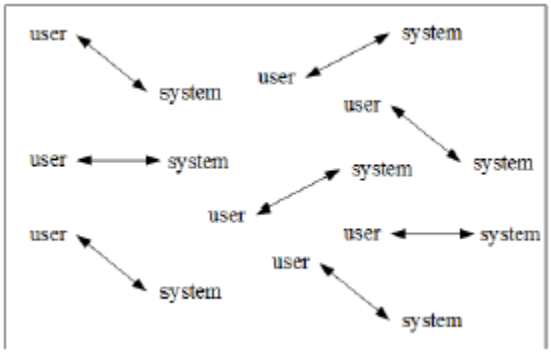

Figure 2. TDoSE: It illustrates the domain of software engineering theory and practices with tuples (user, system).

Software Engineering 2004 (SE 2004) is a document that provides useful recommendations that can help higher education institutions to design, develop, and deliver strong software engineering curriculum within their computer science undergraduate programs [12]. Later, this document was revised in 2014 to address the advances in software engineering discipline [1]. A common contributor to both SE 2004 and SE 2014 is the Software Engineering Education Knowledge (SEEK) that presents the contents for the software engineering curricula to educate students enrolled in computer science programs, and adopt them in the software engineering practices. The SEEK recommendations include two main areas: knowledge and pedagogical guidelines. SEEK also presents 10 knowledge areas - as listed by Lethbridge et al. in [12]. We can now group these areas into two broad categories so that they can be addressed distinctly in this paper as follows:

· Theoretical knowledge: The knowledge that a software engineer must possess is listed in [1] and [5]. It focuses the fundamentals of computing, mathematics, and engineering, software modeling, design, and analysis, software verification and validation, and software processes.

· Practical Skills: The importance of practical skills is significantly emphasized by Landwehr et al. in a recent article [10]. It mainly focuses on the deployment of theoretical structures to applications; hence, it includes professional practices, software evolution, software quality, and software management.

This paper mainly focuses on the knowledge areas associated with the theoretical knowledge (or structure), because they contribute to the design and construction of a software product from a project definition. This structure, when dealt with a real application, can induce difficulties in handling different types of projects. In general, a software development project can be identified as an evolutionary type project or a greenfield type project [11] - an evolutionary project means that the software is already built and the current software version needs to be modified partially with software engineering methodology, and a greenfield project means a new software should be developed entirely.

A poor design associated with these two types of projects bring different problems and challenges to software engineering education and practices. For example, the evolutionary project has to deal with the understanding of every stage of software engineering methodology used by another software engineer. The challenges include the knowledge extraction and backward tracking and tracing to determine the processes applied. In contrary, in a greenfield project, the challenges include forward tracking and tracing as well as the knowledge extraction. The tracing problem is defined previously with respect to object interaction. For example, in 2004, authors of [6] analyzed the traces resulted from the object interactions and surveyed eight trace exploration tools and techniques. Hence, the problems and challenges can be grouped into the following three categories so that a software engineer can build the system as a tracking-enabled and tracing-enabled software system:

· Knowledge Extraction: Every stage of a software engineering process - from user requirements to software specifications - generates knowledge that is useful to develop a software system and meet user expectations. This knowledge must be extracted to design and construct a reliable, reusable, and maintainable software system.

· Backward Tracking and Tracing: One of the problems in software engineering design and construction - due to complex nature of the design procedures - is the backward tracking and tracing of the tasks performed. While it supports heavily the evolutionary type projects, it is also useful to greenfield type projects. Hence, a tight connection must be established between every step of the software development.

· Forward Tracking and Tracing: In the design and construction of a new project, based on software engineering principles, the selection of suitable models and design patterns is a problem and challenging. Hence, forward tracking and tracing capabilities must be built into the development phases systematically. While it supports heavily the greenfield type projects, it is

also useful to evolutionary type projects. Once again, a tight connection must be established between every step of the software development.

The SEEK recommendations also provide 19 pedagogical guidelines [1]; however, it does not specify any pedagogical tools or techniques that should be used to educate students for the understanding of software engineering theory and its applications. While it benefits individualization and customization of software engineering curricula, they can create significant variations in the standard of computer science programs between institutions as well as between professors who develop and deliver the courses. Therefore, it is essential and useful to introduce tools and techniques that facilitate consistency between different programs, and promote effectiveness among software engineering graduates and professionals, which is the goal of this paper. One of the important contributors to software engineering education and practices is the preparation of documents - mainly the user requirements and system specifications documents. Hence, the writing and presentation skills are also required, and their importance has been stated in a recent article by Beslmeisl et al. [3]. Therefore, one of the aims of the proposed framework is to structure the model to facilitate writing (and presentation) requirements in software engineering.

1.1 Contributions and Solutions

The main contribution of this paper is a systematic organization of the standard software engineering components [1, 11, 12] and the establishment of tighter connections between them to help students learn software engineering efficiently, educators of software engineering deliver courses effectively, and software engineers identify strengths and weaknesses of a system and present the product in an understandable form to a user. It can enhance software engineering practices significantly. In essence, it can help remove the vagueness of user requirements, simplify the complexity of the systems, and supply sufficient knowledge to software engineers.

Hence, the solution that this paper provides is a software engineering schema - Software Engineering Theoretical (SETh) Framework - as an important tool for an academic setting, as well as for industries with emerging data science applications. The proposed schema is new and can help to structure and build a strong computer science curriculum, and develop software products in a systematic and organized fashion, especially supporting the emerging global, collaborative, and big data software engineering applications and the associated software development problems and challenges.

In this paper, SETh is viewed as a framework that allows physical (or structural), functional, and logical operations to build a software product that is reliable, reusable, and maintainable. It can also be viewed as an object and an abstract data with structural, functional, and logical characteristics when building an object-oriented software engineering framework. The proposed SETh framework is composed of several visual models that can help connect the software engineering components from a project definition to a software product so that the backward tracking/tracing and forward tracking/tracing can be achieved in an efficient and effective manner through a systematic approach.

A SETh-based project development is characterized with the standard definition of computer science - input, processing, and output. The input is a project definition (or a problem statement), the processing is the design, implementation, and testing of the theory of software engineering, and the output is the well-documented software product [3]. In addition - as per the standard definition of software engineering stated in SEEK recommendations - a software engineer is responsible for delivering two systematically organized documents - requirements document and specifications document - and a complete system software that neatly addresses the project definition or the problem statement of the project.

The contributions of SETh framework presented in this paper includes the systematic organization of the contents of a general software engineering curriculum and practices. It provides a number of visual models to explain software engineering theory and practices from user to system (or project definition to software system): The Basics of Software Engineering (TBoSE), The Concept of Software Engineering (TCoSE), The Domain of Software Engineering (TDoSE), The Processes of Software Engineering (TPoSE), The Relations of Software Engineering (TRoSE), and The Methodology of Software Engineering (TMoSE). These models can facilitate the understanding of the software engineering phases applied and help the development and revision of the implementation. The two segments, user-to-GUI and GUI-to-system, of TCoSE model in Figure 3 can help us split the tracing concept and develop the two concepts - backward tracking/tracing and forward tracking/tracing - associated with the proposed SETh framework.

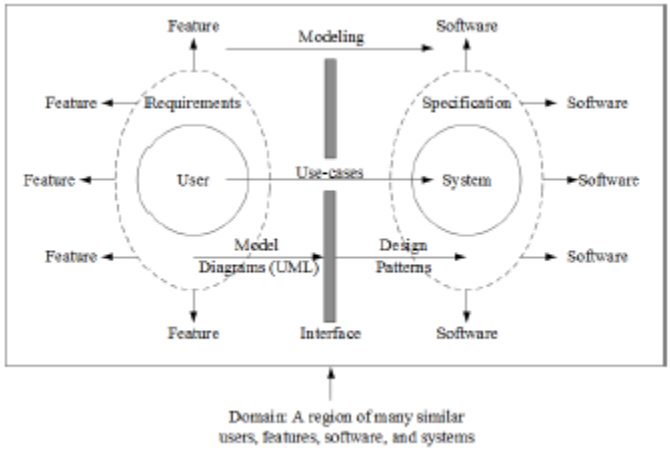

Figure 3. TCoSE: The concepts of software engineering.

The rest of the paper is organized as follows: In section 2, the proposed SETh framework is explained in detail. It mainly focuses on the six visual models that enable the understanding of knowledge extraction, the backward tracking and tracing, and the forward tracking and tracing. In section 3, a simple example - the development of a subsystem for traffic safety using longitude and latitude information- is provided to illustrate how the SETh framework can be applied. In section 4, a detailed conclusion is presented.

2 SET*h* FRAMEWORK

A software engineering development process can be defined as a two-way procedural mapping between a user and a system, where the system specification must satisfy user requirements and the user requirements must help the development of system specification. It is illustrated in Figures 1, 2, and 3. The user creates requirements and the software engineer extracts features from the requirements for the system, and develops system specifications, focusing on the extraction of efficient software characteristics to build a software system. The main goal of the proposed SETh framework is to detail the procedural mapping, using the standard definitions of software engineering and the recommendations provided by SEEK [1, 12], and define an eight layer model as follows (these are the standard components of software engineering and can be found in software engineering sources, e.g. [11], [15], and [14]):

· Project Definition: A project definition mainly includes a detailed description of the problems to be addressed, and the scope of the entire project.

· User Requirements: The user requirements include domain analysis, feature extraction, and the development of user requirements document. It must also include a use-case analysis to determine the complexity of a system.

· Models: The models are the mapping of the user requirements to a set of client objects by using several diagrams, called UML diagrams, which include class diagrams, sequence diagrams, and activity diagrams.

· Graphical User Interface: It is an organized visual display using several meaningful client objects that can provide users an user-friendly access environment to systems' functionalities.

· Designs: The designs are software patterns - also known as design patterns - that can facilitate coding and model implementations using the client objects so that the user requirements can be satisfied.

· Coding: It transforms the design patterns into a set of specific codes (software modules) that implement these patterns using a programming language, such as Java programming, to develop the final software system (a collection of connected software modules) or product.

· System Specification: It must map user requirements clearly and structurally to a set of specifications from which a system can be developed. It also includes testing procedures to validate working mechanism of the software system and its subsystems.

· Software Product: It is the end product of the software engineering process. It must satisfy the system specification and meet (or satisfy) the expectation of the users.

Using these software engineering descriptors, a number of visual models are proposed, developed, and presented in this section to facilitate the understanding of the basics, domain and analysis, concepts, processes, relations, and methodology, and understand their tight connections

during the project development using software engineering. The proposed SETh framework recommends the use of these descriptors in a loop as illustrated in Figure 1.

2.1 The Basics of Software Engineering

The basic elements of software engineering are the users and the systems. These elements and the essential communications between them are presented in the TBoSE model illustrated in Figure 1. One of the communications flow is the sharing of a user requirements document, and the other flow is the sharing of a system specification document. The revisions of these documents are the other essential communication flow within the user and the system elements themselves. The goal of a software engineering methodology is to achieve a win-win situation for the user and the system. The SETh framework recommends students and software engineers identify and generate TBoSE elements first to initiate the software engineering methodology successfully for their projects.

2.2 The Domain of Software Engineering

The domain of software engineering model is a collection of TBoSE models as illustrated in Figure 2. In general, a system consists of many subsystems or connected with other systems to be more efficient and function effectively. However, a subsystem is a system itself; therefore, a TBoSE model must be defined for a subsystem as shown in Figure 1. As such the proposed SETh framework defines the software engineering problem domain as a space that is formed by many tuples (user, system) as illustrated in Figure 2. Students and software engineers who construct this domain can easily visualize the entire system and efficiently build the system to meet the users expectations. Therefore, the SETh framework emphasizes the inclusion of the construction of TDoSE in software engineering curriculum and the practices. This model allow us to address the emerging problem associated with global, collaborative, and big data software engineering and applications.

2.3 The Concept of Software Engineering

The concept of software engineering is described in Figure 3 using the (user, system) tuples - the basic elements of software engineering. It first illustrates the use of use-cases (and use-case analysis) to connect a user and a system through an interface (e.g. Graphical User Interface - GUI). It also illustrates that the user develops requirements (i.e., the user requirements document), and provides clear information for a software engineer to extract features and construct a feature space. A technique that a software engineer can use to extract features, using domain analysis, is called Feature-Oriented Domain Analysis. This technique is described in the document available at [8]. The software engineer then can develop system specifications (i.e., the system specification document) by modeling the relations between software modules and features. Hence, the system specification can be defined as a clear description of many model diagrams (e.g. UML diagrams) and design patterns related to the envisioned system. Hence, it can be described mathematically:

$$M = f(F), f : F \rightarrow M. \quad (1)$$

Although, the model is described with a mathematical function *f*, in practice it can be a procedural approach (model) that develops software modules step-by-step from user required features for their envisioned system. If it is defined mathematically, it can help the optimal selection of software modules for features, and it is not the scope of this paper. However, it will be addressed as a part of the planned future educational research.

2.4 The Processes of Software Engineering

The processes of software engineering model TPoSE presented in Figure 4 provides an eight layers architecture for software engineering processes. It introduces all the software engineering components that are suitable for the proposed SETh framework. The eight layers of this visual model are related to: 1. project definitions, 2. users, 3. models, 4. GUI, 5. designs, 6. coding, 7. system, and 8. software product. The layers 1 and 2 involves with the processes associated with the requirements document and domain analysis; the layer 3 involves with the development UML diagrams with use-case analysis; layer 4 involves with the construction of GUI clients; layer 5 involves with the selection of design patterns; layer 6 involves with the coding (e.g. Java); and layers 7 and 8 involves with the development of specification documents and software testing, respectively. The architecture of TPoSE helps the extraction of features (i.e. the construction of a feature space) $F_1, F_2, \ldots, F_m$, the development of UML diagrams $U_1, U_2, \ldots, U_n$, establishment of GUI clients $G_1, G_2, \ldots, G_o$, selection of design patterns $D_1, D_2, \ldots, D_p$, and software modules $M_1, M_2, \ldots, M_q$. The object orientation technology used on the system development and coding can benefit from these objects and the associated processes.

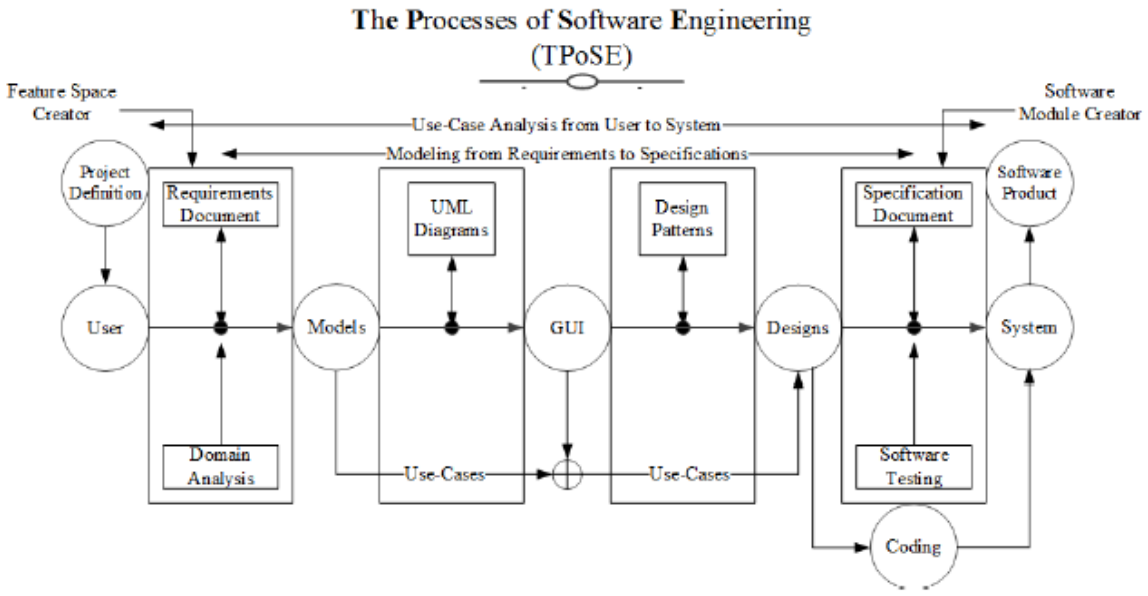

Figure 4. TPoSE: Proposed approach with associated components. It provides an eight layers architecture, represented by circles - project definition, user, models, GUI, designs, coding, system, and software product. Four interfaces between the layers are presented in vertical rectangles.

2.5 The Relations of Software Engineering

One of the contributors that can facilitate backward tracking/tracing and forward tracking/tracing capabilities is the TRoSE visual tool presented in Figure 5. It connects the objects and processes identified by the TPoSE model. As an example, the TRoSE model shows the connectedness of features F_3 with the software module M_1 through the UML diagram U_2, GUI client G_2, and design pattern D_3. The connected path (an example) that is established clearly shows a footprint which facilitates backward tracking/tracing and forward tracking/tracing. The tracking occurs when a clear footprint is available through one-to-one mapping, and the tracing occurs when there is an ambiguity in the path traversal due to one-to-many or many-to-one mappings.

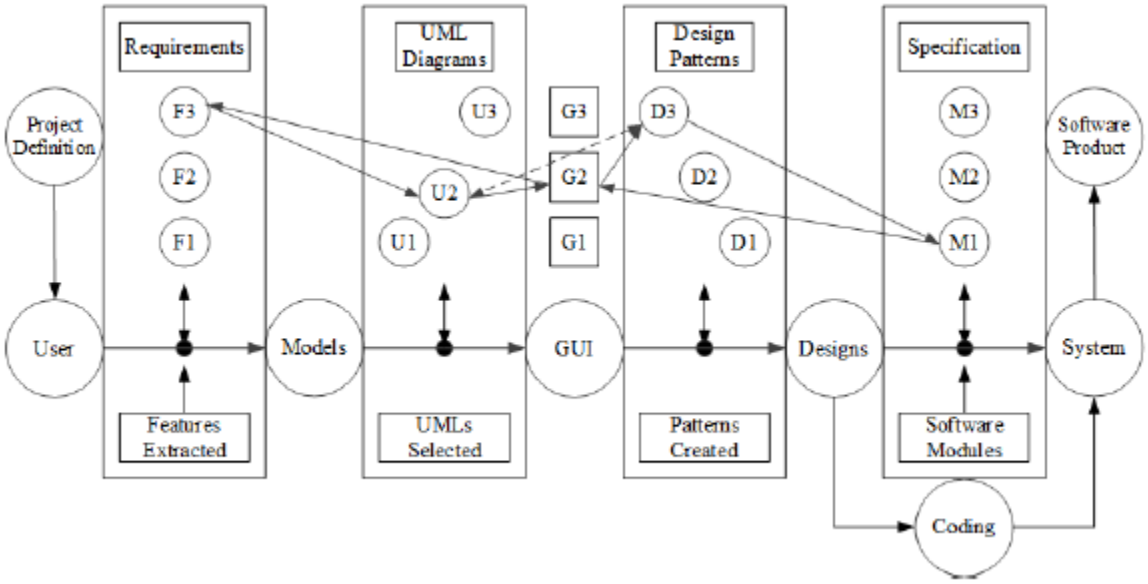

Figure 5. TRoSE: F - Features, U - UML Diagrams, G - GUI Components, D - Design Patterns, and M - Software Modules.

It creates an artifact, called *triangularization*, which helps develop a systematic approach to revise the model frequently during the system development. It is illustrated by the pentagon model presented in Figure 6. This pentagon model allows students and software engineers to recognize the connected components that should be focused while performing revisions on the entire software engineering process, and writing user requirements documents, and specification documents along with a document that described the GUI clients in detail. Thus SET*h* framework requires three documents: user requirements, system specification, and GUI description. For example, the components *F*, *U*, and G are the three components that must be focused more while writing user requirements document; *G*, *D*, and *M* are the three components that must be focused more while developing system specification documents; and *D*, *G*, *M* are the three components that must be focused more while writing GUI description document. Hence, when the envisioned system is very complex, the triangularization is a godsend for students and software engineers. This is one of the areas our future research will focus on using several case-studies with industry collaborations.

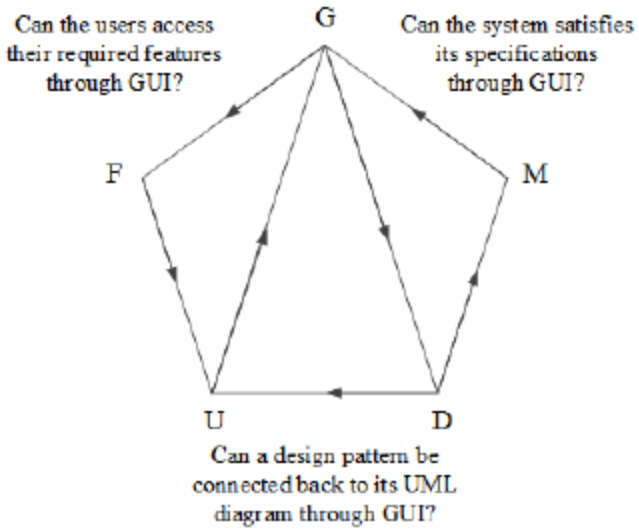

Figure 6. Triangularization: A pentagon model for describing triangularization characteristics of the TRoSE model.

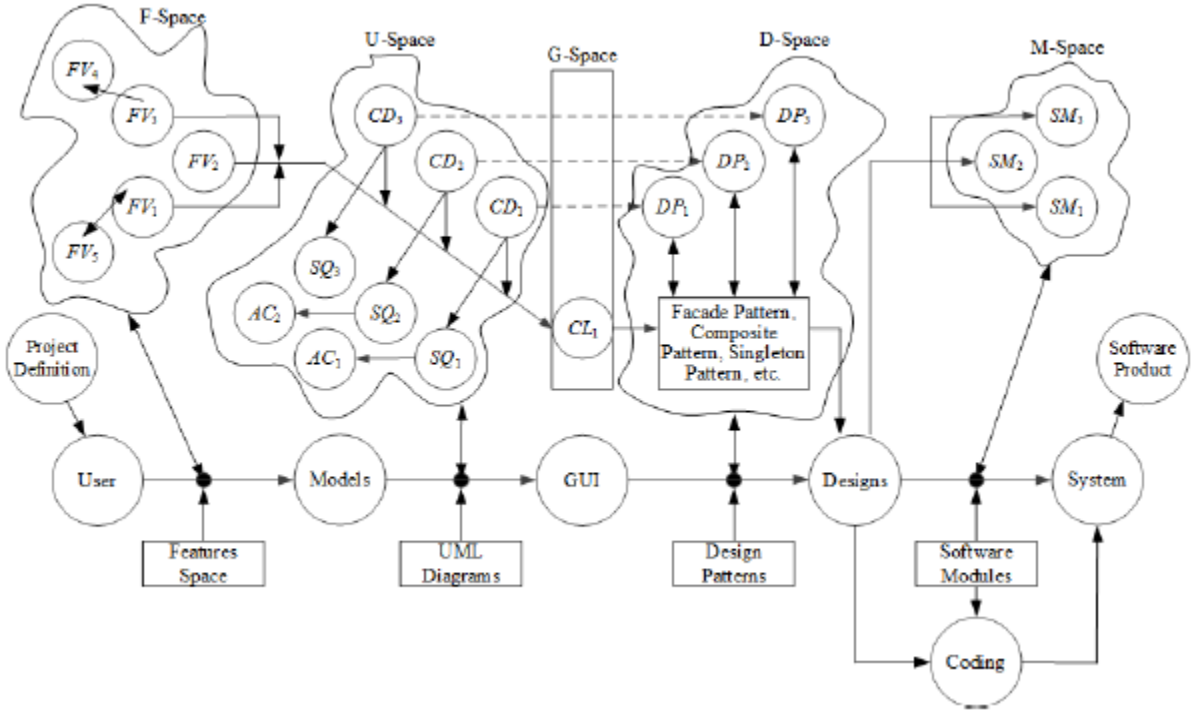

Figure 7. TMoSE: The ultimate goal of software engineering is to map the features space (user requirements) to software modules (specifications) of the system through a series of steps defined by models, graphical user interface, and coding along with model diagrams and design patterns.

2.6 The Methodology of Software Engineering

The methodology of software engineering TMoSE presented in Figure 7 is a model that provides more details of TRoSE model and its activities; hence, it also facilitates the backward tracking/tracing and forward tracking/tracing capabilities, but efficiently. This model allows visualization of steps involved in the development of software modules from the required features through models (e.g. class, sequence, and activity diagrams), GUI (e.g. client 1, client 2, and client 3), coding (e.g. using design patterns such as facade, singleton, and composite patterns one could create programs). It introduces five spaces, F-spaces, U-space, G-space, D-space, and M-space, and their ordered connectivity to help the development and the implementation of the software engineering methodology. Figure 7 illustrates that a software engineer creates a F-space (the space that consists of all the user required features) based on a domain analysis, then constructs a U-spaces (that consists of a set of UML diagrams), creates G-space that consists of a set of clients for GUI, implement D-space with suitable design patterns for the clients so that the coding can be performed and a system specification can be developed efficiently, and finally the M-space that is constructed with several software modules. The TMoSE model can allow a software engineer, who developed this model or another software engineer to revise the model, with an easy backward or forward tracking/tracing functionalities via connected paths.

3 DISCUSSION USING AN EXAMPLE

The main purposes of the visual models of the framework are to facilitate the identification, the extraction, and the development of the five spaces, F-space, U-space, G-space, D-space, and M-space, and build connections between them as illustrated in TMoSE model. Therefore, in this section, a simple example is selected to illustrate that process. The example is related to the development of a software system for the National Highways Traffic Safety Administration (NHTSA) that analyzes a fatal crash incidents.

A fatal crash incidents data set may provide many relevant information about the crash incidents occurred in highways, cities, rural areas, and so on. However, this section only considers its subsystem that uses the location information and the frequency of the crash incidents occurred at that locations to illustrate the proposed framework. Let us assume that the location information is described by the longitude and latitude variables along with the number of fatal crash incidents occurred at those locations. Suppose the user (i.e. NHTSA) wants a subsystem that has the following functionalities:

· It can count the number of fatal crash incidents occurred within a range of longitude values so that the user can study them in vertically segmented regions.

· It can count the number of fatal crash incidents occurred within a range of latitude values so that the user can study them in horizontally segmented regions.

· It can count the number of fatal crash incidents occurred within a range of longitude values and a range of latitude values so that the user can study them in grids.

These three user requirements clearly indicate that the subsystem should have a feature that allows the user to input longitude information (x), another feature that allows the user to input latitude information (y), and a third feature that allows the user to input both longitude and latitude information (x, y) as a tuple. The subsystem then should output the crash frequencies, as responses, in these three different cases. Hence, the user requirements are clearly defined and they can be used to generate the user requirement document by following the guidelines provides in TCoSE and TDoSE diagrams.

3.1 Components of TPoSE Model

The goal of TPoSE model is the listing of the components required for the development of the system specification based on the user requirements. The list of components needed for the TPoSE model are the features required by the user, and the UML diagrams, design patterns, and software modules selected by the software engineer for the specification. These components can help build the F-space, U-space, G-space, D-space, and M-space of the model. For this crash data analytics system, we can generate the following lists:

· Features: There are two feature that we can identify from the user requirements, and they are: Longitude (F_1) and Latitude (F_2). They provide the structural and logical characteristics of the system.

· UML diagrams: The structural characteristics will be used by the user depending on whether the longitude calculation (U_1) or latitude calculation (U_2). They provide the functional and logical characteristics of the system.

· GUI: A single client (G_1) must be established because it can sufficiently allow users interact with the system.

· Design Patterns: The features are independent as per user requirement; however, they must be combined (D_1) if the user requested.

· Software Modules: A single module (M_1) written in pseudo code using structural, functional, and logical characteristics of the objects identified.

These components must be combined to create a spatial structure for the TPoSE model so that it can be included in the specification document. This structure is defined in TRoSE and TMoSE models.

3.2 Components of TRoSE Model

For the purpose of TRoSE model, we need to build the connections between the components that we listed for the crash data analytics system above in TPoSE model. We can generate simple connection structures as follows:

$$F_1 \rightarrow U_1 \rightarrow G_1 \rightarrow D_1 \rightarrow M_1 \text{ and } F_2 \rightarrow U_2 \rightarrow G_1 \rightarrow D_1 \rightarrow M_1$$

We can easily observe a fork-shaped structure for the connections between the components.

3.3 Components of TMoSE Model

For the purpose of developing TMoSE model, we need to generate feature vectors. For example, longitude and latitude are considered as features; hence, a feature vector is a tuple, such as (Longitude min, Longitude max) and (Latitude min, Latitude max).

· F-space: Each user requirement is considered as a feature and combined to form a hierarchy of features as appropriate.
  – FV_1 = (*Longitude min, Longitude max*)
  – FV_2 = (*Latitude min, Latitude max*)
  – FV_3 = ((*Longitude min, Longitude max*), (*Latitude min, Latitude max*))

· U-space: Objects are selected to define features and connect them as appropriate. It facilitates object-orientation.
  – CD_1 = *LongitudeUser*
  – CD_2 = *LatitudeUser*
  – CD_3 = *LongitudeLatitudeUser*

· G-space:Clients are established for a graphical user interface, and at least one client is defined for the system.
  – *CL = SingleClient*

· D-space: Design patterns are selected. Facade pattern can help return a tuple (x, y) to the client, instead of returning all x values and then y values to the client.
  – *DP = CrashFacade*

· M-space: Software modules are developed in the form of pseudo code .
  – *SM = CrashDataAnalytics*

The M-Space is presented in Listing 1 which incorporates a Facade pattern - a commonly used pattern in many applications including in a hotel and flight bookings example, presented at [16].

Finally, the triangularization can also help organize the format and the content of the three documents: user requirements, system specifications, and GUI description. Especially, students in general have difficulties is writing and presenting documents, and this triangularization guides them to select, connect, describe the needed components. It can also help educators in the evaluation of students' submitted work, such as assignments and projects.

4 CONCLUSION

The paper presented a SETh framework and a set of visual models that can help students learn, educators teach, and software engineers practice software engineering theory and principles efficiently. It can also help students extract knowledge throughout the phases of software engineering implementation, and software engineers apply them to extract, modify, and reuse the

knowledge for the greenfield or evolutionary type projects. These visual models can also help apply learning algorithms to make the software engineering applications more efficient. The SETh framework and the models can also be easily customized to integrate new experiences that a software engineer acquired. The backward tracking/tracing and forward tracking/tracing skills that the students or software engineers acquired from the use of these concepts can help address the problems resulted from the vagueness of the user requirements and the complexity of the envisioned system. Software engineers can also use these concepts to rebuild or create a new system. Hence, it can benefit both software engineering education and practices. The SETh framework and the visual models can also contribute to the new direction global perspective and collaborative software engineering and the emerging needs to develop and manage big data systems. A future research is to define the relations between the user and system elements mathematically, study the optimal selection of software modules for user-required features, and automate the models. It means the function (or mapping) $f: F \rightarrow M$ will be established and studied in detail. The future research will include a detail study on the triangularization aspect of the SETh framework as well. While SETh is generic, its multi-space modeling structure can make it highly suitable for the emerging data science applications. The multi-space structure can also help integrate SETh with agile development process and make the combination a hybrid software engineering framework.

REFERENCES

[1] Mark Ardis, David Budgen, Gregory W Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. 2015. SE 2014: curriculum guidelines for undergraduate degree programs in software engineering. Computer 48, 11 (2015), 106–109.

[2] Sarah Beecham, Tony Clear, Daniela Damian, John Barr, John Noll, and Walt Scacchi. 2017. How best to teach global software engineering? Educators are divided. IEEE Software 34, 1 (2017), 16–19.

[3] Magdalena Beslmeisl, Rebecca Reuter, and Jürgen Mottok. 2016. The Importance of Writing in Software Engineering Education. In International Conference on Interactive Collaborative Learning. Springer, 315–321.

[4] Tony Clear, Sarah Beecham, John Barr, Mats Daniels, Roger McDermott, Michael Oudshoorn, Airina Savickaite, and John Noll. 2015. Challenges and recommendations for the design and conduct of global software engineering courses: A systematic review. In Proc. of the 2015 ITiCSE on Work Group Reports. ACM, 1–39.

[5] Carlo Ghezzi and Dino Mandrioli. 2005. The challenges of software engineering education. In Proceedings of the 27th international conference on Software engineering. ACM, 637–638.

[6] Abdelwahab Hamou-Lhadj and Timothy C Lethbridge. 2004. A survey of trace exploration tools and techniques. In Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, 42–55.

[7] Daniel Kadenbach and Carsten Kleiner. 2016. Evaluation of Collaborative Development Environments for Software Engineering Courses in Higher Education. In International Conference on Social Computing and Social Media. Springer, 365–372.

[8] Kyo Kang, Sholom Cohen, James Hess, William Novak, and Spencer Peterson. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. (Jan. 1990). Retrieved January 6, 2018 from https://www.sei.cmu.edu/reports/90tr021.pdf

[9] Vijay Dipti Kumar and Paulo Alencar. 2016. Software engineering for big data projects: Domains, methodologies and gaps. In Big Data (Big Data), 2016 IEEE International Conference on. IEEE, 2886–2895.

[10] Carl Landwehr, Jochen Ludewig, Robert Meersman, David Lorge Parnas, Peretz Shoval, Yair Wand, David Weiss, and Elaine Weyuker. 2017. Software Systems Engineering programmes a capability approach. Journal of Systems and Software 125 (2017), 354–364.

[11] Timothy Christian Lethbridge and Robert Laganiere. 2005. Object-oriented software engineering. McGraw-Hill New York.

[12] Timothy C Lethbridge, Richard J LeBlanc Jr, Ann E Kelley Sobel, Thomas B Hilburn, and Jorge L Diaz-Herrera. 2006. SE2004: Recommendations for undergraduate software engineering curricula. IEEE software 23, 6 (2006).

[13] Nazim H Madhavji, Andriy Miranskyy, and Kostas Kontogiannis. 2015. Big picture of big data software engineering: with example research challenges. In Big Data Software Engineering, IEEE/ACM 1st International Workshop on. 11–14.

[14] Ian Sommerville. 2010. Software engineering. Addison-wesley.

[15] Rod Stephens. 2015. Beginning software engineering. John Wiley & Sons.

[16] James Sugrue. 2010. Facade Pattern Tutorial with Java Examples. (Jan. 2010). Retrieved January 6, 2018 from https://dzone.com/articles/design-patterns-uncovered-1

[17] Will Tracz. 1995. DSSA (domain-specific software architecture): pedagogical example. ACM SIGSOFT Software Engineering Notes 20, 3 (1995), 49–62.

[18] Jim Whitehead, Ivan Mistrík, John Grundy, and André Van der Hoek. 2010. Collaborative software engineering: concepts and techniques. In Collaborative Software Engineering. Springer, 1–30.