A Secure Software Engineering Design Framework for Educational Purpose

Abel A. Reyes Angulo[1], Xiaoli Yang[1], Quamar Niyaz[1], Sidike Paheding[2], Ahmad Y Javaid[3]
[1]Electrical and Computer Engineering, Purdue University Northwest, IN 46323
[2]Applied Computing, Michigan Technological University, MI 49931
[3]Electrical Engineering and Computer Science, The University of Toledo, OH 43606
{areyesan, yangx, qniyaz}@pnw.edu, spahedin@mtu.edu, ahmad.javaid@utoledo.edu

***Abstract*—Ensuring software security is a critical task for a deliverable software system in today's world, and its proper implementation guarantees the quality and security of the information ingested, stored, and processed by the system. It is imperative to introduce computer science and computer engineering students (CS/CE) with the secure software design practices early in their curriculum. This approach will help them understand fundamentals of secure programming, vulnerabilities in software systems, and secure software development before joining the industry workforce. In this paper, we propose an educational framework that integrates software security concepts in a software engineering design course. We envision that the framework will engage CS/CE students applying security principles and practices in different phases of the software development life cycle (SDLC) process. Our work focuses on review of common security requirements, policies, and mechanisms related to specific use cases as well as how those requirements are defined during the software design.**

***Index Terms*—software engineering, software development life cycle, software security, secure design.**

I. Introduction

One of the primary reasons that adversaries discover and exploit vulnerabilities in various software systems is the lack of adoption of security principles and practices in programming and software design [1]. According to Taylor and Kaza [2], most of the successful attacks on federal computers in the United States are resulted from software bugs and poor quality of software, highlighting the fact that the poor quality of delivered software in terms of security is a concern, and the industry is in dire need of professionals with strong knowledge of secure software design and development. It is critical to integrate secure coding practices at the early stages of programming training for CS/CE students in order to academically train those professionals. According to Chi et al. [3], students in programming classes often associate their goals with the functionality of the software they create, paying little attention to how the code may fail or produce improper results. It is necessary to embrace security practices into the entire software development process, i.e. software development life cycle (SDLC). Looking at the recent growth in software security issues, it is critical in the software industry that embedding security into a system during its design and development phases is better than adding security into a finished product. According to Conklin and Dietrich [4], it is essential to teach students secure programming techniques during their first interaction with programming rather than change their programming style later, and encourage students to adopt these good programming practices throughout their college education. It adds a training component to the CS/CE education exposing students to real-world security problems such as software vulnerabilities and teaching them how to mitigate those vulnerabilities without compromising the software's functionality. This is important for increasing the knowledge and skills of future workforce in software development industry [3]. In this paper, we present a secure software design framework for educational purposes to train CS/CE students incorporating security policies and mechanisms into software design and development. The primary goal of this project is to promote secure software development in the CS/CE undergraduate curriculum.

The rest of the paper is structured as follows. Section II presents a summary of related work in secure software design. Section III discusses the proposed framework for implementing security in the SDLC. In Section IV, the proposed framework is illustrated using a software design case study. Finally, Section V concludes the paper with a brief overview of future enhancement of the framework.

II. Related Work

This section briefly reviews previous efforts towards secure software development and discusses the motivation of this work. All the works discussed in this section share a common premise – the importance of including secure components during the software development phase.

The Unified Modeling Language (UML) [5] assists software engineers during the software design phase, but it does not offer features to integrate security in the design. In order to bring the security features in the software design, a UML extension called UMLpac was proposed by Peterson et al. [6]. It supports security packages that maintain all abstraction levels and the original system diagram while providing comprehensive security features. In this work, we incorporate the process described in UMLpac to integrate security components during the static modeling of the case study used in our framework. Jurjens et al. [7] introduced a method, UMLsec, for effectively developing behavioral and structural UML design models, which are based on security requirements. Each step in the method is supported by model generation rules in

978-1-6654-8009-3/22/$31.00 ©2022 IEEE

the UML profile so that security requirements become less fallible, regular, and systematic engineering activities. Hatebur et al. [8] described the systematic approach of UMLsec step-by-step through the implementation of a case study. In our work, we followed a few steps from their implementation, such as the use of functional requirements, security requirement and security domain knowledge tables to design the dynamic models with security components for our case study.

Lodderstedt et al. [9] incorporated security requirements into UML, where they proposed a technique called SecureUML to represent access control and authorization in the UML. This technique can be used to supplement the implementation of other frameworks, such as UMLpac or UMLsec. Doan et al. [10] proposed a UML framework for security design and constraints checking that includes role-based control, mandatory access control, and life time sensitivity. The resulting framework encourages secure software design by tracking security requirements as UML elements are added, modified, and removed. Georg et al. [11] proposed an aspect-oriented modeling that allows developers to encapsulate security concerns into a design in an effective and systematic manner. The authors have emphasized the illustration of security concerns in various UML models in their work.

The reviewed works assisted us in identifying the importance of security implementation during the design phase of software development as well as provided us with the conceptual tools to develop this secure software design framework for an educational purpose in undergraduate-level software engineering courses.

III. The Proposed Framework

In this section, we describe our educational framework for implementing security in the SDLC including security requirements and security packages in the design phase. The SDLC consists of various phases including requirements gathering, design, implementation, testing, and deployment of a software system. There are various SDLC models (e.g., waterfall, spiral, and agile) used for software development based on the project size and complexity; majority of them share the common phases. There is a need to build secure software from its inception as it will incorporate security into each SDLC phase; failing to do so may bring security flaws and changes must be made once the software is deployed, which is an additional and time-consuming task. Following are the most common phases identified within various SDLC models [12]:

1) *Requirements analysis*: A requirement is defined as a set of tasks the software should be able to do. The software requirements are gathered, listed, and analyzed during this phase.
2) *Design*: This phase defines how to design the system to meet requirements gathered in the previous phase.
3) *Implementation*: In this phase, software developers implement the design.
4) *Testing/Assurance*: During this phase, the testing or quality assurance team ensures that the implementation does what it is supposed to do in an efficient and error-free manner.

SDLC Phases	Regular tasks	Secure software tasks
Requirements	• Functional Requirements • Use cases	• *Security Requirements* • *Abuse cases*
Design	• Static and Dynamic Modeling	• *Architectural Risk Analysis* • *Security-oriented Design*
Implementation	• Coding	• *Code review*
Testing	• Functional Testing	• *Risk-based Security Tests* • *Penetration Testing*
(a)	(b)	(c)

Fig. 1. SDLC phases integrated with security related tasks.

The proposed educational framework will engage students to include security components for each SDLC phase to make their developed software robust and reduce security vulnerabilities in the software. Fig. 1 depicts a schematic of the key SDLC phases that will be enhanced for secure software development. In the figure, column (a) lists the SDLC phases, column (b) lists the tasks for each phase in a regular SDLC, and column (c) shows the additional tasks to be completed in each phase as part of the secure SDLC.

In this preliminary work, we focused on including security components for the first two phases of SDLC, i.e. requirements analysis and software design. The tasks during the requirements analysis phase includes not only functional requirements gathering, but also determining security requirements based on the functional ones. These requirements include security policies such as confidentiality, integrity, and availability along with the supporting mechanisms such as authentication, authorization, and auditing [13]. The policies control sensitive information from being leaked to unauthorized parties (confidentiality and privacy), ensure that sensitive information is not tampered by unauthorized parties (integrity) and a system is responsive to requests (availability). The policies must be supported by mechanisms; for example, an authentication mechanism helps to prevent unauthorized users from using the software. Furthermore, security requirements are influenced by external factors such as general policy concerns resulting from country regulations or organizational values. Following the listing of security requirements, abuse cases are used to identify tasks that a software system is not supposed to perform [14].

During the design phase, the static and dynamic models will be created using two approaches: architectural risk analysis and security-oriented design. An architectural risk analysis, also known as threat modeling, attempts to identify weakness and vulnerabilities during the design phase so that developers should be aware of cybersecurity issues and how they can be exploited by different types of users during the design phase [15]. The security-oriented design approach, also known as threat-driven design, consists in the extended analysis

of system requirements, covering use cases, misuse cases and mitigation use case [16]. It requires developers eliciting different responses from various users, including normal users, malicious users, and attackers, in dynamic models. Fig. 2 depicts the tasks required for the two phases of SDLC with security components.

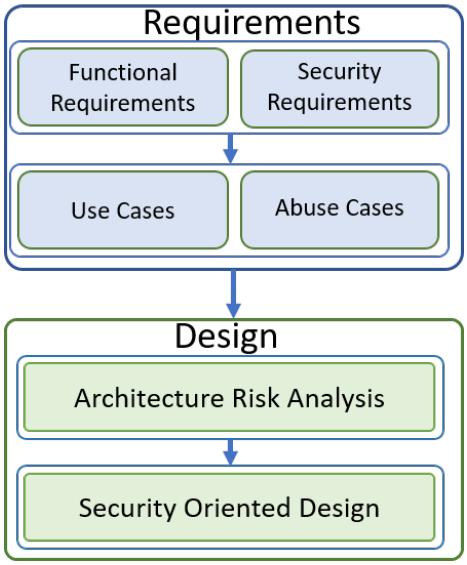

Fig. 2. Requirements and design phase in SDLC with secure component.

In our educational framework, we use UMLpac [9] and UMLsec [7] approaches for integrating security in requirements analysis and design phases. UMLpac was presented as an extension of UML with the capability to enable security features in such a way that they can be easily integrated into the traditional UML class diagrams. The main idea behind the implementation of UMLpac is to enable software requirement analysts and designers to layout security components directly into UML diagrams (i.e. class diagrams) of a system without loosing the characteristic level of abstraction. On the other hand, UMLsec was proposed as a profile to develop and analyze secure system models. UMLsec is based on elements such as stereotypes, tags, and constraints, which are used to specify security requirements for a software system. Stereotypes, in this context, are used to label security-critical parts, ensure dependent parts, analyze behavior models, and introduce attack models. The usage of stereotypes is mentioned in UMLsec to create secure UML models based on security requirements. The purpose of UMLsec is to identify possible security vulnerabilities at an early stage in the SDLC.

IV. Case Study for the Educational Framework

We incorporate security components in a software design case study adopted from a case study based textbook on software modeling and design [17]. In the educational framework, secure software development will be illustrated by enhancing activities involved in the requirement gathering and design phases of the SDLC. The adopted case study implements a software for an ATM banking system, which has been described as follows in the textbook: *"A bank provides several ATMs to its customers that are linked to a centralized server through Internet. Each ATM has a card reader, display, keypad, cash dispenser, and receipt printer. Customers can withdraw money from their accounts, perform balance queries, and transfer funds to others' accounts. An ATM operator staff can start and stop an ATM in order to refill the cash dispenser with bills and perform maintenance."* The reason for using this case study is that similar case studies have been discussed in several learning resources of software engineering design.

A. Requirements gathering

This is the first phase in the SDLC based on the problem description. It includes the task of functional requirements and we will include security requirements in it. System requirements are represented in a traditional SDLC using a *system context diagram*) and a *use case* diagram, which describe the functionality and behavior of the system. These diagrams serve as reference points for the software design phase. Furthermore, use case diagram is supplemented by the implementation of activity diagrams that explain how each use case works as a sequence of activities performed by the user, system(s), or interaction between them. For secure SDLC, we include system vulnerabilities and tasks that the software is not supposed to perform in addition to functional requirements. The security requirements describe potential vulnerabilities of the system based on functional requirements from the problem description and they are represented by *abuse case diagram*, context diagram with environmental description, security requirement, and security domain tables.

1) Functional Requirements: A use case diagram graphically depicts the interaction between users and the system in order to meet functional requirements [18]. The ATM Banking system's use case diagram is depicted in Fig. 3 (a). According to UMLsec a table for functional requirements is required where developers can list functional requirements of the software system from the use case description. For each functional requirement, this table lists the requirement identifier (e.g. R1, R2, . . . , Rn), description of the requirement, list of instances involved in the requirement, and list of constraints (e.g. physical constraints). Table I displays a list of functional requirements for the ATM Banking system, including constraints and the instance references that interact with the system for each requirement.

2) Security Requirements: Security requirements for functional requirements are expressed through stereotype ≪`complements` ≫. They are listed in Table II corresponding to functional requirements mentioned in Table I. These security requirements should comply with security policies as a response to system vulnerabilities. For example, as shown in the first row of Table II, the required integrity supports the system's safety and protects the system from any data manipulation. Based on the corresponding use case diagram, an abuse case diagram is created, and this captures security vulnerabilities in the use case diagram. Fig. 3 (a) and (b) show

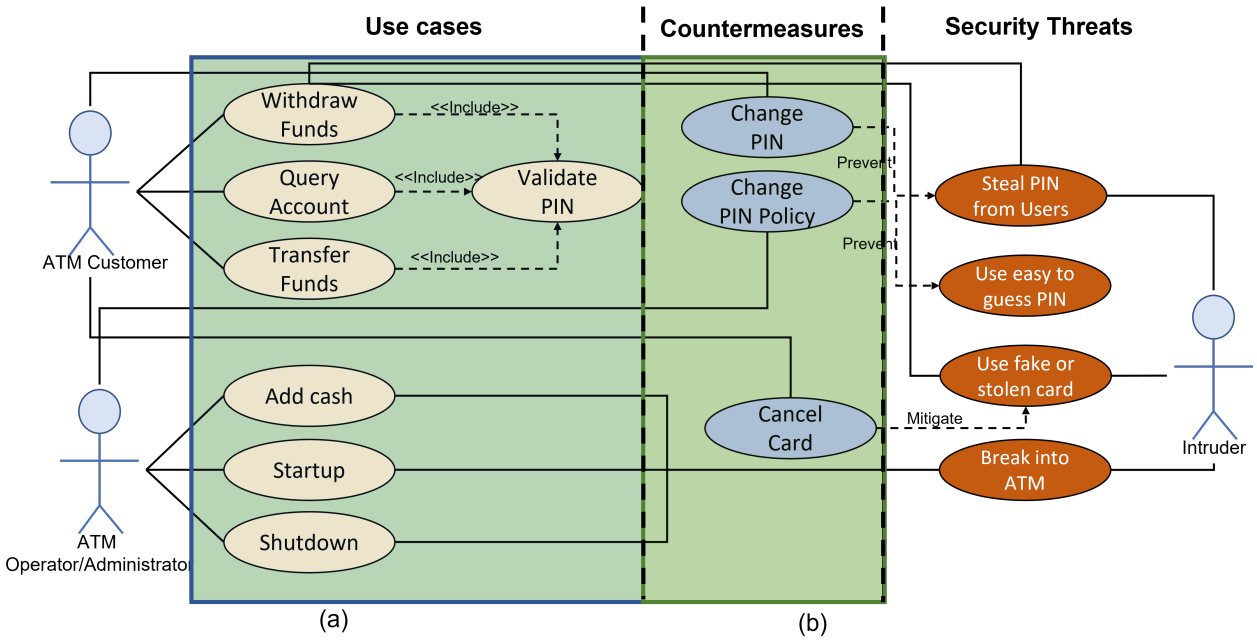

Fig. 3. Use case and Abuse case diagrams of an ATM Banking system

TABLE I
FUNCTIONAL REQUIREMENTS FOR ATM BANKING SYSTEM CASE STUDY GIVEN IN [17]

Id	Requirement	≪RefersTo≫	≪ Constraints ≫
R1	System should allow users to withdraw funds from their accounts	ATM and Customer Banking System	Keypad/Display, Card Reader, Banking System, Cash Dispenser
R2	System should allow queries from users.	ATM and Customer Banking System	Keypad/Display, Card Reader, Banking System
R3	A user should be able to transfer funds from his/her bank account to another user's bank account.	ATM and Customer Banking System	Keypad/Display, Card Reader, Banking System
R4	An operator should be able to add money to ATM	ATM Operator and Banking System	Keypad/Display, Cash Dispenser
R5	An operator should be able to start ATM	ATM Operator and Banking System	Keypad/Display, Banking System
R6	An operator should be able shutdown ATM	ATM Operator and Banking System.	Keypad/Display, Banking System.

the abuse case diagram for the ATM Banking System, which introduces potential vulnerabilities that an attacker could exploit in the ATM Banking System and possible mitigation methods. The assailant is played by another actor *intruder*. Each vulnerability is represented in orange color use case, and the mitigation are represented in blue use case. For example, use case *Change PIN* is intended to prevent use case *Steal PIN* from users; thus, associated with actor *ATM customer*, who will perform/request this mitigation use case via the ATM Banking System.

B. Software Design: Static Modeling

In this section of the design phase, we leverage UMLpac [9]. UMLPac integrated the concept of security tile/package to maintain the level of abstraction between the system implementation and security features. It has been analyzed that security tiles/packages work well for object-oriented systems. UMLpac employs a stereotype construct known as ≪`security package`≫ to modify class diagrams to represent security features. There are three stereotype attributes in it that should be highlighted: ≪`security tile`≫, ≪`risk factor`≫, and ≪`security descriptor`≫. A *security tile* defines a security package's security descriptor. A *risk factor* is a value assigned by system analysts based on the available information about vulnerabilities. This value can be a discrete number or a label from a finite set (i.e. dangerous, harmless, regular). The type of *security descriptor* identifies what information is in the security tile (e.g. principles, survivability, and accountability, third-party software and rules).

Fig. 4 depicts an example of a security tile called *Data Backup*, which will be linked to the secure package Data Backup and used in class diagrams with the secure component. A class diagram is a type of static diagram that depicts the structure of a system by illustrating classes and their attributes, operations, and object relationships [18]. The entities that interact for one of the listed requirements (R1) in Table I are shown in Fig. 5 for the *Withdraw Funds* use case. To ensure information security and promote security policies such as confidentiality, integrity, and availability, the following security packages will be related to this use case:

- *Validate Input*: to prevent unauthorized access during the customer input process.

TABLE II
SECURITY REQUIREMENTS FOR THE ATM BANKING SYSTEM

No.	Requirement	≪Complements≫	≪RefersTo≫	≪Constraints≫/Mechanism
1	ATM configuration should be protected from modification for customers against attackers or ATM operator should be informed.	R4, R5, R6	Configuration is asset, ATM Banking system knows asset, ATM Operator is stakeholder, against Attacker	Terminal/Authentication mechanism i.e., MAC (Message Authentication Code).
2	PIN should be protected to be stolen or guessed by an attacker for a customer.	R1, R2, R3	Shared keys are assets, ATM customer is stakeholder, against Attacker	Terminal/key exchange (KE)
3	ATM customer should be able to cancel cards to avoid unauthorized use.	R1	Card reader is asset, ATM customer is stakeholder, against Attacker.	Card reader/encryption
4	System should be protected against potential break into the ATM.	R3	Machine is asset, ATM operator is stakeholder, against Attacker.	Machine/Authentication
5	System should provide the ability to work consistently until failure, and recovery to the most recent working state.	R5, R6	Machine is asset, ATM customer is stakeholder, against Attacker.	Machine/Auditability
6	System must avoid the injection of improper data/script in the input fields	R1, R2, R3	I/O devices are assets, ATM customer is stakeholder, against Attacker.	Keypad/input validation

TABLE III
SECURITY DOMAIN KNOWLEDGE TABLE FOR THE ATM BANKING SYSTEM

No.	Requirement	≪Complements≫	≪RefersTo≫	≪Constraints≫/Mechanism
1	System should protect the configuration through a proper authentication of the users and the authorization of privileges according to each of their roles.	R4, R5, R6	System is assets, ATM customer and operator are stakeholder, against Attacker	Authentication/KE.
2	Communication between system and database should be protected against impersonation cases from attackers.	R1, R2, R3	System and database are assets, users are stakeholders, against Attacker	Encryption/KE.
3	System must record and administer the Log activities from the user for auditability purposes.	R1, R2, R3	Machine is asset, ATM customer and Operator are stakeholder, against Attacker.	Accountability/Log Management
4	System must constantly Backup the most recently working state to recover from failure.	R5, R6	ATM Operator and Machine are assets, ATM Customer is stakeholder, against attacker.	Database survivability/Backup

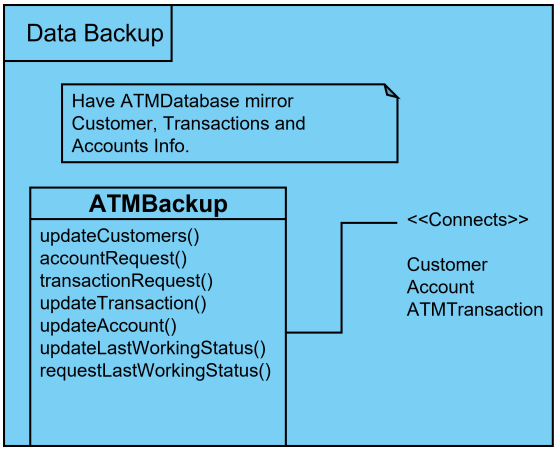

Fig. 4. Data backup Security tile

- *Audit Logger*: to ensure integrity of the data for audit purpose.

C. Software Design: Dynamic Modeling

This section of the design phase will be adhered with UMLsec [7]. Sequence diagrams are used for dynamic modeling as they describe how and in which order a group of objects interact to achieve a goal. Fig. 6 shows a sequence diagram for the use case *Withdraw Funds* from the banking service with security components that employ UMLsec. To achieve this, the security packages are adopted to mitigate the identified potential vulnerabilities by the objects highlighted as green boxes. For example, to confirm requirement R1 for authentication and authorization in Security Domain Knowledge Table III, *message 1* sent from the ATMClient subsystem to the Withdrawal transaction manager business logic is validated using the mechanism related to*Validate Input* security package, the Withdrawal Transaction Manager is adopting the Validate Input security package. Similarly, to meet requirement R3 in Table III, log transaction message 1.4 is sent from the Withdrawal Transaction Manager business logic to the Transaction Log entity using the mechanism

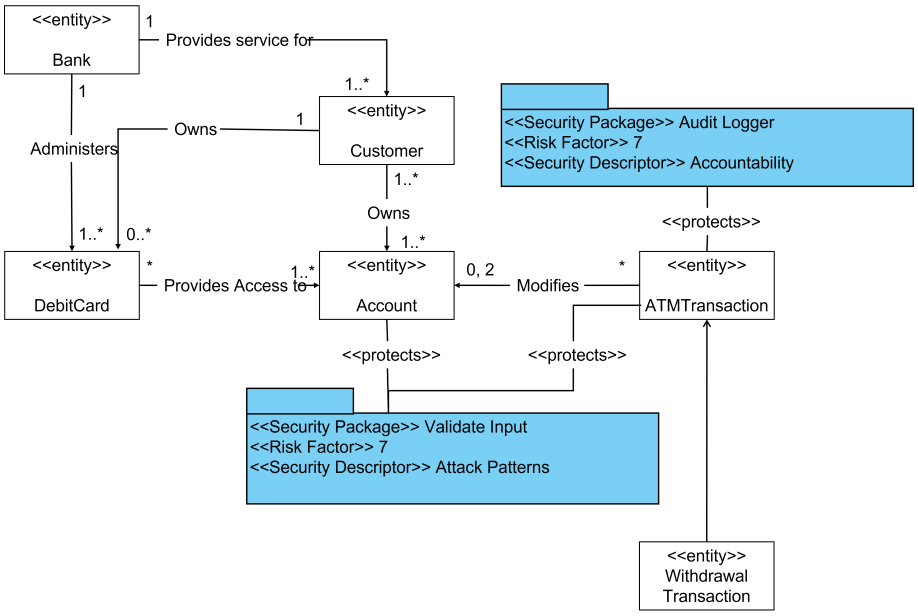

Fig. 5. Class diagram listing entities involved in *Withdraw Funds* with secure component

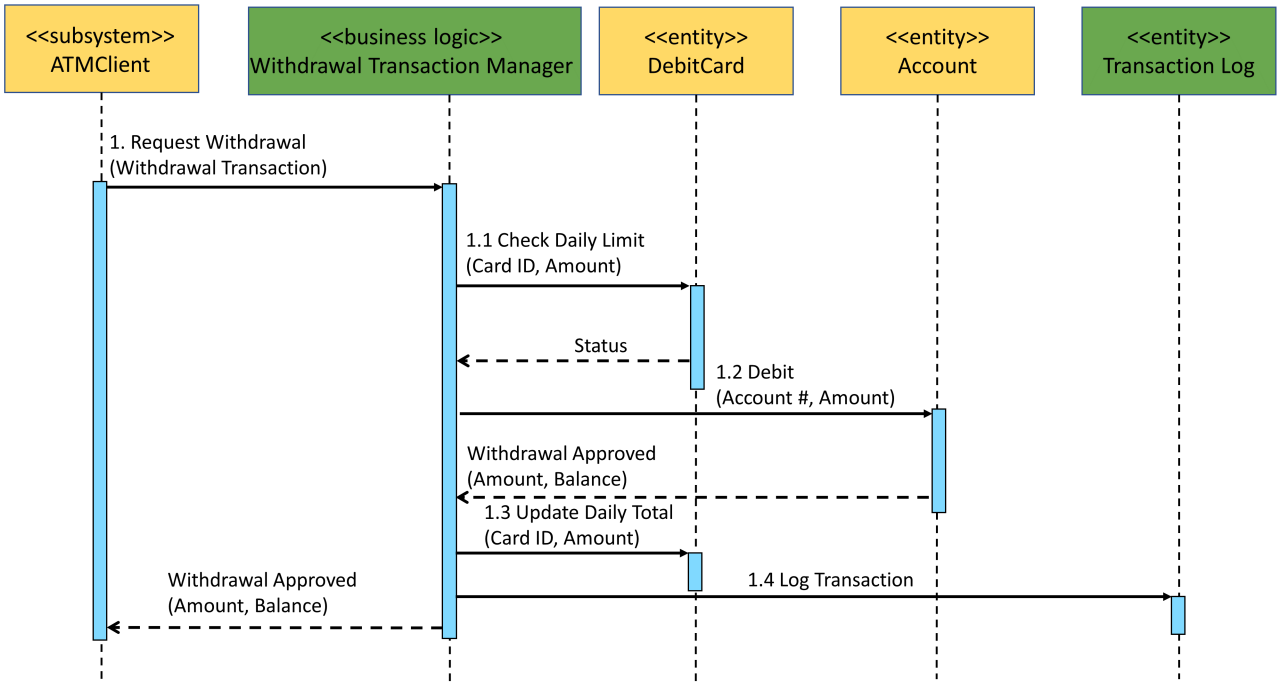

Fig. 6. Sequence Diagram: Banking service Withdraw Funds [17] with secure component

related to *Audit Logger* security package. Transaction Log is adopting attributes of the Logger security package to ensure auditability of transactions made on the system. In this way, the sequence diagram for *Withdrawal Funds* use case with UMLsec depicts the communication of various entities during the withdrawal of funds, which is combined with the security package required to mitigate potential intrusions/attackers to the ATM Banking System.

In the sequence diagram integrated with UMLsec, two distinct categories of entities are distinguished by color. The entities in orange color remain unchanged from the original sequence diagram. The green color entities are modified entities from the original sequence diagram, which means they retain their default configuration but have new methods and/or attributes added to behave as if a security package is embedded.

V. Conclusions and Future works

In this paper, we discussed a preliminary educational framework for teaching secure software development that could be integrated in a software engineering design course for undergraduate-level CS/CE curriculum to increase security awareness in students. The framework incorporates security in the SDLC by modifying the use case diagram, static, and dynamic design models for an ATM Banking system case study. This case study will be completed with the integration of security components into implementation and testing phases. The enhanced case study will be used for explaining various activities for secure software development in a software engineering course. Two or three more case studies will be added in the framework that will be given as projects or hands-on assignments to students for applying the knowledge gained from the ATM Banking case study. Students' evaluation will be performed for the usability of this framework for introducing secure software practices in software engineering design courses.

Acknowledgment

This project is based upon work supported by the National Science Foundation (NSF) grants #2021345 and #2021264. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] H. Shahriar and M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, pp. 1–46, 2012.

[2] B. Taylor and S. Kaza, "Security injections: modules to help students remember, understand, and apply secure coding techniques," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pp. 3–7, 2011.

[11] G. Georg, I. Ray, and R. France, "Using aspects to design a secure system," in *Eighth IEEE International Conference on Engineering of Complex Computer Systems, 2002. Proceedings.*, pp. 117–126, IEEE, 2002.

[3] H. Chi, E. L. Jones, and J. Brown, "Teaching secure coding practices to stem students," in *Proceedings of the 2013 on InfoSecCD'13: Information Security Curriculum Development Conference*, pp. 42–48, 2013.

[4] W. A. Conklin and G. Dietrich, "Secure software engineering: A new paradigm," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pp. 272–272, IEEE, 2007.

[5] G. Booch, I. Jacobson, J. Rumbaugh, *et al.*, "The unified modeling language," *Unix Review*, vol. 14, no. 13, p. 5, 1996.

[6] M. J. Peterson, J. B. Bowles, and C. M. Eastman, "Umlpac: an approach for integrating security into uml class design," in *Proceedings of the IEEE SoutheastCon 2006*, pp. 267–272, IEEE, 2006.

[7] J. Jürjens, "Umlsec: Extending uml for secure systems development," in *International Conference on The Unified Modeling Language*, pp. 412–425, Springer, 2002.

[8] D. Hatebur, M. Heisel, J. Jürjens, and H. Schmidt, "Systematic development of umlsec design models based on security requirements," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 232–246, Springer, 2011.

[9] T. Lodderstedt, D. Basin, and J. Doser, "Secureuml: A uml-based modeling language for model-driven security," in *International Conference on the Unified Modeling Language*, pp. 426–441, Springer, 2002.

[10] T. Doan, L. Michel, and S. Demurjian, "A formal framework for secure design and constraint checking in uml," in *Proceedings of the International Symposium on Secure Software Engineering (ISSSE'06)*, Citeseer, 2006.

[12] Y. B. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan, "Software development life cycle agile vs traditional approaches," in *International Conference on Information and Network Technology*, vol. 37, pp. 162–167, 2012.

[13] R. L. Krutz and R. D. Vines, *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing, 2010.

[14] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Proceedings 15th Annual Computer Security Applications Conference (ACSAC'99)*, pp. 55–64, IEEE, 1999.

[15] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, 2004.

[16] D. Xu and K. E. Nygard, "Threat-driven modeling and verification of secure software using aspect-oriented petri nets," *IEEE transactions on software engineering*, vol. 32, no. 4, pp. 265–278, 2006.

[17] H. Gomaa, *Software modeling and design: UML, use cases, patterns, and software architectures*. Cambridge University Press, 2011.

[18] B. Dobing and J. Parsons, "How uml is used," *Communications of the ACM*, vol. 49, no. 5, pp. 109–113, 2006.