Putting Theory into Practice: The Results of a Practical Implementation of the Secure Development Life Cycle

Cynthia Y. Lester
Department of Computer Science
Tuskegee University
Tuskegee, Alabama, USA
cylester@tuskegee.edu

***Abstract* – Software engineering is defined as a discipline concerned with all aspects of software production from inception to the evolution of a system. It has often been referred to as the "cradle-to-grave" approach to producing reliable, cost-efficient software delivered in a timely manner that satisfies the customer's needs. However, with the introduction of the Internet and the World Wide Web, software engineering has been required to make changes in the way that new software products are developed and protected. In order to protect systems from hackers and saboteurs in a global society where e-commerce, e-business, and e-sharing are the "norm", professionals should have sound knowledge in methods to protect data. Consequently, the area of information assurance (IA) has become one of great significance and it is important that the next generation of technologists are trained in development techniques that can ensure the confidentially and integrity of information. Traditionally, courses in secure software development are offered at the graduate level or in a stand-alone software security course at the undergraduate level. The aim of this paper is to present a framework for introducing software security to undergraduate students in a traditionally taught software engineering course. The paper focuses on and presents the results of a practical implementation of software security concepts learned through a service-learning project. The results from the study suggest that software security can be effectively introduced in a traditionally taught software engineering course through the implementation of a hands-on learning experience.**

Keywords – agile methods; secure software development service-learning; software development; software engineering; software security; traditional software development methodologies

I. INTRODUCTION

Securing information is not a new idea. In fact, securing data has its origins in World War II with the protection and safeguarding of data which resided on mainframes that were used to break codes [1]. However, during the early years, security was uncomplicated since the primary threats included physical theft of the system, espionage and sabotage against product resources [1]. Yet, it was not until the early 1970s that the concept of computer security was first studied. With the invention of the Advanced Research Projects Agency Network (ARPANET) by the U.S. Department of Defense in 1968 and its growing popularity in the early 1970s, the chance for misuse increased in what is now known to be the origin of the modern day Internet.

In 1990, it was reported that there were less that 50 million users of the Internet in the U.S. However, by 2008 the U.S. reported approximately 230,630,000 Internet users [2]. Therefore, it stands to reason that with more users and more advanced systems, the user population of today's technology would be more technically savvy than those user groups of yesteryear. However, the average user is now less likely to understand the systems of today as compared to the users of a decade ago. Further with the rapid pace at which new technologies are being introduced to the public, it becomes even more difficult for users to understand how to protect their systems and information from unwanted interruptions, threats and vulnerabilities.

In the *Report of the Presidential Commission on Critical Infrastructure Protection*, it was stated that "education on methods of reducing vulnerabilities and responding to attacks" and "programs for curriculum develop at the undergraduate and graduate levels" were recommended to reduce the number of vulnerabilities and malicious attacks on software systems [3]. Additionally, in the 2003 *National Strategy to Secure Cyberspace* four major actions and initiatives for awareness, education, and training were identified which included [4]:

- Foster adequate training and education programs to support the Nation's cybersecurity needs
- Promote a comprehensive national awareness program to empower all Americans -businesses, the general workforce, and the general population - to secure their own parts of cyberspace
- Promote private-sector support for well-coordinated, widely recognized professional cybersecurity certifications
- Increase the efficiency of existing federal cybersecurity training programs

Consequently, protecting data has become a topic of importance. In order to protect data from hackers and saboteurs in a global society where e-commerce, e-business, and e-sharing are the "norm", professionals should have sound knowledge in methods to protect data. Therefore, the

area of information assurance (IA) has become one of great significance.

Information assurance as defined in the CNSS Instruction Handbook No. 4009 are measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and nonrepudiation. Additionally, the measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities [5]. In order for students to gain training in information assurance, a series of courses are often taken, which include traditional computer science courses but also courses in information security, network security, computer security, cryptography, software security, etc. However, unless an institution has an information assurance track or program, students may not have the opportunity to gain exposure to many of these concepts, especially those concepts found in a software security course.

Therefore, the aim of this paper is to present a framework for introducing students to concepts of software security in a traditionally taught software engineering course. The paper begins by presenting several conventional software development methodologies discussed in a traditionally taught software engineering course which lends to an argument for a paradigm shift. Additionally, the paper presents a project in which students were engaged during the course of the sixteen week semester which focused on the practical implementation of software security concepts. The results of the project are discussed as well as challenges and future work.

II. TRADITIONAL SOFTWARE DEVELOPMENT METHODOLOGIES

Software engineering is defined as "being concerned with all aspects of the development and evolution of complex systems where software plays a major role. It is therefore concerned with hardware development, policy and process design and system deployment as well as software engineering [6]."

The term software engineering was first proposed at the 1968 NATO Software Engineering Conference held in Garmisch, Germany. The conference discussed the impending software crisis that was a result of the introduction of new computer hardware based on integrated circuits [6]. It was noted that with the introduction of this new hardware, computer systems were becoming more complex which dictated the need for more complex software systems. However, there was no formalized process to build these systems which put the computer industry at jeopardy because systems were often unreliable, difficult to maintain, costly, and inefficient [6]. Consequently, software engineering surfaced to combat the looming software crisis.

Since its inception, there have been many methodologies that have emerged that lead to the production of a software product. The most fundamental activities that are common among all software processes include [6]:

- *Software specification* – the functionality of the system and constraints imposed on system operations are identified and detailed
- *Software design and implementation* – the software is produced according to the specifications
- *Software validation* – the software is checked to ensure that it meets its specifications and provides the level of functionality as required by the user
- *Software evolution* – the software changes to meet the changing needs of the customer

Typically, students are introduced to these activities in the undergraduate computer science curriculum through a software engineering course. This course is sometimes a survey course which exposes students to a variety of life cycle models used in industry. The course is often taught from a systems approach which places an emphasis on creating requirements and then developing a system to meet the requirements. In the traditional view of software development, requirements are seen as the contract between the organization developing the system and the organization needing the system [7].

A traditional view of software development is the waterfall method. The waterfall method was the first published software development process and forms the basis for many life cycles. It was noted as a great step forward in software development [8]. The method has stages that cascade from one to the other, giving it the "waterfall" name. Figure 1 is an example of the waterfall life cycle [9].

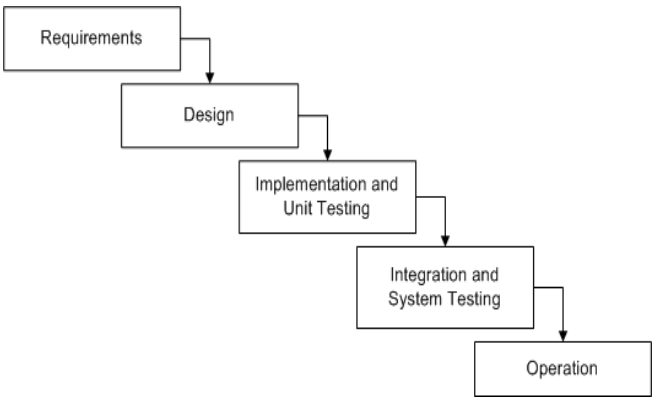

Figure 1. Waterfall model

It has been noted that the method might work satisfactorily if design requirements could be addressed prior to design creation and if the design were perfect prior to implementation [8]. Consequently, one of the main disadvantages of this model is that requirements may change accordingly to meet the needs of the customer and

the change is difficult to incorporate into the life cycle. As a result of this shortcoming, additional life cycles emerged which allowed for a more iterative approach to development.

Evolutionary development is based on the idea of developing an initial implementation and then exposing the build to the user for comment and refinement [6]. Figure 2 is an example of the evolutionary development method [6].

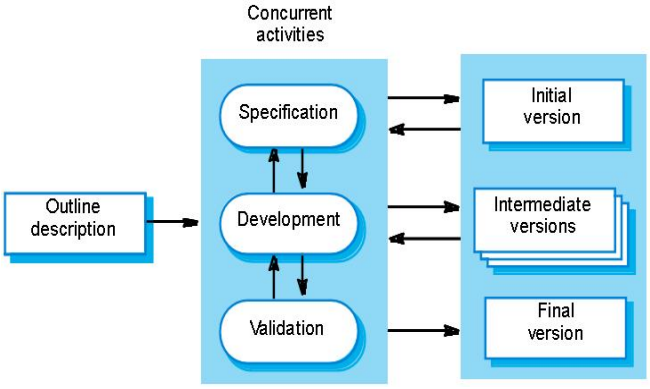

Figure 2. Evolutionary development

There are two fundamental types of evolutionary development:

- *Exploratory development* – developers work with customers to discern requirements and then the final system is delivered
- *Throwaway prototyping* – used to quickly development a concept and influence the design of the system

The advantage of evolutionary development is that it is developing specifications incrementally [6]. As customers have an opportunity to interact with the prototype, specifications are refined which leads to a better, more useful, usable, and used software. However, while this approach is somewhat better than the waterfall model, it is not without its criticisms. Sommerville notes that the process is not visible and that the systems being developed are often poorly structured [6]. The next model presented is stated to be an improvement over both the waterfall and evolutionary development models.

The spiral development model is an example of an iterative process model that represents the software process as a set of interleaved activities that allows activities to be evaluated repeatedly. The model was presented by Barry Boehm in his 1988 paper entitled *A Spiral Model of Software Development and Enhancement* [10]. The spiral model is shown in figure 3. The spiral model differs from the waterfall model in one very distinct way because it promotes prototyping; and, it differs from the waterfall and evolutionary development method because it takes into consideration that something may go wrong which is exercised through risk analysis.

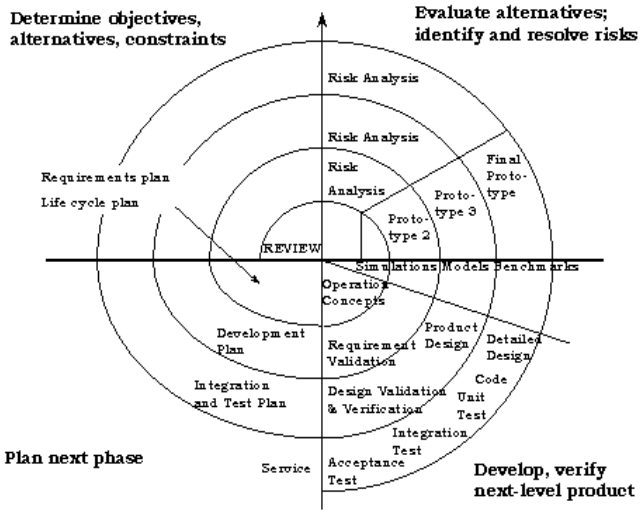

Figure 3. Spiral model

It is noted that this life cycle provides more flexibility than its more traditional predecessors. Further, this method produces a preliminary design. This phase of the life cycle was added specifically in order to identify and resolve all the possible risks in the project development. Therefore, if risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed in order to determine a possible solution.

The activities that formulate this view of software engineering came from a community that was responsible for developing large software systems that had a long life span. Moreover, the teams that used these methodologies were typically large teams with members sometimes geographically separated and working on software projects for long periods of time [7]. Therefore, software development methodologies that resulted from this view of software engineering were often termed as "heavyweight" processes because they were plan-driven and involved overhead that dominated the software process [11]. However, great difficulty occurs when these methodologies are applied to smaller-sized businesses and their systems, because these methods lack the agility needed to meet the changing needs of the user. The next section presents an overview of an emerging process methodology which is an alternative to heavyweight processes, agile development.

III. AGILE METHODS

In an effort to address the dissatisfaction that the heavyweight approaches to software engineering brought to small and medium-sized businesses and their system

development, in the 1990s a new approach was introduced termed, "agile methods." Agile processes are stated to be a family of software *development methodologies in which* software is produced in short releases and iterations, allowing for greater change to occur during the design [11]. A typical iteration or sprint is anywhere from two to four weeks, but can vary. The agile methods allow for software development teams to focus on the software rather than the design and documentation [11]. The following list is stated to depict agile methods [11], [12]:

- *Incremental design* - the design is not completed initially, but is improved upon when more knowledge is acquired throughout the process
- *User involvement* - there is a high level of involvement with the user who provides continuous feedback
- *Short releases and iterations* - allow the work to be divided, thereby releasing the software to the customer as soon as possible and as often as possible
- *Informal communication* - communication is maintained but not through formal documents
- *Minimal documentation* – source code is well documented and well-structured
- *Change* - presume that the system will evolve and find a way to work with changing requirements and environments

More specifically, the agile manifesto states:

"We are uncovering better ways of developing software by doing it and helping others to do it.
Through this work we have come to value:
Individuals and interaction over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan
That is, while there is value in the items on the right, we value the items on the left more."

It is stated that agile processes have some very precise advantages over its heavyweight predecessors which include the following as stated by Tsui and Karam [12]:

- *Low process complexity* - processes are simple which promotes easier implementation and understanding
- *Low cost and overhead* - processes require only a small number of activities that do not directly lead to the production of software
- *Efficient handling of changes* - processes are designed and developed with the presumption that requirements will change and the methodology is prepared to incorporate those changes
- *Quick results* - processes have low overhead which results in a final product being produced quicker than with traditional heavyweight processes. Also, agile processes are designed for continuous integration which allows for constant improvement and the implementation of additional functionality as the project progresses.
- *Usable systems* - the customer is involved and therefore when changes occur, the process can quickly adapt, yielding a product that the customer really wants and wants to use

However, agile methods are not without their critics. Just as the traditional methods have disadvantages, agile methods do as well. According to researchers, listed below are the main disadvantages of agile processes [11], [12]*:*

- *May not be scalable* - agile processes are typically used by small teams and may have problems scaling to adjust to larger systems without losing their agility
- *Heavy reliance on teamwork* - the processes are generally used by small teams who are centrally located and who depend on informal communication to accomplish a task; team work can be destroyed if cross-team communication mechanisms have not been designed and used
- *Reliance on frequent customer access* - it has been stated that it is sometimes difficult especially after software delivery to keep the customer involved in the process; consequently without customer involvement, agile methods may not be able to properly validate requirements or adjust to change
- *Cultural clash* – Extreme programming (XP) is probably one of the best known and most widely used agile methods [13], [14]. It was originally designed to address the needs of software development by small teams who faced changing requirements and system environments. However, XP often clashes with the more commonly accepted software engineering ideas and management techniques. Therefore, the use of agile methods by development teams may make it difficult to conduct performance evaluations and team member progress reviews.

However, just as with traditional software methodologies, agile methods do not often address software security. Moreover, when these approaches to software development are taught in traditional software engineering courses, security is mostly absent from the instruction. Hence, the increasingly important need to include a discussion of software security in the software development process taught to undergraduate students. The next section explores secure software development and its life cycle.

IV. CHARACTERISTICS OF SECURE INFORMATION

The *Morris Worm* was the first known network security breach to impact thousands of computers that were connected to (ARPANET) [15], [16]. It was reported that Robert Morris, a graduate student at Cornell University, wrote a program that exploited bugs that he noticed in several UNIX applications [16]. The basic premise of the

program was that it connected itself to another computer on the network, copied itself to the new location, and executed both the original version and the copy. This action would then be repeated in an infinite loop to other computers, thereby causing thousands of computers to become infected. He became the first person to receive a felony conviction in which he was sentenced to serve 3 years probation, 400 hours of community service, and pay a fine of $10,000 [16].

Since it began operating in 1998, the Computer Emergency Response Team (CERT) Coordination Center has tracked and reported the number of security vulnerabilities [15]. A snapshot of early security vulnerabilities is depicted in Figure 4 [15]. The chart shows the growth in security incidents from the first incident in 1998 until 1995, which was the last for which statistics were available according to the reference.

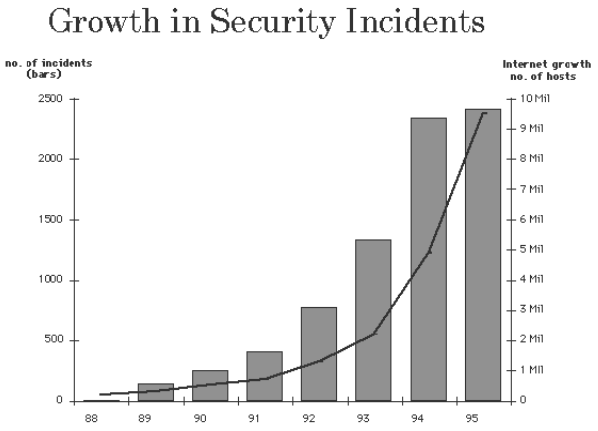

Figure 4. Vulnerabilities Report

More recently, according to statistics published by the Computer Emergency Response Team (CERT), between 1995 and 2008 approximately 44,074 vulnerabilities had been cataloged [15]. It has been reported that these software vulnerabilities and software errors cost the U.S. approximately $59.5 billion annually [17].

Software errors have grown in complexity. In 2000, NIST reported that the total sales of software reached approximately $180 billion and the software was supported by a workforce that consisted of 679,000 software engineers and 585,000 computer programmers [17]. Some of the reasons that software errors have grown in complexity are that typically, software now contains millions of lines of code, instead of thousands; the average product life expectancy has decreased requiring the workforce to meet new demands; there is limited liability among software vendors; and, there is difficulty in defining and measuring software quality [17].

Consequently, it is imperative that students in computer science and information technology be trained in the concepts of security and how to design and develop secure software so that they can contribute viably to the fast changing technological demands of this global society. The traditional development strategies expose students to the methods for software development, but as they consider how to guard against hackers, how to protect critical information, and how to lessen security threats, a question of what is "good" information arises. Therefore, before students can understand and have an appreciation for the secure software development life cycle, they must first be exposed to the qualities and characteristics of "good" information.

The value of information has been stated to come from the characteristics that it possesses [1]. While some characteristics may increase the value of the information as it relates to use by users, other characteristics may have a more significant value among security professionals. However, all characteristics as defined below are critical as it relates to secure information [1].

- *Availability* - allows users who need to access information to access the information without impediment or intrusion. Further, availability means that users can receive information in the desired format.
- *Accuracy* - as defined by *The American Heritage College Dictionary* is conformity to fact; precision; exactness [18]. As accuracy relates to secure software it means that the software has the value that the user expects and that it is also free from errors.
- *Authenticity* - is the state or quality of information being original or genuine. The information should not be a replication of other information. Whitman further reveals that information is authentic when it is the information that was originally created, placed, stored or transferred.
- *Confidentiality* - only those persons with "certain" rights can have access to the information. It means that only authorized persons or systems can gain access to the information.
- *Integrity* - is adherence to a strict code or the state of being unimpaired [1]. As it relates to the integrity of information it is the state of being uncorrupted or the state of being whole.
- *Utility* – the condition of being useful. If the information being provided is not useful or presented in a format that cannot be used, then the information loses its value or its quality of being "good" information.
- *Possession* - the condition of being owned or controlled. Whitman and Mattford note that while a breach in confidentiality always results in a breach of possession, the opposite may not be true [1].

V. THE THEORETICAL APPROACH TO THE SECURE DEVELOPMENT LIFECYCLE

There are many approaches to the development of robust software that can ensure that the information being used by users is available, accurate, authentic, possesses confidentially, integrity, is useful and can be controlled. However, the question becomes how to introduce this model at the undergraduate level when a specialized course in software security is not available or when typically students only take one course in software development. The following section presents the secure software development life cycle taught in a traditionally taught software engineering course and introduces a method which allowed students to gain practical experience in implementing security concepts.

A misconception among students as well as with computing professionals is that security should be thought of in the later phases of the software development life cycle. However, if systems are to withstand malicious attack, a robust software development model or a secure software development must be used. One viewpoint of the secure life cycle discussed in class was developed by Apvrille and Purzandi and a modified version is presented in Figure 5 [19].

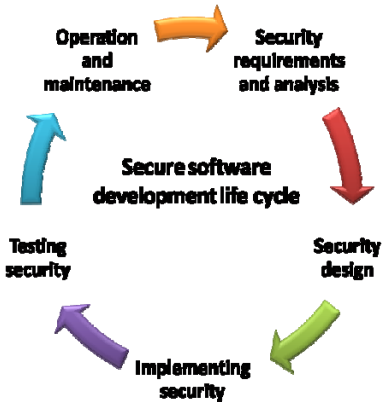

Figure 5. Secure life cycle

A. Security requirements and analysis

While requirements are being gathered from users and stakeholders, focus should also be placed on establishing a security policy. In order to develop a security policy, attention needs to be given to what needs to be protected, from whom, and for how long [20]. Additionally, thought needs to be placed on the cost of protecting the information. The result of this phase should be a set of guidelines that create a framework for security [1].

B. Security design

During the design phase it has been stated that the security technology needed to support the framework outlined in the requirements phase is evaluated, alternative solutions are explored, and a final design is agreed upon [1]. It is recommended by Viega and McGraw that the following be the focus of this phase [20]:

- How data flows between components
- Users, roles and rights that are explicitly stated or implicitly included
- The trust relationships between components
- Solutions that can be applied to any recognized problem

At the end of this phase a design should be finalized and presented. The design should be one that can be implemented.

C. Implementation

The implementation phase in the secure development life cycle is similar to that which is found in traditional methodologies. However, when implementing a software project with security in mind, it is important to consider a language or a set of languages that may have security features embedded, one that is reliable when it comes to denial-of-service attacks, and that can perform error checking statically, etc. Further, it is important to understand the weaknesses of languages, for example buffer overflows in C and C++.

D. Testing

Testing in the secure development life cycle is different than in traditional methodologies. In traditional methodologies, testing is done to ascertain the behavior of the system and to determine if the system meets the specifications. Security testing is used to determine if a system protects data and maintains functionality as intended. As mentioned previously the six concepts that need to be covered by security testing are availability, accuracy, authenticity, confidentiality, integrity, utility, and possession. It has been stated that security testing is most effective when system risks are uncovered during analysis, and more specifically during architectural-level risk analysis [20].

E. Maintenance

It has been stated that the maintenance and change phase may be the most important phase of the secure development life cycle given the high level of cleverness seen in today's threat [1]. In order to keep up with the changing threats to systems, security systems need constant updating, modifying, and testing. Constant maintenance and change ensure that systems are ready to handle and defend against threats.

VI. THE PRACTICAL APPROACH TO THE SECURE DEVELOPMENT LIFE CYCLE

A. Course Description

The course chosen for the practical implementation of the secure development life cycle was the traditionally taught *CSCI 430 – Software Engineering* course under the instruction of the author. A brief description of the course is to provide students with an engineering approach to software development and design; and, to expose students to current research topics within the field [21]. The software engineering course was modified to reinforce the need to think about security features and requirements early in the development process so that security protection mechanisms are designed and built into the system rather than added on at a later time.

The prerequisites for the course are to have successfully completed *CSCI 230 - Data Structures* and *CSCI 300 - Discrete Mathematical Structures* with a grade of C or better.

B. Course Learning Outcomes

Learning outcomes are extremely important when developing a course. The learning outcomes describe the specific knowledge and skills that students are expected to acquire. The learning outcomes for the CSCI 430 course include the following: at the end of the course, a student should be able to:

- Describe in detail the software process
- Identify various software process models and determine which model should be used for a specific project
- Implement each phase of the software process
- Work effectively and efficiently in a team environment to produce a large scale project
- Identify and discuss current research topics related to the software engineering discipline

It was the anticipation of the author that through the hands-on experience of developing a project that included security concepts, students would gain an understanding of the importance of secure software engineering and their approach to development would be enhanced. Further, as students use and understood the concepts presented during class, conceptually they would be able to apply the principles to a semester long project. It was decided to use the concepts found in service-learning to design the project. The next section provides a high level overview of service-learning.

C. Service Learning

Service-learning is defined as a method of teaching through which students apply their academic skills and knowledge to address real-life needs in their own communities [22]. Service-learning provides a compelling reason for students to learn; it teaches the skills of civic participation and develops an ethic of service and civic responsibility. By solving real problems and addressing real needs, students learn to apply classroom learning to real world situations [22]. Service-learning has been shown to be an educational technique that facilitates a student's growth in academics, communication, social maturity, critical thinking, collaboration, and leadership skills [22]. Students who are involved in meaningful service-learning have further been shown to perform better on tests, show a sense of self-esteem and purpose, connect with the community, and want to be more civically engaged than students who do not participate in service-learning activities [22].

There are many key components that are encompassed within service-learning. The author has chosen some of those activities that were included in CSCI 430 and, they are presented in the next sections.

1) Reflection. Reflection fosters the development of critical thinking in students. Reflection and critical thinking (problem-solving) are essential tools that will help students be successful in school, career, and life. Service-learning reflection includes the following activities by the student:

- Assessing personal interests, knowledge, skills, and attributes that will be useful in performing the service-learning project.
- Thinking about how to take effective steps to meet the identified needs.
- Self-evaluating one's progress toward meeting the goals of the project.

2) Working as a team. The students learn to work for a common goal and by doing so acquire a variety of skills, such as how to lead, how to be accountable, how to communicate ideas, how to listen to others, and how to set a goal and work effectively as a team to reach the goal.

3) Experiential learning. Service-learning uses direct experience and hands-on learning to help the student learn to take the initiative, assume responsibility, and develop effective problem-solving skills.

The next section describes the course project that was designed based on the concepts found in service-learning and a modified version of the secure software development life cycle.

VII. THE PROJECT

A. Project Statement

The semester long project selected for the fall 2009 semester was to develop an electronic voting/tallying system for the hotly contested position of the University's Queen. During past years, there have been errors in the selection process of the University's Queen; which has resulted in a process where contestants and the student body have little confidence. Students were required to develop a software product that meets the needs of the customer and helps to refine the election process and ballot-counting process for the University's Queen contest. Students were

part of a team which was expected to meet with the customer (or representative) so that each phase of the process could be implemented. The team was also expected to produce a deliverable by the set deadline for each phase of the process and to also deliver it and make presentations to the customer (or representative).

B. Project Learning Outcomes

The learning outcomes of the semester long project included that after the completion of the project students would:

- Have a working knowledge of the secure software development life cycle
- Understand and have a working knowledge of secure software engineering principles
- Be able to describe software vulnerabilities
- Develop and execute security measures
- Work effectively and efficiently in a team environment to produce the semester long project

C. Project Requirements

Students were given basic requirements from the instructor for the software application; however, the majority of the requirements were gathered from stakeholders. Since the project was infused with software security concepts there were both standard project requirements as well as security requirements.

D. Project Deliverables

Each item that the student team submitted was considered a deliverable. The project had four deliverables which were the requirements document, design document, implementation, and the test plan. The following is an overview of the project deliverables which were previously presented in work by Lester [23], [24].

1) Requirements Document. The first document students were required to submit was the requirements document. The requirements document was considered the official statement of what the students would implement. It included both the stakeholder requirements for the software application, which students named the *MISS System*, and a detailed specification of system requirements. To gather the requirements students met with stakeholders who included Administrators in the Office of Student Life, contestants from past elections, and student body leaders who were in charge of election results. The initial document was meant to get the students active in the planning and development of the system. After completion of the requirements document, students had an idea of the way they wanted the system to look, how the system would be accessed, and by whom (i.e., password authentication, access control).

2) Design Document. The team was required to use one of the decomposition styles discussed in the course. The design document was required to have an introduction, an overview of the design strategy chosen, and the diagrams, charts, and/or details required as part of the decomposition strategy chosen. The design document was also meant to be an in-depth description of the system design. The design showed how data flowed between system components and the trust relationships between components. Both the system and security requirements were described and explained how they would be implemented. Further the document identified vulnerabilities to the system and possible solutions were presented.

3) Implementation. Students were required to implement the project based on the requirements and design documents. To implement the project students chose the Java programming language.

4) Testing. Students were required to develop a test plan which required them to perform requirements-based testing and structural testing (inclusive of security testing).

Table 1. provides the timeframe for project deliverables.

TABLE 1. PROJECT DELIVERABLE TIME TABLE

Deliverable	**Deadline**
Requirements document	Week 8
Design document	Week 12
Implementation	Week 16
Test Plan	Week 16

VIII. RESULTS AND DISCUSSION

In order to determine the effectiveness of the service-learning project, the following actions were taken:

- For each deliverable a grade was determined based on the submitted document and the oral presentation of the document.
- After the completion of the each phase of the project, an exit interview with team members was conducted.

This section presents an overview of the results of these activities.

1) Requirements Document. The requirements documents was required to have the following sections as outlined in the textbook for the course by Sommerville [6]:

- Introduction
- User definition
- System architecture
- System models
- System evolution

Additionally, the document was graded on organization, grammar and style.

Results revealed that students had a good understanding of the user definition and the system architecture. However, system models proved to be a difficult topic for students to master. Yet, the overall quality of the document showed that students engaged in high-level critical thinking and problem solving, which was one of the goals of the service-learning project.

2) Design Document. To implement the design, students were required to choose one of the decomposition strategies discussed in class. Students chose to use object-oriented decomposition. Therefore, the parts of the document were required to include the following:

- Class diagrams
- Static diagrams (collaboration or sequence)
- Dynamic diagrams (activity or state)
- Security plan and evaluation

Results revealed that students had a good understanding of class and static diagrams, but had some difficulty with dynamic diagrams. All students had previously taken a theory course in which activity and state diagrams had been discussed, but students still struggled with the implementation of these diagrams as it related to the service-learning project.

A review of the design document also revealed that while the students gave heavy consideration and thought to security, the plan was limited in scope. The security plan addressed the characteristics of secure information, but students had difficulty with the design of the plan and how the plan would be evaluated.

3) Implementation. The requirement for this phase of the life cycle was an executable software application that met the requirements. To implement the *MISS System* students chose the Java programming language. The results from this phase of the project were mixed.

One of the challenges that students faced was the time constraint. The project was to be completed during the course of a sixteen week semester. Students naturally thought that because they had been previously engaged in semester-long projects in other courses that this project would be similar and that there would be enough time to complete all phases of the life cycle, especially the implementation phase. However, students quickly realized that this conjecture was incorrect as the end of the semester quickly approached. Further, unforeseen challenges such as changing requirements and teaming issues caused implementation delays; consequently, impacting the implementation of several requirements.

Students also had difficulty with the porting of the software application from the platform on which it was developed and the platform on which the application was to execute. The host platform was controlled by the Department of Computer Science. It was one of which the students had extensive knowledge because it was the platform on which they used for development and implementation of projects for other classes. However, the target platform on which the application was to execute was controlled by Campus Technology. It was completely different and one with which the students were quite unfamiliar. Therefore, the porting of the software application proved to the most difficult part of the project as host-target development was not considered during the requirements phase of the development life cycle.

4) Test Plan. The requirement for this phase was a test plan that included test case design. The test plan was based on Sommerville's structure of a software test plan for large and complex systems but modified to be less formal and represent the smaller nature of the *MISS System* [6]. The modified version of the software test plan included the following components:

- The testing process
- Test case design
- Hardware and software requirements
- Constraints

Test case design was an integral part of the software test plan. Test case design can be described as the process in which the system is executed in a controlled environment using established inputs into the system. The goal of the process is to create test cases that can discover defects and errors with the system and to also show that the implemented system meets the requirements of the stakeholders. The next section describes requirements-based testing and structural testing, which were used as part of the testing process.

Requirements should be designed so that they can be tested. Therefore, requirements-based testing is used to ensure that individual requirements are tested and to also provide a level of confidence to the stakeholders that their needs were being met. To test the requirements of the *MISS System* students developed and completed the following simple table as shown in Table 2.

TABLE 2. REQUIREMENT TESTING

Requirement	Test Case	Outcome

Structural testing is an approach that is used to test the system based on developer's knowledge of the structure and implementation of the software. This type of testing is typically used throughout a computer science curriculum as students who are learning to program also develop test cases based on the structure of their programs. By having knowledge of the code, student-developers can design test cases that can potentially uncover errors or problems. However, structural testing is not designed to detect missing or unimplemented requirements. Table 3 is an example of a simple table that was developed and students were asked to complete to the meet the objectives of structural testing.

TABLE 3. STRUCTURAL TESTING

Code	Test Case (Input)	Outcome

Results from testing revealed that this part of the life cycle was also quite challenging for the students. Students had some difficulty in determining test cases to test the requirements. Further, since some requirements were not

implemented, they could not be tested. Results also revealed that structural testing was a little easier for the students as this concept is one with which they are familiar because as previously stated, a modified version of structural testing is taught throughout the curriculum.

IX. CONCLUSION

In conclusion, the aim of this paper was to present a theoretical and practical framework for introducing to undergraduate students the secure software development process. The paper presents the results of a practical implementation of software security concepts learned through a service-learning project.

The author acknowledges that while there are many development methodologies that exist to train students in software security, many consist of steps that cannot be implemented in a one-semester course, especially with undergraduate students. Further, the author found that it was quite difficult for students to complete the secure development life cycle and develop a "truly" secure system, because it was costly as it related to resources (i.e., time, platform and personnel). This finding is consistent with the research perspectives of Devanbu and Stubbline [25].

Future work activities include that the author plans to revise the project, the deliverables and the timeframe for the deliverables. Additionally, the author plans to review the life cycle chosen for the project and will create a modified version of a life cycle for students to implement. The exit interviews revealed that students wanted less time for requirements/design and more time for implementation and testing.

As software becomes more complex and vulnerabilities and threats to these systems become just as complex, it is important to introduce to the next generation of technologists ways that systems can be made more secure. As educators it becomes our responsibility to train these students so that developing secure software is not just introduced in theory, but in practice as well.

ACKNOWLEDGMENTS

The author wishes to thank the students enrolled in the fall 2009 CSCI 430 – Software Engineering class for their hard work, the Tuskegee University Office of Student Life for serving as customers for the project and the Tuskegee University Office of Campus Technology for their assistance on the project.

REFERENCES

[1] M.E. Whitman and H.J. Mattford. Principles of Information Security. Boston: Course Technology. 2004.

[2] Internet users as percentage population. http://www.geohive.com/charts/ec_internet1.aspx (Accessed December 20, 2010).

[3] J. Elli, D. Fisher, T. Longstaff, L. Pesante, and R. Pethia. "A Report to the President's Commission on Critical Infrastructure Protection." [Electronic Version] http:// www.cert.org/pres_comm/cert.rpcci.ex.sum.html#edu (Accessed on April 1, 2008).

[4] The National Strategy to Secure Cyberspace. (2003). [Electronic Version]. http://www.uscert.gov/ reading_room/cyberspace_strategy.pdf (Accessed on June 13, 2011).

[5] http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf. (Accessed January 17, 2008).

[6] I. Sommerville. (2007). *Software Engineering 8th Ed*. Addison Wesley, 978-0-321-31379-9, Boston, MA.

[7] C. Angelov, R.V.N. Melnik, & J. Buur. (2003). The synergistic integration of mathematics, software engineering, and user-centered design: exploring new trends in education. *Future Generation Computer Systems*. Vol. 19, 299 – 1307.

[8] B. K. Jayaswal and P.C. Patton (2007). Design for trustworthy software: Tools, techniques for developing robust software. Prentice Hall, 0-13-187250-8, Upper Saddle Rover, NJ.

[9] Codebetter.com http://codebetter.com/blogs/raymond. lewallen/downloads/waterfalllModel.gif. (Accessed on October 10, 2009.

[10] B. Boehm. (1988). A Spiral Model of Software Development and Enhancement. *IEEE Computer 21*, 5, 61-72.

[11] I. Sommerville. (2011). *Software Engineering 9th Ed*. Addison Wesley, 978-0-13-703515-1, Boston, MA.

[12] F. Tsui and O. Karam. (2011). *Essentials of Software Engineering 2nd Ed*. Jones and Bartlett Publishers, 13:978-0-7637-8634-5.

[13] K. Beck (1999). Extreme programming explained: Embrace the change. Addison Wesley.

[14] R. Jefferies, A. Anderson, C. Hendrickson. (2000). Extreme programming installed. In: The XP Series. Addison Wesley.

[15] CERT Coordination Center, CERT/CC. [Electronic Version]. http://www.cert.org/ (Accessed July 12, 2011).

[16] M. Quin. Ethics for the Information Age 4th Ed. Boston: Pearson Education. 2011.

[17] Software Errors Cost U.S. Economy $59.5 Billion Annually: NIST Assesses Technical Needs of Industry to Improve Software-Testing [Electronic Version] http://www.nist.gov/public_affairs/releases/n02-10.htm (Accessed on April 1, 2008).

[18] Accuracy; Integrity. American Heritage College Dictionary. (1993). New York: Houghton Mifflin Company.

[19] A. Apvrille and M. Purzandi. "Secure Software Development by Example," IEEE Security & Privacy, vol. 3, no. 4, July/August, 2005. p. 10 – 17.

[20] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way.* Boston: Addison-Wesley. 2001.

[21] C. Lester. (2009). *CSCI 430 – Software Engineering Syllabus*.

[22] K. McPherson. Service Learning. New Horizons for Learning. http://www.newhorizons.org/strategies/service_learning/front_service.htm. 2005.

[23] C. Lester. (2010) "Shifting the Paradigm: Training undergraduate students in software security." Proceedings of the Fourth International Conference on Emerging Security Information, Systems and Technologies. Venice, Italy, July 18 – 25, 2011.

[24] C. Lester. (2010). "A practical application of software security in an undergraduate software engineering course." International Journal of Computer Science Issues, Vol. 7, Issue 1.

[25] P.T. Devanbu and S. Stubbline. "Software engineering for security: a roadmap," Proceedings of the Conference on the Future of Software Engineering. Limerick Ireland, June 4 – 11, 2000.