ESIDE: THE INTEGRATION OF SECURE PROGRAMMING EDUCATION INTO THE IDE

by

Michael P. Whitney

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2015

Approved by:

Dr. Heather Lipford

Dr. Celine Latulipe

Dr. Bei-Tseng Chu

Dr. Mohamed Shehab

Dr. Richard Lambert

ABSTRACT

MICHAEL P. WHITNEY. ESIDE: The integration of secure programming education into the IDE. (Under the direction of DR. HEATHER LIPFORD)

Securing the world's technological resources has become one of the largest challenges of modern day. From healthcare to national security, power grids to public safety, the world is reliant on computer systems and their ability to perform in a safe and secure manner. To this end, higher education must graduate software developers who comprehend the importance of security and take steps to ensure the safety of our technological systems. Unfortunately, academia's efforts are falling short as the world continues to experience a shortage of these individuals.

In this dissertation, I present a novel educational approach to improve on academia's current methods of secure programming instruction. Known as ESIDE: Educational Security in the IDE, it complements current methods of instruction (e.g., modified courses, elective courses, security tracks) by infusing instructional guidance and materials in a contextually based real-time manner into the student's IDE in a method similar to Microsofts Grammar Check (a.k.a., green squiggly). The effect of which is an exponential increase in exposure to the principles and practices of secure coding.

I designed the ESIDE to provide an interactive educational experience for all levels of programming students across the curriculum. Currently, it runs as a plugin for the Eclipse IDE and monitors the student's code writing process for potentially vulnerable code patterns. When vulnerable code is discovered, ESIDE initiates an educational process based on the type of vulnerability discovered. I have evaluated this model in

formative and summative studies at multiple institutions and educational levels and have received promising results. I plan to continue implementing lessons learned and knowledge gained into future research so that I might create a more robust educational resource.

ACKNOWLEDGMENTS

As I reflect back on my work I feel it is more than appropriate to acknowledge those that helped me along the way. I would like to start by saying thanking Dr. Chu as he was the first person I met at UNCC. I am thankful for his support throughout the years and honored to have had him on my dissertation committee. I am also in his dept for his support during my job seeking process. I am sure his efforts helped me land those interviews and secure a position in higher education.

Of utmost importance is my need to express my gratitude for everything that Dr. Heather Richter Lipford has done for me. From taking me on as one of her students, to guiding me through my education, to serving as the chair of my committee, to introducing me to the most wonderful person in the world, she has been a truly amazing person that continues to be awesome. Thank you for everything Dr. Heather Richter Lipford!

I would also like to thank the remainder of my committee members which consisted of Dr. Celine Latulipe, Dr. Mohamed Shehab and Dr. Richard Lambert. I appreciate the insight and support that each provided not only during my dissertation but also throughout my academic career. I feel myself very fortunate to have worked with such a talented group.

Throughout the years I have been able to collaborate with an amazing group of individuals. Berto Gonzalez, Vikash Singh, Tyler Thomas, Jun Zhu, Dr. Jing Xie, Alex Adams, Dr. Okan Pala, Mingming Fan, Stephen MacNeil, Carlos Seminario, Jill Morgan, Dora Bradley, Kimberly Lord, Katie Froiland, Erik Northrop, Jinyue Xia,

Dr. Jason Watson, and Dr. Andrew Bessemer have all been been there for me in one form or another. Thank you to everyone for sharing this experience with me.

I would like to acknowledge my funding sources that allowed me to pursue my degree. My first source came in the form of a graduate assistant under the guidance of UNC Charlotte's CIO Jay Dominick. Thank you Jay. I would also like to thank UNC Charlotte's GASP (Graduate Assistant Support Plan). Importantly, I am thankful for the support that I received from the NSF-DUE #1044745 grant as those funds directly supported my dissertation work. All of these funds helped keep my focus on my work rather than financial strains.

Lastly, and most importantly, I would like to thank Shannon Kirby. I would not be where I am today without her. Thank you Shannon Kirby!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Securing the world's technological resources has become one of the largest challenges of modern day. From healthcare to national security, power grids to public safety, the world is reliant on computer systems and their ability to perform in a safe and secure manner. For this purpose, our world's software developers must comprehend the importance of security and take steps to ensure the safety of our technological systems. Unfortunately, the world is experiencing a shortage of these professionals [22].

Those professionals who can secure our world's technological resources apply defensive practices in one of two dominant categories - hardware and software (each has multiple subcategories). Hardware security refers to the defensive practices put in place to protect physical devices such as server mainframes and storage devices. Software security refers to the efforts taken to secure the data on the hardware as controlled by programs and applications (e.g., bank account information).

A vital subcategory of software security is the practice secure programming, the purpose of which is to create secure software through the implementation of secure coding techniques. Doing so fortifies the software against attacks (e.g., stolen passwords). Unfortunately, it is evident that many professionals do not possess secure coding skills as most of today's software vulnerabilities are caused by insecure

code [30, 71]. If our technological world is to remain safe and secure, it is imperative that academia addresses this issue and teach our future professionals these much needed skills.

Academia's response has been to incorporate security instruction into its curriculums in both classic and innovative ways. It has modified courses, created classes, developed programs, implemented security tracks, and integrated thread-based approaches [7, 15, 63, 70, 73, 74] in an attempt to deepen the professional pool. Each of these approaches has their own advantages and disadvantages but when examined collectively, they fall short in exposing all students to the breadth and depth of the much-needed instruction [52]. This is unacceptable as "the ability to write secure code should be as fundamental to a university computer science undergraduate as basic literacy" [3, p. 56].

While restructuring current practices is not probable, it is possible to complement academia's approach to secure code instruction with my developed Eclipse plugin known as ESIDE (Educational Support in the IDE). This plugin provides instructional guidance and materials in a contextually-based real-time manner (e.g., Microsoft's Grammar Checker [33]) inside the student's IDE. Such an approach could theoretically provide continuous secure programming instructional support throughout the code learning process and across the curriculum. To be successful, it must build upon the lessons learned and knowledge gained from each of academia's approaches.

Modifying classes (e.g., cursory elements added to existing courses) and providing elective courses is a common academic approach to secure coding instruction as it exposes most students to a breadth of knowledge and some students to a depth of

knowledge. The unintended effect of which is twofold. First is the creation of a mindset that views security as a topic that only requires minimal attention. Second is a system that presents security as a secondary topic that students can learn if they are willing to enroll in an elective class. While this approach may not be the best, it does show promise in the possibility of reaching all students in the program with course modifications if the instructors are amiable to the idea.

Security tracks (e.g., certificates / minors) and developed programs (e.g., baccalaureate degrees) are positive examples [41] of academia's approaches to holistic instruction as they develop principles and practices throughout the educational experience [2]. This is because security consideration is an integral and continuous part of the student's educational path. While these approaches create some exemplary graduates, they also only attract a small portion of the students enrolled in a college and secure coding is not always an integral component of the security track. The result of which is a large group of graduates that were not provided with secure coding instruction.

The threaded model acknowledges the need to incorporate security principles and practices at the earliest point possible in the student's education. The concept is to thread security throughout the curriculum so the student will develop the holistic security mindset [7]. Towson University has been successfully accomplishing this task by using checklists in various courses throughout their department [15, 27, 28, 61, 62, 63]. This integration has been positive yet it still requires a commitment of the faculty to relinquish some of the class time for the incorporation of the checklist materials.

All of these models have room for improvement. If the ESIDE approach is to be

successful, it must build on lessons learned from each of these models. Of importance, it must provide continuous, experiential-based instructional support and express the importance of secure coding as early as possible across the curriculum. It must also assume that faculty are not secure coding experts with extra time in their courses.

I believe it is possible to build on these lessons and create a successful instructional approach: An approach that teaches in context by turning the student's IDE into a real-time secure programming instructional resource. Such a resource would enhance the educational process, as the available support would provide contextually relevant, experiential-based instruction. With this in mind, I have examined the potential of the ESIDE approach through an exploration of methods for integrating secure coding instructional support into the IDE so that they might be useful across the curriculum.

1.1 Thesis Statement

I hypothesize that the ESIDE approach to secure coding instruction can be an effective means to educate students on the principles and practices of secure coding as they traverse through their educational experience. In addition, I postulate that the ESIDE approach will serve as an important influence in the development of a security-based mindset amongst student users. By doing so, students will have a transferable skill that allows them to consider the larger security picture and potential vulnerabilities that might exist in any coding environment. In this manner, they are better prepared to either apply their learned remediation skills or seek out resolutions to new vulnerabilities.

To that end, my thesis statement is as follows:

The ESIDE approach to secure code education complements the code learning pro-

cess by integrating real-time secure code instruction into the IDE. When ESIDE is deployed, it is possible to enhance a student's secure programming mindset, knowledge, and abilities with ESIDE.

My intention is to explore the effectiveness of ESIDE throughout the academic experience. At each academic level, I will examine the receptiveness to ESIDE support, the appropriateness of the interaction with ESIDE materials, and the effect of knowledge gained and practices implemented due to said interactions. I aim to use the knowledge gained and lessons learned in this research to further develop an approach that is easily incorporated by both students and faculty throughout the educational experience. To that end, I intend to use the following research questions as guidance during my research:

1. Can ESIDE positively influence a student's secure coding mindset?
2. Can ESIDE motivate and incentivize students to learn about secure programming (vulnerabilities, coding, etc)?
3. Can ESIDE motivate and incentivize students to implement secure programming practices?

1.2 ESIDE Approach Framework

My envisioned ESIDE approach offers real-time educational support within the student's IDE. In simplistic terms, I intend to provide instructional materials the moment a student writes potentially vulnerable code. Because students are contextually "in the moment" when the support appears, students will be more receptive to understanding the meaning of the support. A secondary effect is the exponential increase in instructional exposure. In comparison, typical code-review instructor

support is sparsely available, yet ESIDE has constant availability of supportive instruction, which has been proven to be successful in other instructional areas [53].

In order to provide the secure coding support, I modified the Eclipse plugin known as ASIDE (Application Security in the Integrated Development Environment). Originally, this plugin functioned as a code warning and remediation system for advanced Java developers [79, 80]. By modifying some of ASIDE's functionality, it was possible to change it into an educational support system. The first modification step was to make ASIDE interact with code patterns that are relevant to the student. The second was to change the warning system into a layered education system. With these modifications, ESIDE served as the means to deliver secure coding instructional support.

1.3 Theoretical Framework

As I move forth with the creation of an educational resource, I find myself motivated by the Integrative Learning theory. The Integrative Learning Theory explains how the blending and synthesis of multiple perspectives helps students make connections across curricula [6, 77]. By doing so, the outcome is greater than the sum of its parts [42]. With this in mind, students who are provided with the opportunity to integrate security concepts with programming practices across the curriculum can be expected to have a better depth of understanding than those who acquire this knowledge in separate times and settings. More information about this theory can be found in Section 2.2.1 on page 15.

By keeping the Integrative Learning theory in mind during my work, I am motivated in the creation of a resource that helps students advance their knowledge and skills

across the curriculum. I believe this theory will also serve as a means to help me better frame and understand the effectiveness of integrating a security-based learning resource into a programming-based environment.

1.4 Contributions

The secure programming instructional approaches I have reviewed have been in place for decades, yet insecure programming is still a leading cause of information technology security problems [71]. Unfortunately, this is an indication that those approaches are not doing a good job at nurturing a set of secure programming skills amongst a generation of up and coming professionals. In this thesis, I aim to enhance student secure coding skills through the creation and deployment of an IDE-based instructional approach with ESIDE. I have done this for different levels of programming courses and have evaluated the effectiveness of this approach in realistic environments. Now complete, I have multiple contributions to offer.

One prominent contribution is the initial development of the ESIDE approach. In the Conclusions chapter I demonstrate the potential benefit that ESIDE has to offer to departments, faculty and students. While I developed ESIDE as an Eclipse plugin, the ideology can be applied outside of these constraints.

My second contribution is an understanding of the student's perspective and impressions of ESIDE. This contribution as discussed later in this work highlights why the student does or does not interact, learn, and incorporate supportive materials. I also address what it is that the student likes and dislikes about ESIDE. Future researchers and educators can then use this contribution to further incentivize and motivate students with ESIDE. This will lead to better designs and understandings

on course uses.

I have also contributed a set of specific early, intermediate and advanced secure programming instructional materials for use within ESIDE and outside of ESIDE (e.g., course instructional materials). For each of these, I have also contributed the effectiveness of the instructional materials. In short, the sheer amount of exposure has influenced the student's secure coding mindset. Specific instructional material descriptions can be found in Chapters 4 and 5 and discussed in the Conclusions chapter.

Finally, I will be providing insights into the process of infusing ESIDE into the curriculum. This includes knowledge gained on the technical side of deployment and the human side of working with faculty and students. While discussed in the final chapter, I have found an initial need to work closely with the faculty in early and intermediate courses so that ESIDE complemented their offerings.

1.5 Activities

During this work, I developed the ESIDE approach through the deployment of multiple research activities. In addition to the literature research that I used as a foundation for development, I have conducted formative and summative studies to collect both quantitative and qualitative data. The studies ranged from single setting individual interactions to semester long deployments as conducted with early, intermediate, and advanced students at three different universities. Methods of data gathering included surveys, pre post-tests, interviews, observations, focus groups, interaction logs, and written code.

1.6 Scope of Study

In this research, I have instituted restrictions in an effort to control the scope of my work. As such, I delimited my research with the following conditions:

1. Locations: I conducted all classroom-based studies at the partnering NSF grant institutions [29] or with partners as approved by the ESIDE team.
2. Tools used: Eclipse was the only IDE used.
3. ESIDE development: I served as the main developer of educational materials as provided by ESIDE. I also served as the main developer of user interaction. I worked individually and collaboratively with the software development team on newly developed code.
4. Vulnerability identification: Throughout the development period, the top three of OWASP's Top 10 Application Security Risks-2013 [51] were addressed by ESIDE. I worked individually and with the software development team to help drive further vulnerability identification or development. When appropriate, I created the supporting educational materials for the introduced vulnerabilities.
5. Population: Students and Faculty of participating institutions.

1.7 Researcher Context

Being a University of North Carolina Charlotte (UNCC) Human-Computer Interaction PhD student, a Microsoft intern, a faculty member of Southern Illinois University Carbondale (SIUC) and UNCC, PI on multiple grants, as well as possessing an Educational Administration Doctorate, I have been provided the opportunity to develop a unique set of teaching and researching experiences. During this time,

I have experienced the challenges of providing meaningful learning opportunities to students worldwide. From micro-gaming language learning applications to virtual network security education, I have always attempted to discover new and effective educational methods. I drew from these experiences as I developed a new means to deliver secure coding instruction.

1.8 Organization

I have divided the remainder of this work into several chapters, each of which contains a contributing body of material that helps support my thesis statement (Section 1.1 on page 4). In Chapter 2, I present background and related research of educational practices, vulnerability assessment practices, and integrated support practices. In the subsequent chapters, I present an overview of support tools, completed research, ESIDE tool development, and finish with my conclusion.

in Chapter 3, I provide an overview of the ESIDE tool and previous study findings. By doing so, the reader will be grounded in the interactive aspects of the ESIDE plugin. In addition, the ESIDE plugin has been through multiple iterations as influenced by previous studies. The presentation of these studies serves as a means to indicate how previous lessons learned and knowledge gained have guided my work. Portions of the chapter also serve as a means to distinguish between previous work and my own.

In Chapter 4, I present my advanced deployment studies and in Chapter 5 I do the same for my early and intermediate studies. I use chapter 6 to present the new design of ESIDE as influenced by student and faculty feedback. Finally, in my concluding chapter 6 I summarize on my lessons learned, knowledge gained, and present as well

as future implications.

CHAPTER 2: BACKGROUND AND RELATED WORK

Technologies have redefined our concept of safe and secure. From bank accounts to credit purchases, medical records to electronic voting, we depend on our technologies to protect us and our information. To this end, everyone involved with the development and deployment of our world's software must comprehend the importance of security and take steps to ensure the safety of our technological systems.

Unfortunately, this is not the case as our world is experiencing a shortage of individuals capable of implementing the cybersecurity needed to protect our systems [22, 23]. While academia, industry, and governments have all taken steps to remediate this problem the shortage continues [72].

Academia's response to the world's demands has been to provide training in both historic and innovative ways. They have modified courses, created individual classes, developed new curriculums, implemented security tracks, and integrated thread-based approaches. While each offering has its merits, the amount of students graduating with the skills necessary to design and maintain secure technologies are not enough to accommodate the world's needs any time soon [37].

Industry's response to the shortage is the provision of in-house and public training. So important is this training that at one point, Microsoft's Bill Gates initiated a three-month code freeze while developers received Secure Development Lifecycle [32]

training [4]. Others such as IBM [26], Oracle [43], Adobe [1], CISCO [14] have followed suit with their own training. In an effort to consolidate resources, many companies have collaborated with SAFECode (Adobe, CA Technologies, EMC Corporation, Intel Corporation, Microsoft Corp., SAP AG, Siemens AG, and Symantec Corp.) [56, 57]. The effect of which is a collaborative effort aimed at developing skills and abilities amongst employees and future professionals.

Governments have also made efforts to increase the amount of cybersecurity professionals. Working with its constituents, the United States government developed the National Centers of Academic Excellence in Information Assurance, put in place the Comprehensive National Cybersecurity Initiative (CNCI) [8], and the National Initiative for Cybersecurity Education (NICE) [40] as run by the National Institute of Standards and Technology (NIST) [38]. India is another example of governmental intervention as their University Grants Commission requested their nation's undergraduate and graduate schools to include cybersecurity in their curriculums [65]. The United Kingdom has done similar funding doctoral academic centers of excellence training centers [5].

Drawing from NIST's definition of cybersecurity, these organizations are attempting to create a workforce that can "protect or defend the use of cyberspace from cyber attacks" [39, p. 58]. Security vulnerabilities can be introduced at many phases, but an underlying source of this protection and defense need is the reality that most cybersecurity issues are caused by software vulnerabilities which, in turn, are by created by the programmers who write the insecure code [16, 81, 83]. While programming might be a subset of the training programs available, I believe it is the most important.

By training programmers how to implement secure coding standards, principles, and practices [12, 13, 49, 50] our cybersecurity vulnerabilities would be greatly reduced.

2.1 Secure Coding: Mindset, Concepts and Principles

One of the goals of secure coding instruction is the development of a security mindset [54]. Unfortunately, computer science programs have historically imposed a task completion mindset [35] on their students. They focus on developing an application's functionality (e.g., what it is supposed to do) and once they obtain functionality, they move on to the next task. The problem lies with their "supposed to do" perception as it dismisses the possibility of what an application "can be made to do" which is how attackers perceive applications. Switching this perception is fundamental to the creation of a security mindset.

The security mindset draws from some basic, yet powerful, security principles. At the heart is the goal to ensure that information is confidential, has integrity, and is available (CIA). This means that only those with permission have access to the information, data is true to its representation (un-tampered), and that permission holders have access to the resources when needed. To accomplish this goal, the developer must implement security controls into their code (e.g., limit access to bank accounts), the effect of which is the reduction of exploitable software vulnerabilities.

The security controls consist of a large set of secure coding practices, each of which has specific coding language techniques to mitigate potential vulnerabilities. For example, OWASP has categorized 214 practices into 14 different categories: Input validation, output encoding, authentication and password management, session management, access control, cryptographic practices, error handling and logging, data

protection, communication security, system configuration, database security, file management, memory management, and general coding [49]. To teach each of these specific practices is unreasonable. A more appropriate method is to use specific categories and practices as a means to create a security mindset capable of researching and implementing secure coding practices when needed.

The rest of this chapter is a discussion on relevant topics related to secure coding instruction. I first start with learning theory to create a foundation of the learning process. I then build on this foundation the secure coding principles and practices that students should learn. On top of this, I explore how various institutions are teaching this material. I then finish with the lessons learned and future directions.

2.2 Education: Theoretical Underpinnings

In simple terms, the human learning process involves the acquisition of knowledge, behaviors and skills. A better process constitutes better acquisition. Considering that my overall goal is to enhance secure coding education, I find it necessary to first ground my work in the educational theory of Integrative Learning. I intend to use this theory as means to frame my educational approach.

2.2.1 Integrative Learning Theory

The Integrative Learning Theory addresses the potential of making connections across curricula [6, 76]. The concept is that our brains do not categorize facts based on disciplines but rather, connect stored information based on knowledge and use patterns. The greater the connection, the easier it is to recall and apply information. When the learning process facilitates these connections, students learn more effectively [9].

Importantly, Integrative Learning Theory frames education as a holistic development process that synthesizes multiple curricula perspectives into a complementary experience [6, 76]. Common in the medical schools, this approach produces students that are better prepared to address complex problems by bringing concepts and methods together from multiple areas [77].

In relation to secure coding, the ESIDE educational approach promotes connections across the curriculum. Considering that security plays a role in all forms of IT, it is more than appropriate to use ESIDE as a means to infuse course specific security concepts and principles across the curriculum. By doing so, students will find it easier to make connections on how security is a fundamental component to all aspects of information technology. It will also prepare the students to seek out these connections in situations that do not promote security as a topic of importance.

2.3 Secure Coding Education

Implementing secure coding education in post-secondary institutions is not a straightforward task. As is common, there are many influences on curriculum (what you teach) and instruction (how you teach). Government and industry feel that there are specific coding practices [12, 13, 49, 50, 58] that graduates should have, which might conflict with academia's educational objective to broadly teach abstracted principles and theories. If academia merely trains their students then they become reliant on industry and run the risk of teaching stagnant information. On the other hand, academia must create employable graduates.

The resolution is for academia to maintain a balance between the two. Programs must train for societal needs whilst educating for conceptual growth. To this end,

secure programming instruction in academia should infuse industrial practices into their educational goals while still teaching general principles.

2.3.1 Secure Coding Educational Goals

Drawing from the Summit on Education in Secure Software: Final Report [7] and ACM's Computer Science Curriculum [55], upon graduation students should: Have a basic understanding of security, understand security's role during design and deployment, know current industry and government security issues and related remediation methods, understand usable security, evaluate external code, have experience, and be able to "think like an attacker." To accomplish these goals and maintain a balance between education and training, academic programs should implement secure coding practice models at every stage of the instructional process to holistically develop the student. The following are recommended implementation practices for the early, intermediate and advanced stages of education.

2.3.2 Secure Coding Practices - Early Through Advanced Students

As the student moves through the instructional process, there are key points where secure coding concepts and practices can complement a course's instructional content. The first points occur with early students as they are learning the basic principles of programming. The following is presentation of those points mapped to the student's level as influenced by reviewed syllabi, courses, and faculty/government/industry recommendations [11, 49, 59, 66]. While the following draws from the typical Java learning progression, these interventions are also applicable to other languages.

2.3.2.1 Secure Coding Practices for Early Students

Early students develop a foundation of coding knowledge by mastering the principles of variables, objects, assignment statements, conditional statements, loops, i/o, arrays, and functions. Secure coding lessons should complement this process in two stages. The first stage is during the period where students control all their data and have no external data entering their code. At this point, it is important for them to learn the following:

- Data types: Students should know the limits / ranges of each data type (e.g., byte = -128 to +127, short = -3,768 to + 3,767) and the problems that could arise if the wrong data type is used (e.g., expecting a number and using a letter).
- Data reasonableness: Students should learn to reflect on the type and size of data they are using and what affect it might have on their code (e.g., a very large number in a loop).

The commencement of the second stage begins with the introduction of external data into the student's code (e.g., console input, JOptionPane). At this point, it is critical that the student begins to develop their security mindset by reflecting on what might happen to their program if they receive unexpected input (e.g., strings instead of numbers). Once they have considered possible problems, they should learn how to validate obtained data. The following is a list of input validation points that students should learn how to implement at this stage:

- Data type checks: Know how to verify that numbers are numeric characters and names are alphanumeric.

- Bounds check: Know how to ensure that input falls within a certain range or size (e.g., age should be between 0 and 150 or the length of a name should be between 0 and 100 characters).

By having early students consider potential problems with unexpected input and put in place these two checks, they will be well on their way to developing a security mindset.

2.3.2.2 Secure Coding Practices for Intermediate Students

Students at the intermediate level expand their knowledge base by learning how to apply such things as exception handling, recursion, unit testing, application programming interfaces, hash maps, and simple graphical user interfaces. During this process, students should continue to consider potential problems with unexpected input while they learn how to apply the following security concepts:

- Input validation: Check that the data is of the correct type, it is in a reasonable range and of reasonable length.
- File management: Validate supplied filenames, ensure file paths are correct.
- Error handling: Implement a means that recovers from errors (e.g., try-catch) and a method that creates logs of those errors with pertinent information (e.g., data is out of bounds).
- Output Encoding: Sanitize all output to entities such as clients, OS commands and data queries (e.g., SQL, XML, and LDAP).

The main goal of integrating these security concepts into the intermediate student's learning process is to enhance the security mindset. The most important take away during this stage (and others) is that the student identifies potential vulnerable areas

and uses problem-solving skills to remediate security issues. By doing so, they are on their way to becoming robust programmers.

2.3.2.3 Secure Coding Practices for Advanced Students

Advanced student programming becomes more intricate and technical as students learn to implement robust interactions in their applications. Using an advanced network-based application development course as an example, students learn about computer networks, web technologies and standards, network-based programming methodologies, languages, tools and standards. The implementation of these topics help students develop advanced Java skills and techniques.

This expansion of learning into interactive applications makes for a wide range of secure coding educational opportunities. Using OWASP's secure coding practice guide as a reference, the following is a condensed list of topics that advanced students should understand upon graduation. The expanded version can be found at [50].

- Input Validation: Encode to a character set (UTF-8) then white list validate data on a trusted system whenever possible. Reject failed input.
- Output Encoding: Sanitize all output to entities such as clients, OS commands and data queries (e.g., SQL, XML, and LDAP).
- Authentication and Password Management: Fail authentication securely, all controls are implemented on a trusted system (The server), passwords or connections should be encrypted, require long complex passwords, obscure password entry, enforce temporary password change, reset links should have a short expiration time, and re-authenticate prior to performing critical operations.
- Session Management: Use random session identifiers as created on a trusted

system (The server), make logout available on all protected pages and have it fully terminate the session, implement a short timeout, no concurrent logins with same user ID, and periodically generate a new session identifier.

- Access Control: Restrict access to files, functions, services, application data, user and data attributes, and security information. Limit transactions based on time, force logouts if privileges have changed, and implement a least privilege policy.
- Cryptographic Practices: Perform a cryptographic protecting functions on a trusted system (The server), fail securely, use an approved random number generator.
- Error Handling and Logging: Implement generic error messages, free allocated memory when an error occurs, ensure important events are logged, restrict access to logs, log input validation, authentication and access control failures, and possible tampering events.
- Data Protection: Least privilege, encrypt sensitive information, remove revealing comments and documentation, disable auto complete, and implement appropriate access controls to stored server data.
- Communication Security: Encrypt sensitive data, use transport layer security (TLS), and specify character encodings for connections.
- Database Security: Use parameterized queries, use input validation and output encoding, use secure credentials for access, and close connection as soon as possible.
- File Management: Don't share absolute paths, white list file names and types,

limit file types to be uploaded, and require authentication prior to uploading a file.

- Memory Management: Ensure buffer is as large as specified, check buffer boundaries is calling functions in a loop, don't rely on garbage collection.

While not exhaustive, this list provides solid guidance as to the types of practices the advanced student should consider as they develop applications. At this level, the students should have a developed security mindset that is capable of identifying potential issues and implementing a means to remedy them.

2.4 Secure Coding Instruction in Academia

There are many challenges to providing secure coding instruction in post-secondary institutions: Academia is slow to change, curriculums are tight, course content is at capacity, faculty have limited secure coding backgrounds, faculty have limited time or motivation to learn new topics or modify courses, and only a minimal amount of textbooks cover secure coding [7, 35, 62, 63, 64, 70, 73]. Over the past decade, institutions have incorporated various models into their curriculums in an attempt to overcome these challenges. These models include single courses, new curriculums, security tracks, and thread-based approaches.

2.4.1 Secure Coding Instruction with Single Courses

Single course instruction is a simple resolution for institutions. It requires one faculty member with the expertise or willingness to train. The course complements the curriculum by introducing students to relevant principles and practices. Typically offered as an elective later in the educational process, this course develops the student's perspective of security in a broad manner. Example topics covered in secure

programming elective courses [24, 34, 68] include input validation, injection attacks, logic errors, integer arithmetic, buffer overflow and least privilege.

Because the course typically complements a curriculum, there is little uniformity on course pre-requisites or topics [52]. Due to this randomness, there exists little research on the effectiveness of single courses. In relation to acceptance as a means to instruct secure coding, many believe it is too little too late to too few students [3, 7, 15, 63, 70, 73, 74].

2.4.2 Secure Coding Instruction with Security Tracks

The National Security Agency (NSA) recognizes the security track approach as an acceptable method of security instruction. Institutions who meet the stringent certification criteria are awarded the title of National Center of Academic Excellence in Information Assurance Education [41]. Using Towson as an example [2], this institution added multiple security-based electives to their program. These courses take a holistic education approach and require students to learn the topics: Ethics, information security, cryptography, network security, application software security, operating security, and case studies in computer security.

Many schools (>181) have taken the security track approach [41] and have educated multitudes of graduates. These actions appear to be very effective in the training of students in a wide range of information security topics. However, as indicated by Towson faculty, not all students enroll in this track [28, 62, 63, 64] and the topic of secure coding is a small subset of the track's available electives. For example, of all the National Center of Academic Excellence in Information Assurance Education requirements, only 2 out of 17 knowledge units (basic scripting and intro

to programming) address secure coding topics (e.g., bounds checks, input validation, type checking, and parameter validation). Of the 41 optional knowledge units, two of them mention defensive programming topics (e.g., SQL injection). The secure programming knowledge unit is the only one that directly addresses the secure coding topic [10]. This is evident in University of North Carolina Charlotte's Information and Security Privacy Masters concentration program which only offers 11 security / privacy courses with one of them being a secure programming elective course [69]. Arguably, secure coding is an important enough topic that all students receive instruction. Therefore, the core programming courses of the curriculum must include secure coding principles and practices.

2.4.3 Secure Coding Instruction with a Thread-based Approach

The breadth of secure coding instruction is quite inclusive as it touches all programming courses at all levels. To reach all students, in all courses, at every level an alternative approach is necessary. The thread-based approach appears to be a promising alternative to meeting these needs.

Popularized by Towson University [28, 60, 61, 62, 63, 64], the software security thread-based approach integrates touchpoints into existing opportunities of the established curriculums. These "security injections" [66] are targeted lab modules (e.g., integer overflow, buffer overflow, input validation) that contain academic level dependent secure coding tasks (e.g., CS0, CSI, CSII), checklists based on security principles (completed by the student), and scorecards (standard means for evaluation). An example of a task for CS0 input validation is to print out an age averaging program and use a checklist to indicate each variable that is input and then check off the applica-

bility of specific checks such as length, range (reasonableness), format and type [67]. The checklist is then used as a scorecard by the instructor.

In 2008, Taylor et. al [62] piloted their software security thread-based approach in three sections of Towson's CS0 and five sections of CS1. They discovered an increase in security knowledge and a positive reception to the security topics. Researchers then implemented lessons learned and knowledge gained into a CSII study to discover similar results [63].

Taylor et. al [63] continued injection module development and began disseminating it in collaborating institutions (two and four year post-secondary) over the next four years [64]. Results continued to be positive in that instructors and students were receptive and student post-test scores increased. Nance et. al [64] has proposed that the security injections could be introduced at the high school level which would facilitate the introduction of security at the earliest point possible.

The threaded approach helps overcome some of the barriers to secure coding instruction as it can be integrated across the curriculum, can reach all students, needs minimal faculty commitment, and can complement the learning process. The only apparent difficulty with implementing such an approach would be faculty acceptance, the development of injections that dovetail into a plethora of topics, and disseminating the information. Even with these barriers, this approach is promising.

2.4.4 Lessons Learned and Future Directions

It is evident that secure coding education needs to be taught at the earliest stages to all students as often as possible [62]. The single class solution is not capable of this as it typically is a high-level course offered as an elective at the end a program. Security

tracks do a wonderful job at educating a select few yet even these few take minimal secure programming courses. The thread-based approach has the potential to educate the most students across the curriculum yet there is still room for improvement.

Drawing from theory, adult students learn best in an experiential environment where they can continuously make connections amongst topics. While the aforementioned secure coding approaches might incorporate experiential learning, they might be out of sync with the learning process which makes it more difficult to connect topics. Realistically, students spend most of their time outside of the classrooms and away from security injections developing their own coding habits. The level of security incorporated into their code is a result of their memory of classroom instruction or their ability to research methods on their own.

Current approaches have yet to capitalize on the time the student spends in their own learning environment. I believe a solution is to move secure coding instruction into the students' IDE with an automated resource such as ESIDE. Such an effort can enhance the learning process by contextually increasing the teachable moments in a manner similar to Microsoft's Grammer Checker [31, 33, 53].

CHAPTER 3: ESIDE DEVELOPMENT AND INTERACTION

The ESIDE Eclipse plugin [47] plays a primary role in the deployment of my approach to integrate secure-code education into the IDE. Its role is to provide the contextually relevant, real-time educational materials in the student's IDE. It does this by searching for specific code patterns and when it finds one, it provides a layered educational opportunity to the student.

ESIDE was modeled after the ASIDE plugin whose purpose was to serve as a code warning and remediation system for advanced Java developers [79, 80]. ASIDE would search for vulnerable code patterns and when found, it would highlight the code and provide an actionable icon. When the user clicked the icon, ASIDE would provide a list of remediation options such as dynamic code generation and code examples.

With minor modifications, the warning and remediation framework becomes the foundation for the ESIDE approach. I accomplish this by having ESIDE search for code patterns that are relevant to the student and by having the warning mechanism changed so it provides a layered educational opportunity. The available functionality, the types of code identified and the educational content provided are all reliant on the level of the student (early, intermediate, and advanced). The advanced student could potentially have all of ESIDE's functionality available to them. The early and intermediate students interact with scaled down versions. In the following, I

provide a fundamental overview of ESIDE with examples for advanced students. A more detailed description of ASIDE's development and future directions can be found at [47, 79, 80, 81, 82, 83].

3.1 Targeted Vulnerabilities

Java has many code related vulnerabilities and remedies that are beyond the scope of the student's developmental needs. Therefore, it is necessary to reduce the list down to the areas that would be most beneficial to the student as discussed in Chapter 2 and based on faculty consultation as well as professional recommendations [51], the list includes the types of vulnerabilities that are caused by lack of input validation, output encoding, and SQL Injection. In the following, I provide a cursory review of these Java vulnerabilities as framed into injection flaws and output encoding flaws prior to discussing the ASIDE plugin.

3.1.1 Injection Flaws

For the most part, a computer application follows a set of instructions given to it by the developer. An injection flaw exists when external data can change those instructions so that the application acts in unexpected ways. An analogy for this concept could be the use of truth serum. Inject it into a suspect and they reveal everything.

In computing, an attacker will typically exploit this flaw by sending (injecting) data in a manner that causes the meaning of a command (instruction) to be changed. An application can mitigate such an attack through the process of input validation. What this process does is cleanse the data of nefarious entries.

Most risks to applications involve the failure to cleanse external data [44, 46]. This

failure opens the door to major vulnerabilities such as Cross Site Scripting (XSS), SQL Injection and file system attacks to name a few.

3.1.2 Output Encoding Flaws

An output-encoding flaw exists whenever a user's viewer (e.g., browser) can mistakenly interpret received information as executable code rather than displayable data. For example, an attacker disguises a cookie stealing script as a link and posts it on a vulnerable forum website. If a user clicks on the link, the script sends their session cookie to the attacker. The attacker then uses the cookie to steal the user's session and falsely interact with the session as the user (e.g., post as user).

The output encoding (aka escaping) process helps mitigate this flaw. This process converts specific characters so that they are interpreted as data rather than syntax-significant characters [45, 78].

3.2 ESIDE Interaction Overview

The ESIDE plugin handles the back and front end of the student's educational experience. It continuously runs in the background searching for vulnerable code patterns. Once found, the educational process is triggered. Interaction begins with the presentation of a clickable icon (Figure 7 on page 58) followed by a hover message (Figure 8 on page 58) that provides a brief explanation. A single click of the icon provides an explanative popup and a list of available remediation options (Figure 9 on page 59). A single click of each option reveals its own explanative popup. Double clicking on the option will either generate secure code or help the students to resolve the vulnerability with an in-depth remediation explanation. Comments included with generated code also include explanations.

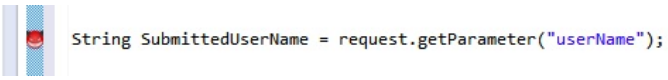

Figure 1: Warning icon

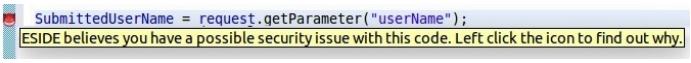

Figure 2: Hover warning

3.3 ESIDE Warning Icon

The warning icon (Figure 7 on page 58) serves multiple purposes in the ESIDE approach. On one hand, it acts as a direct means to indicate an issue and start the learning process. On the other, the repeated exposure should have the effect of causing students to reflect on the reason as to why their code causes the icon to appear, the result of which will be a connection between code practices and icon appearance. Such a connection will lead to a student's ability to predict when the icon will appear. This predictive ability is an indication of a mindset change.

The next stage of instruction begins when a student rolls their pointer over the icon. This action reveals a short, high level explanation that a potential vulnerability might be present and that a single click of the icon will reveal more information (Figure 8 on page 58). The intent of this short message is to educate about the icon's purpose and to instruct on how to reveal more information about the potential vulnerability.

3.4 Remedy List and Accompanying Popups

The next educational layer becomes available when the student clicks the ESIDE warning. Doing so reveals a list of directly related, student level specific, remediation options (Figure 9 on page 59).

Each of these options has their own overview popup. The vulnerability overview

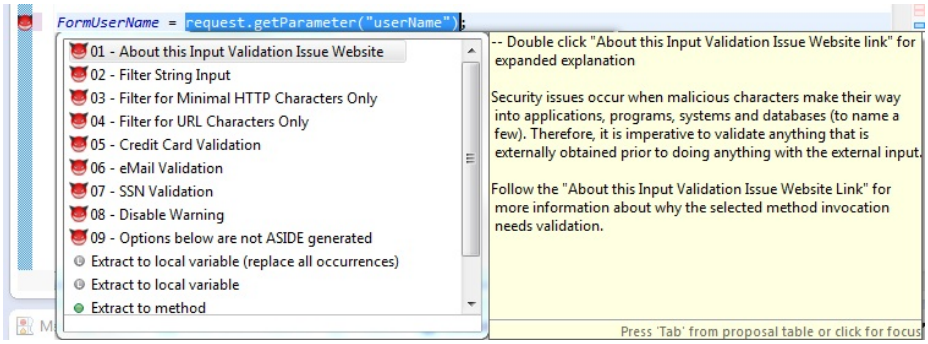

Figure 3: Input validation remediation choices

explanation serves as the default option and provides a general overview of the vulnerability category that the student introduced into the code (e.g., input validation, output encoding, SQL database communications). The purpose of the popup is to help students generally identify, understand, and categorize potential vulnerabilities. During this stage, students also have the option to explore a more in-depth explanation as provided by activating the more info option.

3.5 Educational Pages: More Info Option

The educational pages contain a comprehensive explanation of discovered vulnerabilities and their associated remedies. Divided into three categories (input, output, and database communications), these pages provide in-depth learning resources. Each of these pages starts with an overview of the problem then provides descriptive examples, remediation methods, and links to further information. The content found on these pages will differ depending on student and course levels as described in 3.9 on page 33.

3.6 Real Time Educational Support

One important feature of ESIDE is the real time educational support. The benefits of this feature are twofold. First, the delay that is typically present in the turn in

assignment, wait for feedback, and implement feedback into future work model is non-existent. Second, the real-time support is available when the student is in their learning mode. During this time, the student is more receptive to integrating new materials as a connection to security principles and practices is being made.

3.7 Interactive Code Refactoring

ESIDE also has the ability to generate remediating code (Figure 5 on page 33). When the student activates this option, educational materials are included in the created code. The content of these materials range from simple messages included in the method such as "SafeString" to a more expanded explanation included as comments which will be discussed in subsequent sections and shown in (Figure 12 on page 93).

3.8 Typical ESIDE Interaction Process

The following is an example of a typical ESIDE interaction process that uses input validation as the context of the interaction.

1. Vulnerable code written:

   SubmittedUserName = request.getParameter("UserName");

2. ESIDE warning icon appears in the left hand marker tray (Figure 7 on page 58).
3. Short message provided when icon is hovered (Figure 8 on page 58).
4. Remedy choices with supporting popups become available when icon is clicked (Figure 4 on page 33).
5. User selects the 02 - Filter String Input option.
6. ESIDE generates code to sanitize the string (Figure 5 on page 33).

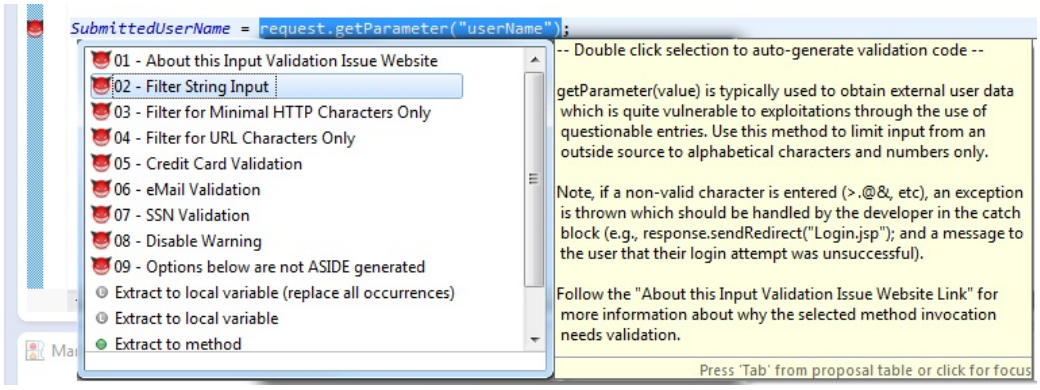

Figure 4: String remediation selection

```
SubmittedUserName = ESAPI.validator().getValidInput("replace ME with validation context",
                    request.getParameter("userName"), "SafeString", 200, false);
```

Figure 5: Generated code for safe string

As shown, ESIDE interaction is simple and straightforward. Minimal mouse clicks provide students an easily accessible learning opportunity. This style of interaction can support students at all levels by modifying the available educational materials and the code patterns relevant to the student level as discussed in 3.9 on page 33.

3.9 Multi-level Support

One of the primary challenges of the ESIDE approach is the provision of support that aligns with the student's development level. Early students will need fundamental information that helps them form a well-rounded knowledge base while advanced students require information that is more technical in nature.

3.9.1 Early-level Support

Early students are just learning the basics of i/o, loops, and arrays. As these principles will serve as the foundation for future coding development, the student should also concurrently learn how to implement the corresponding secure coding concepts (e.g., bounds checks, type checks, and character checks).

ESIDE complements this foundational development period by providing targeted interaction for the student. The point of entry for this interaction is during the period where the student starts to implement console level input and output calls. This developmental period is crucial as it is the first time that students introduce external data into their program.

The educational content provided during this period are reflective in nature. Their intent is to facilitate the thought process of how unexpected input can cause problems. ESIDE encourages this reflective process in a text-based, simplistic manner. Because it is so important to possess a mindset that includes this thought process, ESIDE does not provide auto-generated code as instant resolution might detract from the learning process.

3.9.2 Intermediate-level Support

Students at the intermediate level are learning how to develop basic Java driven applications. They have learned the fundamentals of Java and are learning how to make their applications interact with multiple external entities. This interaction typically comes from communication with flat files, user input from GUI's and output to those GUI's.

During this developmental period, ESIDE expands the student's knowledge base and mindset by providing data sterilization materials. Based on input validation and output encoding, these materials go into greater depth than those available to early students.

Also during this period, students learn how to create and interact with various classes. To facilitate this development in a secure manner, ESIDE also provides

code generation with instructional comments. The educational intent is to help the student begin to understand the process of implementing classes that are specifically written for the purpose of security such as ESAPI (The OWASP Enterprise Security API) [48].

3.9.3 Advanced-level Support

Students at the advanced level have mastered the basics and intermediates of Java. Their programming is in the process of becoming more intricate and technical as they learn to implement robust interactions in their web applications. Not only does this implementation help the student develop advanced Java skills and techniques, but it also helps them come to understand the process needed to store and retrieve database information using SQL communications.

During this technical developmental period, students are just below the introductory professional level. Therefore, ESIDE provides a full assortment of support features that range from auto-generated data filtration code (Figure 12 on page 93) to prepared SQL statements. Because the students are developing code that potentially has real-world vulnerabilities, ESIDE provides information on real world attacks like Cross Site Scripting (CSS). This exposure continues to expand the student's security mindset by conveying the need to consider potential issues at every level.

3.10 Under the Hood

While I have focused my work on the educational implementation of the ESIDE plugin, it is important to provide a high-level overview the inner workings of this resource. By doing so, others might come to understand the flexibility and adaptability of the ESIDE tool and therefore incorporate such a resource in their own educational envi-

```
    /* NOTE: If the following ASIDE generated code detects a problem
     * (e.g., malicious characters entered by user) an exception is thrown.
     * Doing so will skip the rest of the try block code and go directly to
     * (execute) one of the generated catch blocks below.
     * */
    String SubmittedUserName = ESAPI.validator().getValidInput("replace ME with validation context",
            request.getParameter("userName"), "SafeString", 200, false);

    return macAddress;
} catch (ValidationException e) {
    /* This catch block is executed when ASIDE finds input that did
     * not match validation rules (e.g., bad user input). When this happens,
     * the developer should maintain usability in this catch block
     * (e.g., response.sendRedirect("Login.jsp"); with an accompanying
     * unsuccessful attempt indication presented to the user).
     * */

     // Note: Default return generated by ASIDE
    return null;
} catch (IntrusionException e) {
    /* This catch block will be executed when advanced
     * intrusion behavior is detected in ASIDE's try block
     * statement. This exception should also be handled by the
     * developer similar to the ValidationException block.
     * */

     // NOTE: default return generated by ASIDE
    return null;
}
```

Figure 6: Advanced generated code

ronment. A more detailed description of ESIDE can be found at [79, 80, 81, 82, 83].

ESIDE works as a long running background process in the Eclipse IDE. It searches for code patterns that match a set of XML-based heuristic security vulnerability rules. An example of how the rules work with input vulnerabilities is as follows:

> "... two types of rules are supported; Method (API) invocations: for example method getParameter(String parameter) in class HttpServletRequest introduces user inputs from clients into the system; Parameter input: for example argument of the Java program entrance method main(String[] args)." [79, p. 2].

The rules that ESIDE uses are written in XML. This allows for customization of specific code patterns being searched which is useful for education as the interaction can be tailored for specific learning levels (e.g., early, intermediate, advanced).

3.10.1 Code Generation

The option to auto-generate remediation code is available in the intermediate and advanced levels of support. The intermediate level of code generation support implements simple regular expression routines. The advanced version implements the standard validation and encoding routines as available from the open source secure application library, ESAPI [48].

3.11 Previous ASIDE Studies: Lessons Learned

Developers of the original ASIDE plugin created a remediation-based resource and targeted advanced students and professionals as their users. In the following, I review their research contributions as framed into interaction, education, and reliability. My intent is to establish a foundation of ASIDE research knowledge and to differentiate previous work from my own.

3.11.1 User Interaction

In the Spring of 2011, Xie et el. developed ASIDE as a proof of concept plugin for Java in Eclipse [80] and researched if the plugin could serve as an effective aid for programmers by raising awareness and promoting secure programming. Then in 2012, Xie et el. examined if the option to remediate vulnerabilities through code generation (CodeGen Group) would be more effective than an instructional-based option (Explanation Group) [82]. Both studies used advanced students or professionals as their participants.

Results from the first study identified that ASIDE was effective in helping:

> "students to write more secure code when using code refactoring. They

appear to pay attention to ASIDE warnings and follow ASIDE advice to perform input validation and/or encoding" [80, p. 275]).

When researchers tested the effectiveness of code generation over guided instruction, they discovered that only those who had CodeGen available to them implemented code remedies. The Explanation group shared that the instructional material was interesting but that it contained too much security jargon and did not provide concrete remediation examples. Importantly, there was evidence that functionality was of primary concern for both groups. So much so that a participant had removed generated code because they incorrectly thought that it had broken the application. Lastly, all but one of the student participants indicated that they would have been unable to identify the vulnerabilities they created without ASIDE.

Jing et el. identified that the one of the main contributions of their work was to "remind and assist developers with possible mitigation actions" and that "studies involving experienced developers, are needed to show whether ASIDE can improve developers' understanding and practice of secure programming in more contexts" [80, p. 275]. While the researchers were looking at the support of advanced users, they also opened the door to using this plugin as a complementary educational piece.

Of particular significance and influence to my work, almost all participants lacked secure programming knowledge. This was evident with the struggle to understand provided explanations and generated code. In addition, the functionality mindset often took precedence over secure programming. Therefore, I will need to provide educational materials that are not only understandable by the student but also capable of creating a mindset that includes secure programming.

3.11.2 Education

In 2013, Zhu et al explored the potential of the ASIDE plugin to impact learning and practice in an educational environment [83]. After a three-hour study, researchers discovered that students did spend time with the educational offerings and that their secure coding knowledge base increased as reflected by the scores on the post survey. While ASIDE exposure did influence code writing (e.g., code generation or manual implementation), researchers also discovered that the students were confused with the various remediation options available to them.

It is possible that the labels of the options, amount of options, description of the options, or a combination of any or all could have caused the confusion. It is also possible that the duality of learning two new concepts at the same time (functionality of new concept and security of new concept) could have contributed to their state of confusion.

In relation to contributions from this work, learning does occur and code practices do change when students interact with ASIDE. However, students did experience confusion when they interacted with ASIDE. It will therefore be necessary to create educational materials that are appropriate for the student level.

3.11.3 ESIDE Reliability

When providing educational resources, it is paramount that those resources are reliable. Fortunately, ASIDE has been proven to be reliable in the identification of vulnerabilities as compared to static analysis performed by the professional security audit tool Fortify SCA [25]. ASIDE was also proven to be reliable when a professional

compared ASIDE against a security audit based on a manual review of the Fortify findings [80].

3.11.4 Lessons Learned, Knowledge Gained, Directions Influenced

It is important to differentiate between accomplishments of previous research with my current and future research. Original researchers designed ASIDE to be a resource for advanced programmers. Only once did the previous researchers assess ASIDE as a learning resource. For the most part, the integration of secure coding educational materials into the IDE is an unexplored research topic.

The overall goal of my research is to enhance the current state of secure programming education. My intention was to build upon the lessons I learned from previous research in hopes to answer my research questions. I therefore aligned those lessons in accordance to my research questions in the following.

"Can ESIDE positively influence a student's secure coding mindset?" Previous researchers found that students maintained a functionality mindset to the point of rejecting secure-code support [82]. I believe the best way to remedy this issue is to introduce security as early as possible as such an exposure is key to the development of a security mindset [7, 36].

"Can ESIDE motivate and incentivize students to learn about secure programming (vulnerabilities, coding, etc)?" Researchers discovered that students became confused when presented with secure coding information [82, 83]. However, their knowledge base did increase after a minimal exposure to ASIDE [83]. To overcome this interesting phenomenon and increase the overall knowledge gain, I developed the educational interaction materials in a manner that is understandable and appropriate for the

student's level.

Lastly, "Can ESIDE motivate and incentivize students to implement secure programming practices?" Researchers discovered that students did alter their code when code generation and guided instruction was available. However, students still experienced confusion when interacting with ASIDE [83]. In an educational sense, I believe this is a false positive result because students should understand what it is they are implementing. Therefore, I capitalized on this lost educational opportunity through the creation of educational interactions that help properly guide the acquisition of secure coding abilities.

Taken as a whole, it is apparent that the next step is to change the mindset of up and coming professionals by providing educational interactions that complement the functionality lesson at each level of student development. In doing so, students will be better prepared to implement a simplistic means of vulnerability remediation. The challenge is to understand and implement knowledge transfer opportunities in such a manner as to dovetail ESIDE's educational exposure opportunities with that of the learners' level of educational progression.

In pursuit of this knowledge, I have conducted seven ESIDE studies with early, intermediate and advanced students at multiple institutions. These works were conducted as formative studies with advanced students and summative studies with early and intermediate students. I use Chapter 4 to discuss the advanced studies as this work expands on previous ASIDE research findings and Chapter 5 covers the early and intermediate studies as the area is really new and the work is very formative.

CHAPTER 4: ESIDE ADVANCED STUDIES

4.1 Introduction

Previous researchers evaluated ESIDE in a single, 3-hour laboratory study with students in an advanced computing course [83]. Results indicated the potential for the tool to raise awareness of secure coding, as well as increase students' secure coding behaviors. In this chapter I have extend these results with two field studies - one for the period of one assignment and the second for an entire semester, in the same advanced computing course. My results demonstrate that ESIDE does raise security awareness, but that the timing of the tool's introduction, and the support of instructors for tool use may be critical for motivating students to learn and practice secure coding skills.

4.2 Methodology

The overall goal of ESIDE is to enhance the current state of secure programming education. Previous researchers had evaluated ESIDE in a controlled environment for a limited amount of time [83]. Students were aware they were evaluating ESIDE, which may have influenced their attention and interest in the tool. In contrast, the studies I am reporting on here examined ESIDE as deployed in the student's work environment over the period of a single assignment and a whole semester. Student participants were drawn from two sections of University of North Carolina Char-

lotte's (UNCC) Network Based Application Development (NBAD) course which has advanced students (senior undergraduates and Master's students) incrementally develop a Java-based Web application over the period of the semester.

One of my goals is to make ESIDE available for student use throughout the curriculum at all levels. These studies provided me the opportunity to explore ESIDE's viability of achieving my goal. I utilized multiple data gathering techniques to obtain feedback about ESIDE's potential to influence the advanced student's mindset, knowledge base and coding practices as well as to improve the tool for future implementations. Specific goals included:

- How will students interact with ESIDE over time?
- What are the students' impressions of ESIDE?
- How does ESIDE influence the students' awareness and mindset of security?
- Can ESIDE help students advance their secure coding knowledge?
- Will use of ESIDE influence students to integrate secure coding practices into their assignments?

Two instructors from both sections of UNCC's NBAD course allowed us to invite their students to participate in the single assignment study (fall 2012) and the full semester study (spring 2013). Instructors of this upper level Java-based web application course have their students use Java Servlet technology to develop applications such as an online stock trading application or a flight reservation application or a banking application. Throughout the semester, the NBAD students are minimally exposed to secure coding discussions, presentations or examples (e.g., a few slides).

The structure and data gathering techniques for these studies were similar. ESIDE

was available in the college's computer lab and personal pc installation instructions were available through the class Moodle site. For both studies, I provided students with a brief overview of the functionality of ESIDE and then asked them to interact as little or as much as they chose throughout the study. Students from both studies also completed a pre and post-test on secure coding, provided assignments for code reviews, allowed ESIDE to submit interaction logs, and participated in semi-structured and informal interviews. The semester long study also included a focus group.

Interviews consisted of informal interactions throughout deployment and a semi-structured interview at the end of the assignment study. The topics of ESIDE use, impression, and suggestions were explored with this group. Students were also provided with the opportunity to talk about the importance of secure coding. Interview times ranged from a few minutes to 40 minutes. Participation was voluntary, and students were not compensated for participating in the study, although pizza was provided for the focus group.

4.2.1 Study One: 14 Day Assignment

Eight students from two sections (different instructors) of my UNCC's Fall 2012 NBAD course participated in the 14-day assignment-based study. Both section instructors followed the same syllabus. I deployed the study on the fourth assignment when students were required to integrate a higher level of user interaction into their application. They had previously developed static pages for the purpose of login, logout, registration and were then required to implement functionality on top of those artifacts. This functionality included the ability to add accounts, display stock details, buy/sell stock and display stock history. Seven students participated in an

interview once their assignment was submitted.

4.2.2 Study Two: Semester Long Deployment

Two sections of NBAD students (different instructors) participated in the Spring 2013 semester long study. In total, 64 students consented to participate in one form or another. Specific numbers for data gathered will be discussed in their respective sections.

Section 1 completed four assignments. The first three progressively developed a stock trading Servlet application. The class started with static HTML as a framework for subsequent assignments that became more dynamic. The final assignment incorporated AJAX to develop a banking Servlet application.

Section 2 had four assignments to complete. These assignments progressively developed interaction with an MVC-based online airline reservation application and, eventually, a banking application. The third assignment incorporated AJAX for communication between the applications and the fourth assignment built further on the communication with the incorporation of XML and SOAP.

A focus group was held halfway through the semester and was used to ascertain ESIDE usage and interaction. I displayed code on an overhead projector and walked through various forms of interaction the students were exposed to while working on code. Doing so helped facilitate a discussion about their impressions and suggestions about the available interaction (e.g., icons, menus, code gen, etc.) as well as how well the available content supported them. I also discussed their perspectives on secure coding and if ESIDE influenced any change in their behavior.

4.2.3 Pre/Post-tests and Survey

Participants from both studies completed a pre-test and demographic survey (either online or on paper) at the beginning of their respective study. After they finished their assignment or at the end of the semester, they completed the post-test. Content reflected the concepts and principles represented by ESIDE.

For both studies, I drew from pre/post-tests that were developed by previous researchers [83] and designed two pre/post-test versions with similar questions which I counterbalanced between the two class sections. Study 1 surveys consisted of 17 yes/no – true/false questions that ranged from general questions about input validation and output encoding to the identification of code statements that required input validation and output encoding. I condensed the semester long survey down to 10 questions. The following is an example of two survey questions.

1. True / False My application is only used within the company, therefore input from users and other company computers can be trusted.
2. Yes / No Is the following vulnerable to an injection attack, an output encoding attack or both?

   String homeState = request.getParameter(state);

   out.println("Link your home state");

In addition to the secure coding test questions, I also included qualitative survey questions on the post-test which asked about ESIDE impressions and recommendations.

4.3 Results

In total, there were 72 (8/64) students in the two studies. For those who reported, 53 (8/45) were male, 13 (0/13) were female, 29 (3/26) were graduate students, and 29 (5/24) were undergraduates. ESIDE submitted 57 (7/50) individual interaction logs.

4.3.1 Logged Usage, Interaction and Behavior

I utilized a logging feature to track when ESIDE was running, what vulnerabilities it found and how much the students interacted with the available resources. For Study 1, ESIDE submitted logs for 7 individuals. These students had ESIDE running and available for an average of 3.57 days (1-7 day range) and initiated interaction (hovered or clicked the code warning icon) 513 times. This averaged to 7.3 interactions per active student per day over the span of 10 days. No anomalous usage spikes (e.g., everyone accessing the day the assignment is due) were found over the period of the study.

Study 2 ran for 95 days. Logging activity began in the third week as that is when students start writing targeted code. From that point, ESIDE submitted logs for 50 individuals. These students had ESIDE running an average of 5.56 days (1-34 day range) and 32 students initiated 352 interactions (hovered or clicked the code warning icon). This averaged to less than one interaction per student per day from the time ESIDE started logging until the last assignment was due. ESIDE saw the greatest amount of use and interaction in the beginning of the study when targeted code was first written. Usage then clustered around due dates in a decreasing fashion.

When comparing the logs of the two studies, Study 1 (assignment-based) students ran and interacted with ESIDE at a greater rate than Study 2 students. A major contributor to this difference is the timing of the deployments. Study 1 was timed to present ESIDE support at the same time that students were building functionality into their applications that had security implications and likely security vulnerabilities. In doing so, there existed a direct connection between concepts and practices.

Students in Study 2 started receiving ESIDE support three weeks into the semester when they began writing placeholder code (e.g., Login page without functionality). This caused an issue with the support in that students had not yet learned how to make their code functional, yet ESIDE wanted to help them learn how to secure it. Thus, when students later added that functionality, and introduced actual vulnerabilities, they had already viewed and dismissed ESIDE as not particularly helpful.

During both studies, ESIDE provided students the option to auto-generate secure code. Five students in Study 1 opted to use this option but only one student decided to include the generated secure code with their assignment. Four students in Study 2 had ESIDE auto-generate secure code yet none allowed the modification to remain in the final product. One student reported that after reviewing the auto-generated code, he was inspired to create his own remediation methods.

Our interaction log review also revealed that some of the students decided to utilize ESIDE for projects outside of NBAD assignments. This interaction ranged from vulnerability identification to content exploration to code generation. Code was not available to review possible changes but the discovery does reveal that students were interested in using ESIDE across the curriculum.

4.3.2 Student Perceptions

Overall, students from both studies appreciated the available ESIDE support and felt as if it had a lot of potential as an educational resource. They reported that ESIDE's icon, menus and content were unobtrusive, easily accessible, simple to navigate, and explained topics in an understandable nature. When asked if ESIDE influenced them to write more secure code, they reported that while they did learn about secure coding, their primary focus was on the completion of a functional project as evident with the following comments:

- "great tool" would "play around with it more" if "I was not so tied up with and concentrating on finishing an assignment" (p14).
- "I have not had a chance to work with ESIDE because my main focus was getting [the] project done with full functionality before the due date" (p16).
- "was always concerned about getting the project done and not necessarily the quality (in regards to security) of the finished project" (p50).

Interviews revealed a shift in students' mindset and a better understanding of their insecure code writing behavior. They reported that ESIDE's vulnerable code icon helped them understand that an issue existed but they didn't always spend the time to find out why. The icon started to serve as an important educator itself as it raised their awareness of behaviors - "I started to be able to predict when the icon would flag my code before I even wrote it" (p9). Students reported that they were starting to think about their own coding practices and how those practices influence the security of their overall application.

Students also shared why they were hesitant to allow ESIDE to modify their functional code. When students acquired code functionality, they felt as if they accomplished something and wanted to move on to the next task. They felt that any alteration would slow them down by breaking functionality or requiring them to learn how to properly utilize the modification. This is understandable considering they were so focused on a finished product.

4.3.3 Pre/Post-test Analysis

Students in Study 1 (assignment) experienced an average pre/post-test increase of 5.0%. Students who had ESIDE running 4 or more days (n=4) experienced a 15.5% gain while those who ran ESIDE less than 4 days (n=4) saw reduced scores (-4.63%).

I collected a total of 18 matching pre/post-tests for Study 2 (semester long). This number is less than expected as, unfortunately, one of the instructors ran too long on the final day of class when I was going to administer the post-test. After dropping two non-related questions, I evaluated both versions of my pre-tests for goodness of fit and found that no significant difference existed between the two using a chi square test χ^2 (9, N=40)=11.03, p=.27. Pre/post-tests that were less than 75% completed were not used in the evaluation. The overall pre/post average increase for all students was 8.3%. Students who had ESIDE running more than five days performed significantly better on the post-test (t(8)=2.48, p=0.03) with a 15.5% increase vs. those who ran ESIDE less than five days only increased by 1.1%.

4.3.4 Limitations

The components of these studies were summative in nature as I collected data concerning student interaction with ESIDE in their own work environment. Un-

Table 1: Test score summary

	$\bar{x}$ Pre	$\bar{x}$ Post	Change
Study 1 (n8)	60.4	65.4	5.0
Runs >= 4 days (n4)	54.4	69.1	14.7*
Runs < 4 days (n4)	66.4	61.8	-4.63
Study 2 (n18)	30.6	38.9	8.3
Runs > 5 days (n9)	27.8	43.3	15.5*
Runs < 5 days (n9)	33.3	34.4	1.1
*Significant. Note – small sample size.			

derstandably, limitations will exist. All participants were enrolled in an advanced Java class and interaction with ESIDE might be different with intermediate or early students over the same time periods. Knowledge of participation in the study and exposure to a new application might have influenced interaction. However, interaction was optional and any initial influence would most likely have leveled out over time. It is also possible that minimal guidance into ESIDE use might have decreased interaction and results but this was intentional as I was interested in how ESIDE could autonomously support the student.

While limitations did exist, I feel that the data is sound. After an evaluation that takes into consideration the aforementioned limitations, I believe the data will continue to support ESIDE's potential for being a positive resource.

4.4 Discussion

The most significant result of my two studies is that ESIDE led to an increase in students' awareness that their code has security implications. Students reported being able to predict when the warnings would occur, and frequently used some of the language from ESIDE's explanations as I talked with them. Students also reported an appreciation of ESIDE's support and explanations. I believe ESIDE is demonstrating

effectiveness in one of my goals - increasing the security mindset of students.

I also found a significant, but small, increase in secure coding knowledge based on the pre- and post-tests. Students who chose to have ESIDE running as little as 4 different days in the short study and 5 different days in the semester study experienced test score gains while the other group saw no noticeable gain. This is encouraging that ESIDE has the potential to serve as an effective means to complement classroom lessons and increase the learning of secure coding practices and principles. However, my sample size was small, as was the gain. In these studies there was no change to the course content or assignments. Thus, the students who saw gain had no incentives for learning and retaining secure coding information beyond self interest. If ESIDE is used to complement and reinforce instruction, rather than solely on its own, I would expect to see additional gains.

Where I did not see any notable effectiveness was in modifying the students' behaviors - to incorporate secure coding practices into their assignments. These two studies revealed two significant lessons that previous researchers did not encounter in their initial laboratory study [83].

4.4.1 Functionality First

Not surprisingly, students in both studies exhibited a task completion mindset that was focused on the core functionality of their code. They were most concerned with just getting things to work. This was related to the goals and content of the course, and rewarded in assignment and exam scores. Thus, once students discovered ESIDE did not help them complete their assignment tasks, their active interaction diminished. As previously mentioned, I did not wish to change the course in any

way for these field evaluations. Thus, student participation was completely voluntary with little external incentive beyond helping out the researchers.

Students also reported a fear of breaking their working functionality if they followed ESIDE's guidance and improved the security of their code. Once they got their code working, they were then unwilling to risk doing anything extra that might change that functionality. Thus, even while students interacted with ESIDE and became cognizant of their insecure coding practices, they were unwilling to fix them and potentially hurt their assignment.

In the previous laboratory study, while these fears were present, they did not impede behavior as much [83]. The participants in the lab study were still concerned with breaking their functionality, but many still tried out adding secure coding to their assignments (and then removed it later if desired). However, students were working in a lab, and getting compensated for their time. They may have been willing to spend a little more time experimenting with ESIDE. They may have experienced more trust in ESIDE or felt more pressure to change behavior with researchers in the room.

As with other programming concepts, the skills of secure coding are important to both be aware of, but also to actually practice and employ. These issues around functionality concerns point to a need for some support and integration of ESIDE's goals more directly into the course. Much like the real world, if security is not valued and rewarded at least in some way, those writing code will spend their limited time and efforts on the aspects that are. I need to do additional research to understand the incentives that will influence and increase such behavior change, while still balancing the primary needs of learning and practicing course content.

4.4.2 Timing

Students in the assignment study used and explored ESIDE at a much greater rate than the semester-long study. This finding is critical as it is related to the importance of timing the use of a tool such as ESIDE. During the assignment study, students were exposed to ESIDE's warnings and educational materials for the first time as they implemented real functionality with real vulnerabilities into their code. This created a direct connection between their practice and ESIDE's available resources. The semester long group first received warnings on placeholder code, which created a situation where the student was not ready to implement ESIDE's support. This, in turn, might have caused a desensitization which led to diminished interaction. Consequently, when ESIDE flagged new coding issues (e.g., sql code introduced later in the semester) the desensitized students were unaware of these new materials because they viewed the icon as a means to reveal content that they had previously explored. Students may also not have realized when they were ready and capable of implementing ESIDE's recommendations, as the warnings did not change throughout the semester.

I had originally envisioned a tool such as ESIDE being always available, as students write code throughout their curriculum. Yet, these results demonstrate the downside of that as students may not return again and again to ESIDE's warnings if they were not able to make use of earlier warnings. This leads to two design lessons for ESIDE. One is to further investigate methods for the tool to determine when is a good time to provide warnings and interaction with students. The second is to

provide more controls for instructors to customize how and when different aspects of ESIDE be introduced as students progress through their coursework. I will also need to investigate how to motivate repeated interactions across multiple courses, as the types of security explanations and code patterns will change as students move from early introductory courses through to advanced courses.

4.5 Conclusion and Future Work

Computer science curriculums need to provide future software developers with secure programming knowledge and skills. I believe my approach can help to fulfill this need by complementing in-class materials with tool support during students' code construction. In doing so, students at all levels will have the opportunity to expand and reinforce their secure programming knowledge. My evaluations provide support that such an approach can be an effective means for providing instructional support in the IDE over longer periods of time. However, deploying ESIDE also revealed needs for additional investigation of how to best incorporate ESIDE into coursework through incentives and timing of the instructional support.

In the long-term, my goal is to explore if ESIDE can enhance secure coding education at all levels across the curriculum. I continue to modify ESIDE to meet the needs of instructors and students from introductory to advanced level classes. Overall, the educational support process will be similar - targeted code is identified, an educational opportunity is made available, and a possible code generation function can be implemented. I am exploring and evaluating methods of tailoring ESIDE support to the students' level and providing additional customization for instructors.

CHAPTER 5: ESIDE EARLY AND INTERMEDIATE STUDIES

The first ESIDE implementation built upon ASIDE, focused on advanced students (in my case a senior- and Master's level web development course) with real vulnerabilities, and providing additional educational support. I simplified the interface, had ESIDE monitor code for instances of behaviors that caused real-world problems, and developed content to align with classroom progression. I then studied ESIDE's use in a single assignment study over the period of 10 days, and a second semester-long deployment study as presented in Chapter 4 on page 42 . I examined interaction over time, ESIDE's influence on security awareness, secure coding behavior changes as well as student impressions [75].

In general, ESIDE helped advanced students become cognizant of their coding behavior, and led to an increase in both security awareness and secure programing knowledge. However, I found several important factors that limited ESIDE's impact, including how to time ESIDE support to be most relevant to the students, and incentives for motivating students to utilize ESIDE. Students were driven by a task completion mindset, as that was the basis of the assignment grades. Once the code was functioning, students feared making changes to improve the security as doing so risked losing points. Thus, the students who used ESIDE were aware of their own insecure coding behaviors, but feared change as they might break functionality. This

result led to me exploring the impact of assignment incentives in one of the studies reported on in this Chapter.

These findings have helped me better understand ESIDE use amongst advanced students, yet early and intermediate students have very different knowledge and thus require very different support. Thus, this Chapter focuses on the studies I performed using the versions of ESIDE customized for these students. The lessons learned will indicate the potential for a tool such as ESIDE to support secure programming across an entire curriculum.

5.1 ESIDE for Early and Intermediate Students

Of particular importance was to create a support system for early and intermediate students that is as simple to use as possible. As students are learning how to apply classroom principles in the IDE, I did not want to run the risk of cognitive overload by providing too much information on the front end of ESIDE's layered interaction. To this end, I designed ESIDE to be easy to setup, require no knowledge of secure coding, simple to navigate, and to be unobtrusive yet effective in helping students identify their own coding behaviors. My vision is a tool that is accepted by the students as an integral component to their educational process.

My current early and intermediate ESIDE prototype was developed for the Eclipse IDE for Java, and similar to the advanced version, works by continuously monitoring the student's code writing process for instances of pattern matches based on a pre-defined set of targeted educational support areas. When ESIDE finds a match, it initiates a layered educational intervention based on the code the student is writing. In this example, the student is attempting to obtain an external username with re-

quest.getParameter("userName"). In doing so, ESIDE places a discrete warning icon (Figure 7 on page 58) in the left margin of the code editor. Hovering over the icon reveals a short message that encourages further interaction (Figure 8 on page 58). When the student clicks the icon, ESIDE generates a context-specific list of educational options (i.e., input validation for request.getParameter("userName");), with a short explanation alongside each option (Figure 9 on page 59). ESIDE also provides an option to access an additional page that provides a more comprehensive explanation of what was discovered, why it is important, and how to integrate the provided principles into coding practices (Figure 10 on page 59). The pages are expandable, showing high level explanations that should only take a typical student about 1 minute to review. From there, students can drill down into topics and examples that are relevant to the code they are working on. ESIDE can also auto-generate remediation code for some types of vulnerabilities (Figure 11 on page 60), in this case the student filtered a string with ESIDE's getValidSafeString method. When code is auto-generated, educational information is also included as comments. I utilized the auto-generated code for intermediate computing students, but not beginner students.

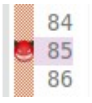

Figure 7: Warning icon

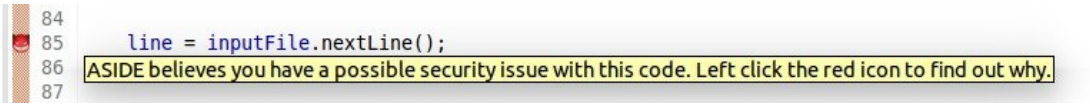

Figure 8: Hover warning

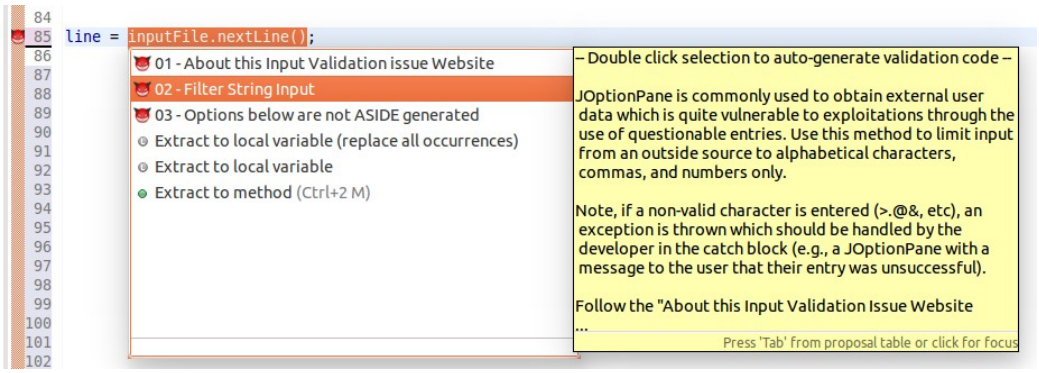

Figure 9: Input validation remediation choices

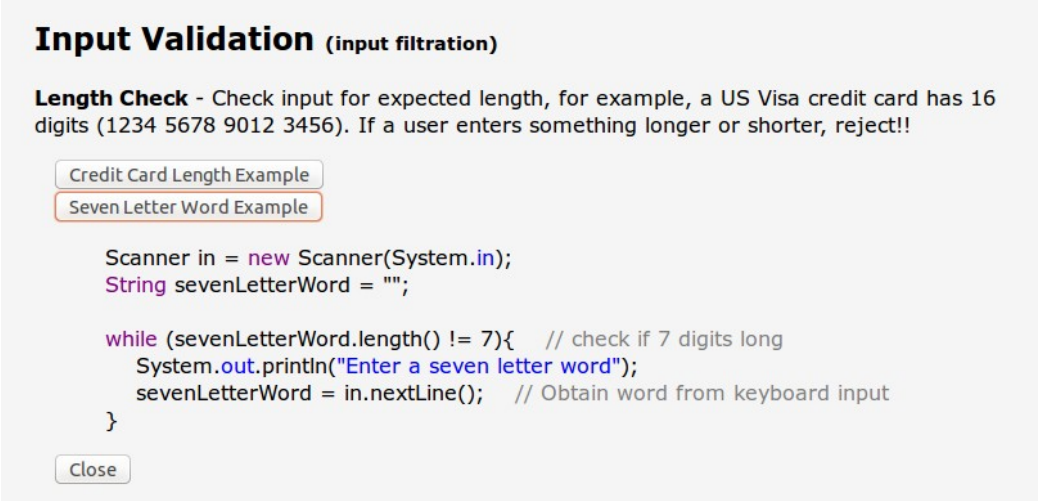

Figure 10: Input validation support page

5.1.1 Contextual Alignment

As the student moves through the instructional process, there are key points where secure coding concepts and practices can complement a course's instructional content. The first points occur with early students as they are learning the basic principles of programming. I identified the below concepts by reviewing syllabi, courses, and faculty/government/industry recommendations [11, 49, 59, 66]. While the following draws from the typical Java learning progression, these interventions are also applicable to other languages.

```
83
84 try {
85     // ESIDE has generated the following code to validate your input
86     line = ESIDE.ValidateInput.getValidSafeString(inputFile.nextLine());
87
88 } catch (Exception e) {
89     // ESIDE found the string not to be valid
90     // As a programmer you should handle this issue in the catch statement
91
92     return null;
93 }
```

Figure 11: Auto-generated remediation code

5.1.1.1 Early Student Learning Concepts and ESIDE Content

Early students develop a foundation of coding knowledge by mastering the principles of variables, objects, assignment statements, conditional statements, loops, i/o, arrays, and functions. Secure coding lessons should complement this process in two stages. The first stage is during the period where students control all of their data and have no external data entering their code. At this point, it is important for them to learn the following:

- Data types: Students should know the limits / ranges of each data type (e.g., byte = -128 to +127, short = -3,768 to + 3,767) and the problems that could arise if the wrong data type is used (e.g., expecting a number and using a letter).
- Data reasonableness: Students should learn to reflect on the type and size of data they are using and what effect it might have on their code (e.g., a very large number in a loop).

The commencement of the second stage begins with the introduction of external data into the student's code (e.g., console input, JOptionPane). At this point, it is critical that the student begins to develop their security mindset by reflecting on what might happen to their program if they receive unexpected input (e.g., strings instead of

numbers). Once they have considered possible problems, they should learn how to validate obtained data. The following is a list of input validation points that students should learn how to implement at this stage:

- Data type checks: Know how to verify that numbers are numeric characters and names are alphanumeric.
- Bounds check: Know how to ensure that input falls within a certain range or size (e.g., age should be between 0 and 150 or the length of a name should be between 0 and 100 characters).

With consideration to these learning points, I developed ESIDE to initiate educational interventions by searching for code patterns they would typically use during this stage, based upon the assignments and code utilized in the courses I evaluated. The following is a list of what ESIDE searched for whilst monitoring the code writing process:

- Scanner Class
  - next, nextLine, nextInt, nextDouble, nextLong, nextShort, nextFloat, nextBigDecimal, nextBigInteger
- JOptionPane Class
  - showInputDialog, showMessageDialog
- InputStream Class
  - read

I then aligned the security concepts content directly to the written code to facilitate the connection process between classroom principles and secure coding practices. For example, as students worked on an assignment where they obtained a person's grade on an exam with nextDouble(), ESIDE provided support on the importance of bounds

checks and how they should be implemented when obtaining numbers from users. The process was reflective in nature that facilitated the thought process of how unexpected input can cause problems.

5.1.1.2 Intermediate Student Learning Concepts and ESIDE Content

Students at the intermediate level expand their knowledge base by learning how to apply such things as exception handling, recursion, unit testing, application programming interfaces, hash maps, and simple graphical user interfaces. During this process, students should continue to consider potential problems with unexpected input while they learn how to apply the following security concepts:

- Input validation: Check that the data is of the correct type, it is in a reasonable range and of reasonable length.
- File management: Validate supplied filenames, ensure file paths are correct.
- Error handling: Implement a means that recovers from errors (e.g., try-catch) and logs those errors with pertinent information (e.g., data is out of bounds).
- Output Encoding: Sanitize all output to entities such as clients, OS commands and data queries (e.g., SQL, XML, and LDAP).

To ensure intermediate students were exposed to both these and the early learning points, I built upon the earlier version and included the following code patterns for the intermediate version.

- File Class
  - File
- Scanner Class
  - nextLine, nextInt, nextDouble, nextLong, nextShort, nextFloat, nextBigDec-

imal, nextBigInteger

- InputStream Class
  - read
- JOptionPane Class
  - showInputDialog, showMessageDialog

The main goal of integrating these security concepts into the intermediate student's learning process is to enhance the security mindset. ESIDE accomplishes this by providing data sanitization concepts which go into greater depth than those available to early students. Thus, ESIDE's instructional pages for the intermediate students provided educational support for three areas: InputValidation, OutputEncoding, and DirectoryTraversal [17, 18, 20]. Intermediate students also learn how to create and interact with various classes. To facilitate this development in a secure manner, I designed the intermediate ESIDE to provide code generation with instructional comments. The educational intent is to help the student begin to understand the process of implementing classes that are specifically written for the purpose of security such as ESAPI (The OWASP Enterprise Security API) [48].

The most important take away during this stage (and others) is that the student identifies potential vulnerable areas and uses problem-solving skills to remediate security issues. By doing so, they are on their way to becoming robust programmers.

5.2 ESIDE Studies

A primary goal of ESIDE is to bring an educational resource to students at all levels. My previous research and development had targeted advanced students and professionals which left the early and intermediate student support space yet to be

explored. Therefore, I built upon lessons learned and knowledge gained from my previous research and modified ESIDE so it could provide level specific support for early and intermediate students. I then conducted formative studies with these populations in an attempt to discover how ESIDE might best support this group. The following is a presentation of these studies and their associated findings.

5.2.1 Studies: Overview

My research and development plan for the newly created ESIDE plugin was designed around an iterative process that used multiple data gathering techniques during the deployment of five formative studies. My intent was to better understand the students at those levels; the usability of ESIDE; its acceptance; student / faculty impressions; how understandable the supportive content is; and how applicable ESIDE support is amongst differing institutions. My overall study goals is to discover answers to the following questions:

- Can I provide a supportive resource that is non-obtrusive and easy to navigate?
- How receptive are students to interacting with ESIDE?
- What are the impressions of ESIDE's support materials? Are they understandable and helpful? Are they aligned with the students' level of academic development?
- What is the student's perception of secure coding both before and after interaction with ESIDE
- What do the students at this level know about secure programming?
- Will students incorporate ESIDE coding support into their code?

Table 2: Study summaries

School / Class	Date	Level	Participants	Female / Male	Study Type
Elon CS1	F12	early	61	25 / 36	Classroom Activity
Elon CS2	Sp13	intermed	22	9 /13	Classroom Activity
JCSU CS1	Sp13	early	5	1 / 4	Focus Group
JCSU CS1	F13	early	4	1 / 3	Walkthrough
Elon CS1	F13	early	57	17 / 40	Assignment

5.2.2 Studies: Activities

Through the course of this exploratory research, I examined ESIDE in four early and one intermediate Java course at Elon University and Johnson C. Smith University (JCSU) in North Carolina. These sites were chosen as UNCC does not use Java or Eclipse in their early courses. Data from two institutions also provides us with the ability to compare results across institutions.

Four of the five studies were conducted as a single session in the labs that students would normally attend. The fifth study was over the period of one assignment where students were free to utilize personal or lab computers. Three instructors from participating institutions assisted with the recruitment process and provided class time for each study. The activities and assignments that the students worked on during the studies were part of the class, but participation in the studies was voluntary. In total, 149 students agreed to participate. I provide an aggregate view of all of these participants below, with details of each individual study in the following sections.

The scenario for each study followed a similar format. An overview was first presented, consent forms were signed, and ESIDE was installed and explained to the students. Then students worked on a class activity (independently or in collaboration). Either during or after that activity, discussions/interviews were facilitated.

Following completion, the written code and interaction logs were automatically collected by ESIDE. A portion of the students also completed assessments (pre, post, or both) during the course of the study. Students also completed demographic surveys.

Contextual alignment was an important goal of these studies. Therefore, as previously described above, I initially developed ESIDEs content around Towsons extensive research materials concerning the integration of level specific secure coding instruction [15, 27, 28, 61, 62, 63]. I then collaborated with the participating instructors in an attempt to identify the best entry point for a study given the classroom concepts and corresponding secure coding concepts. It was determined that later in the semester was best for the early students, as they would already have implemented basic i/o, loops, and arrays which carry with them the concepts of bounds checks, type checks, and character checks [36, 61]. The entry point for the intermediate students was the middle of the semester as they were building on these concepts by exploring how to further interact with external information such as local file access.

Researchers and faculty members from partnering institutions developed three coding activities to be used during this research. Each provided the opportunity for students to implement classroom lessons whilst also providing us the opportunity to complement the classroom lessons with secure coding support. While I attempted to improve upon the ESIDE interaction after each study, the activities themselves stayed consistent throughout this research.

5.2.2.1 Coding Activity 1

The first of the three activities that I designed was to be used in a single setting classroom environment by early students. The activity was a grade averaging program

that used the keyboard and console for data entry and interaction. Students were asked to code in the needed functionality into the program for the following data entry areas: username, the amount of grades to be entered, and each individual numerical grade. Once the data is entered, the program should return a message that contains the username and the grade average. As they implemented basic i/o, loops, and arrays, ESIDE provided warnings alongside their code.

5.2.2.2 Coding Activity 2

The second coding activity was a review-based exercise that provided students with the opportunity to practice their array skills. Students developed a program that would request data from the user, place it into an array, then retrieve specific entries such as lowest or highest. ESIDE provided warnings based on bounds checks and type checks.

5.2.2.3 Coding Activity 3

The third coding activity was a file access activity where students developed a program which took a name from a user, checked it against a female and male list (as found in a separate file), then have the number of occurrences from each list returned to the user. I designed this activity as a small assignment where intermediate students were encouraged to write 100 lines of code prior to participating in the scheduled lab/study time. They then had until the next class meeting time to complete the activity. ESIDE provided input validation warnings based on file access and path traversal, bounds checks, and type checks.

5.2.3 Pre/Post-test Assessments and Survey

The primary purpose of my pre- and post-test assessments was to gain a better understanding of how aware the early and intermediate students are about secure programming concepts related to their respective levels of education. This data would then drive future content available in ESIDE. I created two versions of surveys with 11 similar true/false questions on each (and the intermediate version had an additional multiple choice question). The assessments also provided an opportunity for the students to give qualitative feedback based on their experience with ESIDE. The content of the questions were divided into two areas. The first group of questions targeted awareness of security concepts as related to the overall operation of software programs. The second group of questions focused on awareness of coding techniques used to mitigate potential problems. The following are examples of these questions.

1. Security of software awareness
    (a) The best practice for validating input is to only allow pre-defined safe characters (white list).
    (b) Security problems may arise from unexpected input to your program.
2. Secure code awareness
    (a) If a user entered their age into your program, what should your program do to be considered secure before using the information?
    (b) If a user entered their last name into your program, what should your program do to be considered secure before using the information?

Overall, I maximized the time I spent gathering data from my students throughout my studies and, therefore, piloted the assessments on a limited basis. Results from specific surveys are reported below.

5.2.4 Observational Data

I equipped ESIDE with a logging feature which I used to track when it was running, the amount and type of vulnerabilities found, and students' interactions with ESIDE (e.g., clicking on the ESIDE icon). For some of the studies, ESIDE also submitted the code that students were working on. The study facilitator took notes of observations, discussions, and comments from each session. I also gathered faculty perceptions, when they were present.

5.2.5 Studies: Specific ESIDE Functionality

I designed the early version of ESIDE to complement the two CS1 activities used in my studies (i.e., averaging program and array practice). To accomplish the complement, I had ESIDE search for specific code patterns that students would write during the activity as related to basic i/o, loops and arrays in relation to obtaining user data with the Scanner class. Whenever ESIDE discovered a match, it initiated a supportive interaction with the corresponding security concepts (e.g., bounds checks, type checks and character checks [36, 61]). The content found in the available support included explanations and examples of code that students could cut, paste, and modify if desired. To further support the learning process, I included a specifically written InputValidation class with the assignment. The InputValidation class contained supportive materials in the form of comments that provided an explanation of the available methods. ESIDE provided students the opportunity to

auto-generate code that would filter input for Integers, Doubles and Strings (e.g., ValidateInput.getValidInt(keyboard.hasNextInt(), keyboard);).

The intermediate version of ESIDE was designed around the file access activity where students use the Scanner, File and JUnit classes to check provided names against two separate text file lists. ESIDE's functionality was based on three areas: JOptionPane i/o (e.g., .showInputDialog, .showMessageDialog) and invocation of the File or Scanner class. Support was initiated by ESIDE whenever students wrote code that matched a class invocation or JOptionPane i/o pattern. Output and file access support was based solely on layered content (e.g., pop, educational page). For input, ESIDE provided the layered support as well as the opportunity to auto-generate code that filtered the input based on a provided input validation class (Figure 11 on page 60).

5.3 Results

I conducted five studies over the period of one year with a total of 149 students. For those who reported, 53 were male, 96 were female. ESIDE submitted 79 individual interaction logs and 76 code logs. During this time, I followed an iterative process where lessons learned and knowledge gained from each study was integrated into the next. I first describe each of the individual studies and the unique data gathered from each. I then discuss several overarching results that span these studies.

5.3.1 Study One: Elon CS1 F12

My first study was with a CS1 class at Elon University (F12), and occurred during a 90 minute lab period with 3 different sections of students. The students worked on the averaging activity (sec 5.2.2.1) and interacted with ESIDE as little or as

much as they wanted for approximately 60 minutes. While doing so, I took notes of my observations, answered questions, addressed comments, and discussed their impressions. In total, 61 students participated and 42 of those completed my pre- and post-test assessments in a counterbalanced fashion with about half doing A before using ESIDE and B afterwards, with the other half doing B as a pre-test then A post. At the end of the study, ESIDE submitted 33 interaction and coding logs to a central location. The amount of logs submitted was less than the total number of participants due to a Citrix configuration issue in the lab. The issue was not discovered until after the study.

Based on the interaction logs, ESIDE provided an average of 5 code warning icons per student during this session. Students initiated interaction with these warnings an average of 5.15 times. All of the 33 students interacted with the first two educational layers (icon, popup) and 15 accessed the third educational layer (instructional page). Twenty-two of the students implemented ESIDE support into their code in a manner such as the following.

- String nextLine = ValidateInput.getValidAlphaNumeric (keyboard.nextLine(), keyboard);

During the first of three sections, classroom observations revealed minimal interaction with ESIDE. The instructor explained that this behavior was appropriate and expected due to the complexity of the Eclipse IDE. The instructor further described that Eclipse warnings easily confuse the early students and that the provided explanations are too advanced. The instructor had thus previously advised students to ignore Eclipse warnings and to figure out why they were receiving a warning based

on the materials they had learned in class. For the other 2 sections, I then informed students that ESIDE's content was written to their level. Doing so appeared to relieve aversion to interacting with any Eclipse warnings, and led to more interaction: "better explanation as to what was wrong with code than messages given by eclipse" (p 77) and "the pop-up windows were helpful and wording was understandable" (p 53).

Those who chose to implement ESIDE's code recommendations (22/33 of the participants) appeared to understand why it was important to do so and how the remediations would make their code more secure : "it helped me understand what I was doing wrong" (p 58).

However, a small group of students were not prepared for the lab, and needed to learn the concepts in the first place, rather than practicing them. These students spent time searching ESIDE for the meaning of classroom content, which was not beneficial for them.

5.3.2 Study Two: Elon CS2 Sp13

I conducted the second study with one section of intermediate CS2 students from Elon University. As students worked on the file access (sec 5.2.2.3) activity during their scheduled 90 minute lab time, they were encouraged to interact and discuss their impression of ESIDE. The researcher facilitated discussion by seeding questions and opening up comments to group discussions. At the end of the session, students completed a post-test assessment that contained questions both from Study 1 as well as additional questions related to their activity. In total, 22 participated and 21 completed the post-test assessment. At the end of the lab, ESIDE was unable to

send interaction logs due to an unforeseen conflict with the Citrix lab configurations which was only discovered after the study. However, server logs revealed that 11 students navigated to the third educational layer (instructional page).

During this study, ESIDE introduced the concept of path traversal as related to the accessing files during the lab assignment. Overall, students had difficulty understanding file paths as evident by their many questions/difficulties with providing the correct path to the files they were working were trying to access. When asked how many use or have used paths to navigate their computers only a couple admitted to doing so. It can be postulated that these students are quite visual concerning their interaction with their folders and files and assume that interacting with paths is quite minimal. This was also evident when many of their questions/difficulties with assignment were related to providing the correct path to the files they were working with. The effect was a difficulty in understanding the concept of path traversal.

When ESIDE provided a warning icon, students reported that they felt something was wrong with their code. A common topic in response to the icon was whether or not ESIDE could help make their code "work. This opened up the discussion on the concept of secure/robust vs. functional code. Many at this level had not considered that just as there are many ways to make code work (e.g. for loop vs. while loop), and that there also existed practices to secure code (e.g., bounds check, type check). Similar to the early students, some intermediate students looked to ESIDE to help them make their code functional rather than secure their code.

Two prominent themes based on two groups were observed by the researcher during the lab session. The first came from a group of students whose focus was the

requirements of their project. Most of the their questions were targeted toward the short term gain of "how" to make the code functional rather than the long term understanding of "why" particular code does or does not work. The second theme came from a smaller group who appeared to have a stable grasp of the assignment spent more time on interacting with ESIDE and asked questions as to what it did, how it was accomplished, and why that was important.

5.3.3 Study Three: JCSU CS1 Sp13

I conducted my third study as a focus group with five students from Johnson C. Smith University, over the period of a single lab toward the end of the semester. The group worked collaboratively on the averaging activity (sec 5.2.2.1). I projected the activity on the board and worked through the various requirements together (e.g., obtain a quiz score from the user). Whenever ESIDE support became available, the group discussed the usability of the interaction as well as the understandability of provided content. At the end of the session, three students completed the post-test assessment.

In this study, I discovered an unexpected finding related to my timing of the study. This group of students did not have a basic grasp on console driven i/o concepts, and were thus confused by the ESIDE warnings. The researcher ended up first spending time explaining the Scanner class used in the assignment, before reactions could be gathered on ESIDE. Students felt that a resolution could be to provide an extra instructional layer that explains the classroom concept so that once it was mastered, they could move on to the security component.

5.3.4 Study Four: JCSU CS1 F13

The fourth study was an interview study with four early JCSU students. They worked on the first part of the array activity (sec 5.2.2.2). As ESIDE provided interaction, the researcher questioned them as to their impressions of their interaction and their understanding of the available content. The researcher encouraged them to interact with each layer of support in an attempt to gain a deeper understanding of student understanding of the content provided by ESIDE. This guided interaction started with the warning icon (Figure 7 on page 58) and continued through each layer of available interaction and support. In this way, I gathered more detailed feedback about the language and meaning of all of ESIDE's content.

As the students described their impressions and understandings of ESIDE, I discovered that some of the students did not fully understand the coding concepts that ESIDE was trying to complement. However, once these students clicked on ESIDE's icon and read its first popup about function code vs. secure code, they were able to explain that they should go back and learn how to apply the coding concept at hand prior to continuing interaction with ESIDE. The end result was a reduction in confusion related to student learning expectations and ESIDE.

5.3.5 Study Five: Elon CS1 F13

With my fifth study, I took lessons learned and knowledge gained from my previous single setting studies and performed a week long study over the period of an assignment (sec 5.2.2.2) at Elon University. Two instructors from three sections provided class time to provide an ESIDE and study overview, pre- post-test assessments,

Table 3: Study 5 interaction results

	Interaction logs n	Warning avg.	Interacted at least once	Interaction avg.	Accessed everything
Section 1: Faculty A	16	33.2	12/16 (75.0%)	2.63	5/17 (29.4%)
Section 2: Faculty A	17	22.4	10/17 (58.8%)	1.29	4/17 (23.5%)
Section 3 Faculty B	12	35.8	11/12 (91.7%)	3.00	5/12 (41.6%)
Total	46	29.8	33/46 (71.7%)	2.22	14/46 (30.4%)

and a classroom discussion about their experiences (without the instructors in the room) at the end of the study. My intent was to discover answers to my overall goals (sec 5.2.1) as they relate to a longer deployment of ESIDE. I also explored the how assignment requirements influenced coding behavior. To this end, one of the three sections (Section 1) had a small number of additional security-related requirements (e.g., bounds check) worth 5% of the overall assignment grade.

ESIDE submitted a total of 46 interaction logs. As shown in table 3, Section 1: Instructor A had 17 submissions, Section 2: Instructor A had 17 submissions, and Section 3: Instructor B had 12 submissions. From the logs collected, ESIDE provided an average of 29.2 warnings per student (Figure 9).

Overall, 71.7% of the students interacted with ESIDE to some degree and 30.4% accessed every instructional layer. Interestingly, students in Section 3 had a higher interaction rate overall. This Section also had their midterm exam on the due date of the assignment, thus students could have been using ESIDE as a review mechanism.

In looking at assignment code, and student coding behavior, I found some unexpected results. Both instructors reported that their students were struggling with

Table 4: Study 5 code behavior results

	Assignments collected	Complete assignments	Wrote secure code
Section 1: Instructor A	16	14	11
Section 2: Instructor A	14	11	2
Section 3: Instructor B	12	09	0
Total	43	34	13

arrays and that many had not turned in their assignments. So much so that after my post-tests and post group discussions were complete, the instructors extended the deadline for any student that needed extra time. The instructors also reported that this was unexpected as previous semester students did not exhibit similar behaviors on a similar assignment.

There was a clear pattern between sections when investigating whether students actually implemented a more secure solution in their code. As indicated by their professors, students had difficulty completing this assignment. Of the 43 collected, I found that only 34 had completed at least 90% of the work. Of those 34, 13 had included secure coding practices. Most of which were those who were in Section 1 who had secure coding requirements as part of their assignment. None of those with the mid-term exam on the due date (Section 3) wrote secure code even though their level of interaction was greater than the other sections.

5.3.6 ESIDE Interface

It was important to us to understand if ESIDEs interface is intuitive and easy to use. Through the interviews, focus groups and discussions I discovered that students found the interaction straightforward and simple to navigate. In general, I found that the warning icon did not "really disrupt coding" (p112) or get "in the way" (p86).

However, a small group of students felt that the red color of the warning was a bit aggressive. After testing various colors (e.g., yellow, blue), I went with a darker red which was found to be more acceptable. Students also appreciated the ability to easily "go directly to a page that explained how to help" (p94). Overall, I believe ESIDE's interface provides a simple yet robustly supportive interaction.

5.3.7 Content Impression

A large challenge to complementing an existing course is the alignment of the supportive content. Materials that are too advanced or too early can have a negative effect. I therefore focused on the students' interpretation of ESIDE's content and interaction. All of the students throughout my studies reported that they understood that a problem existed when the warning icon appeared next to their code. As they interacted with the rollovers, popups and educational pages I discovered that about 80

- [p214] - "explained the issue in your code very well"
- [p165] - "the feedback it gives, so the user can get a different perspective into how to do it in a better way"
- [p210] - "it was easy to understand b/c it used practical examples (grades)"
- [p208] - "did a good job explaining what the code would do; explained why it might be beneficial to use"
- [p209] - "the detailed hints on how to make it more secure"
- [p196] - "I liked the suggestions, and how to make the code work "
- [p195] - "I liked that it gave you sample code when you clicked on the icon. That made it very user-friendly"

- [p194]- “explaining problem provides code to fix it”

On the other side of the spectrum, there existed a group of students who had difficulty with the concepts related to the lab assignment (e.g., how to create a while loop, what a file structure looks like). There was a tendency amongst this group to search ESIDE for classroom concept support, the effect of which ranged from a minimal time loss during the search process to further confusion. This confusion is most likely because ESIDE’s explanations assume that the student has a basic grasp of the classroom concepts. Their struggle became evident through observations and comments given as follows:

- [p204] - “i didn’t like that it was hard for me to figure out if my code was right”
- [p112] - “wasn’t really sure as to how it was trying to help me”
- [p98] - “that it did not give examples of how to fix your code”
- [p81] - “I didn’t understand some of what it was saying, confused”
- [p46] - “sometimes the textboxes were a little too long, and there was too much to read”

My remedy was to simplify and shorten the content found in ESIDE’s supportive layers. This included reducing the content in the initial popup box down to a short paragraph and altering informational pages so extensive explanations are only revealed when students explicitly expand sections. In later studies, I found this helped to reduce confusion. For example, I added an explanation regarding code for functionality vs code for security, which appeared to help students understand the difference.

- [p134] - “it shows me that though my code will run with proper user inputs, its still vulnerable”

- [p68] - "that even if there was not an issue it still provided information that may lead to problems in the program"
- [p161] - "the text boxes did a good job explaining why something was a threat."

5.3.8 Piloted Pre/Post-test Results

During these five studies, I piloted pre and post tests in multiple fashions during four of the studies in an attempt to better understand the the level of security awareness students possess at various stages and institutions. This endeavor took into consideration the complexity of assessing secure coding knowledge as general questions are easily guessed as common sense and specific code related questions are reliant on the understanding of how the code works prior to understanding security implications. To this end, I designed the two similar versions of tests based off of the classroom concepts that the students should have known by the time they took the survey and the complementing materials available through ESIDE.

As previously noted, the pre and post tests consisted of a secure code writing section and an overall software security awareness section. Out of my first five studies, I discovered that the average of all V1/V2 pre/post was 63.2%. Their average secure coding awareness score was 54.7% and their average software security awareness score was 68.9%. Overall, the early student scores remained about the same for the pre and post surveys. This was expected as the time they had to learn from ESIDEs content during the lab was minimal and the pre-post test questions were not repeated. The intermediate students performed about the same as the early students with regard to secure coding awareness but their software security awareness was about 10% higher. This could be expected as intermediate students would most likely have been exposed

Table 5: Studies 1-3 awareness results

	n	V1 pre (code/concept)	V2 pre (code/concept)	V1 post (code/concept)	V2 post (code/concept)
Elon CS1: F12 Grp1	22	64.05% (75.97%/43.18%)			61.98% (60.39%/64.77%)
Elon CS1: F12 Grp2	21		67.10% (65.31%/70.24%)	68.40% (79.59%/48.81%)	
Elon CS2: Sp13 Grp1	9			53.54% (51.39%/59.26%)	
Elon CS2: Sp13 Grp2	9				63.64% (59.72%/74.07%)
JCSU CS1: Sp13 Grp1	2			54.55% (56.25%/50.0%)	
Elon CS2: Sp13 Grp2	1				63.64% (62.5%/66.67%)
Totals	64	64.05% (75.97%/43.18%)	67.1% (65.31%/70.24%)	63.35% (72.32%/47.66%)	62.5% (60.71%/65.63%)

software security topics at this point but not necessarily secure coding topics.

Students in the fifth study (assignment) were divided into three sections. The first had a minimal amount of secure code requirements on their assignment. This Section (n=11) experienced an average pre/post-test increase of 10.7% and with a secure coding awareness increase of 14.3% and a software security awareness increase of 4.5%. Section 2 (n=15) experienced an overall increase of 1.9% with a 2.0% secure coding awareness increase and a 1.8% software security awareness increase. Section 3 (n=9) had a midterm based off of the activity the same day it was due and experienced an overall increase of 11.1% with a 14.3% increase in secure coding awareness and a 5.6% increase in software security awareness.

It should also be noted that Section 1 and Section 3 experienced significant gains (p=.05) for overall increases and secure code awareness. However, my pre/post-tests are still in the early stages of development and due to the small numbers, results may vary as more data is collected.

5.4 Limitations

These studies were primarily qualitative explorations that encouraged students to interact with ESIDE and provide feedback in a group setting. Because of this envi-

Table 6: Study 5 awareness results

	n	V1 pre (code/concept)	V2 pre (code/concept)	V1 post (code/concept)	V2 post (code/concept)	Gain
Elon CS1: F13 Sec1	11		62.81% (62.34%/63.64%)		73.55% (76.63%*/68.19%)	10.74%* (14.29%*/4.55%)
Elon CS1: F13 Sec2	15	69.48% (80.61%/50.0%)		71.43% (82.65%/51.79%)		1.95% (2.04%/1.79%)
Elon CS2: F13 Sec3	9		67.68% (66.67%/69.44%)		78.79% (80.95%/75.0%)	11.11%* (14.29%*/5.56%)
*Significant at the p <0.05 level.						

ronment, it is appropriate to be cognizant of possible limitations. The first limitation is the influence of the instructor and researchers in the room. While I believe the interaction was positive, this presence might have influenced the student interaction in either a positive ("I like xyz") to negative (reduced interaction). A second limitation is the possible influence the group had on individual responses to group questions as many of the responses were positive and critical responses were minimal in comparison. A third limitation is the short exposure (less than 60 minutes) most of the students had with ESIDE, which means I will have to base feedback on initial impressions.

Another possible limitation is that Eclipse is a complex IDE that is not typical of entry-level Java courses (although was used in all of the courses I examined). While this produced a cleaner interaction, any deployment in a non-Eclipse environment would require substantial effort to port to other environments. Given these limitations, I believe my results provide support that ESIDE has the potential to influence secure coding practices throughout the semester and provides valuable lessons for improving for longer and more general studies.

5.5 Discussion

I designed ESIDE to help students make the connections between classroom concepts and secure coding practices. My results do indicate that this approach has promise, as many students responded positively to ESIDE. However, I also uncovered several challenges and issues in providing support for secure programming alongside students who are still learning and practicing core course content.

5.5.1 Timing

A critical component of ESIDE is the assumption that students have a basic understanding of the classroom concepts prior to complementing them with secure coding information. Because of this, it is imperative that ESIDE does not introduce new concepts too early as it might have a negative impact on the learning process and future ESIDE interaction. To this end, I designed my ESIDE studies to be timed with expected student competency based on course progression.

I found two distinct areas that will need to be explored in future studies. The first was the difference between institutions. Using basic i/o functions as an example, one university went into a greater depth with this topic and spent more time having their students write code for these functions. The effect was a group of students who understood how to apply ESIDE's code related support. The second university did not spend as much time with these functions with the expectation that their experience writing these functions would be gained over time. These students conceptually understood ESIDE's support but they were not really ready to apply its lessons as they really did not have a handle on how to write i/o functions. The result was

them exploring ESIDE for examples on how to make their code function. Over time, this could have a desensitizing effect as answers to their questions are not available through ESIDE.

The second area related to timing was student preparedness. There existed a small group of students who used lab time to learn classroom lessons rather than apply those lessons. Similarly, they explored ESIDE in search for class related instructional materials as opposed to the available security related materials. While they were not necessarily confused by the available materials, they did not find the functional support they were looking for.

Initially, I conceptualized ESIDE to be continuously available to students as they learn to code over the course of their education. However, these findings reveal the potential of abandonment as students might view ESIDE as being non-helpful. In light of this new knowledge, I will need to explore how to control for timing. The resolution could be from one of three design areas. The first is to have ESIDE ascertain the appropriate time to initiate interaction. The second is to provide the instructor with the ability to control when and what ESIDE provides the students and the third is a means for students to assess when they are ready for support.

5.5.2 Student Awareness and Use

Overall, students reported that they liked the concept of ESIDE and supported the idea of having it available to them throughout their education. They felt as if the warnings were non-intrusive reminders of code issues and that they should investigate as to why they were receiving a warning. As they explored for an explanation, they felt as if the navigation was simple and straightforward due to the ease of finding

answers to their inquiries.

Most students at these levels reported that they were unaware of the security issues related to the code they were writing. After working with ESIDE for just a short time, they became more aware of the larger security picture "it shows me that though my code will run with proper user inputs, its still vulnerable" (p134). Becoming aware of potential issues is the first step in helping students develop a security mindset.

ESIDE has the capability of auto-generating secure code for known issues. This feature was available during the intermediate student study. While logged data was lost, students reported using this feature and appreciating its availability. This is quite promising as students were willing to secure their code. However, such a feature runs the risk of losing an educational moment as those who are task oriented might have the tendency to auto-generate code without reading the supportive content.

Eclipse can be difficult to use and understand as it provides warnings that are too advanced for the students at these levels. Faculty have remediated this difficulty by conditioning their students to ignore available warnings. Despite this conditioning, students actively engaged with ESIDE's warnings. To ensure future engagement, I modified the language used in the initial popups so it could be read in a simplistic and inviting manner (e.g., "ESIDE believes you have a possible security issue with this code. Left click the red icon to find out why.").

5.5.3 Motivation: Incentives to Secure Coding

As can be expected from CS students, most participants revealed a task completion mindset that was primarily focused on the functionality of their code [35]. This is understandable as students are motivated to complete the only the requirements

of their coding activities because successful completion is typically rewarded with higher assignment and exam grades. To this end, I tested if minimal secure coding requirements on a week long assignment would influence the coding process and the students awareness. What I found is that students who had no incentive to learn about or write secure code (Section 2) performed considerably less than their counterparts in the other two sections. Their interaction with ESIDE and their awareness scores were noticeably lower than the other two sections.

The evident motivator for Section 1 was the requirement to write secure code as part of their assignment. The higher level of interaction over Section 2 can be attributed to this incentive. This requirement also had an effect on coding behavior as 11 out of the 14 completed assignments included secure coding. In addition, their secure coding awareness scores greatly increased while their software security awareness only had a minimal increase. This finding relates back to functionality. To expand, the initial layers of ESIDE support are related to the identification and remediation of secure code issues while the last layer goes into depth why this is an issue which is where most of the software security awareness information is found. So, students spent time learning about the practices needed to make their code work as required and then they moved on.

Section 3 interacted with ESIDE at a level similar to Section 1 (points incentive) and also experienced a large increase in their secure coding awareness scores. I have attributed this finding to an unforeseen incentive which can be attributed to students using the assignment to study for the midterm exam which occurred the same day the assignment was due. When the students were provided with the assignment,

the instructor informed them of its purpose of being a review activity. To this end, I postulate that they spent more time exploring ESIDE for information concerning classroom concepts. Once they did not find said information, they moved on. This behavior is supported in their post-test scores as their secure coding awareness increased at a much greater rate than their software security awareness scores which would have only dramatically increased if they had spent time exploring the final layer of ESIDE's support.

These findings reveal the importance of the instructor's role in guiding the student's education. While I designed ESIDE to be as autonomous as possible, it appears as if faculty will need to highlight the importance of secure coding by incentivizing its understanding and use.

5.6 Conclusion

It is imperative that computer science programs provide their students with the knowledge and skills necessary to develop and maintain secure software. Unfortunately, too many students are graduating without the ability to accomplish such a task. To help students learn secure coding techniques and fulfill the need for adept software developers, I present ESIDE. The ESIDE plugin moves secure coding instruction into the IDE where connections between classroom principles and secure coding practices can be made while the student is "in the moment". ESIDE's provided instruction complements classroom progression by identifying code patterns that directly relate to lessons being taught. The effect is a means to develop and expand upon the students' secure programming knowledge base.

My studies show promise for my ESIDE approach. Throughout this paper I have

presented experimentation and shared experiences that demonstrate the usefulness and effectiveness of ESIDE support. Of particular importance, students appreciated the support ESIDE offered, found interaction simple and straightforward, understood its content and overall meaning, experienced an increase in secure coding awareness, and wrote secure code.

5.6.1 Moving Forward / Design

These short term studies provided positive evidence that my approach with ESIDE can work. While this is promising, many more questions have yet to be answered about whether or not ESIDE can serve as an effective resource at multiple points and durations throughout the curriculum. For example, will students continue to access ESIDE throughout the term? Will students become habituated to ESIDE and eventually abandon its support? Will students incorporate ESIDE lessons into classroom practiced over time? I intend to answer these questions in my future experimental studies.

Future design features of ESIDE include the following. First is a means to control for the timing of when students are exposed to ESIDE content. Second is a method for instructors to customize content to be used in cases where they want to directly map classroom materials to ESIDE support (e.g., info on xyz can be found on page xx of your book). Third, I intend to modify the icon from a single indicator for all issues to multiple indicators based on the category of the identified code. Finally, I intend to continue tool development based on feedback from my studies and evaluations.

CHAPTER 6: DESIGN GUIDELINES FOR ESIDE

My research focus has been to better understand if ESIDE could positively influence a student's secure coding mindset (RQ1), and if ESIDE could motivate and incentivize students to learn about (RQ2) and implement (RQ3) secure programming practices. Of primary concern is the influence that ESIDE's interface has on the findings to these questions. To this end, I designed ESIDE to be engaging and understandable by implementing a simple, un-obtrusive, and easy to navigate interaction that provides appropriate student level content. In chapters 2 and 3, I presented student needs at each level and ESIDE's interface. During this chapter, I will present student interactions and design implications that I discovered throughout my research. I also discuss faculty impressions, suggestions, and implications on the current design of ESIDE. I finish with a conclusion of this work and future directions.

6.1 Early Student Considerations

To review, early students are just learning the basics of i/o, loops, and arrays. As these principles will serve as the foundation for future coding development, the student should also concurrently learn how to implement the corresponding secure coding concepts (e.g., bounds checks, type checks, and character checks). ESIDE complements this foundational development period by providing targeted interaction for the student. The point of entry for this interaction is during the period where the

student starts to implement console level input and output calls. This developmental period is crucial as it is the first time that students introduce external data into their program.

6.1.1 Early Findings and Design Implications

Students found the warning icon noticeable yet non-intrusive and that they understood the contextual meaning of rollovers, popups and the educational page. Students also indicated that they understood that a problem existed when the warning icon appeared next to their code. As the students began further interaction, I discovered a trend where students who understood (about 90%) how to implement the requirements of the assignment found the ESIDE's messages understandable to the point of altering their own code. This interaction actually led students to ask specific questions about secure coding, which led to a group discussion amongst the students, faculty and the researcher.

On the other side of the spectrum, there existed a few students who had difficulty with the lab assignment (e.g., how to create a while loop). These students reported that the assignment confused them and when they read ESIDE's information, they became more confused because it did not help them understand the assignment. This confusion indicates that the timing of ESIDE exposure was too soon for these students. A simple resolution that reduced confusion was to provide explicit language throughout the student interaction that addresses the difference between functional and secure. The message was that security is on top of functionality and students should first understand how to implement functionality before proceeding. A second resolution to be discussed in the faculty Section (6.4 on page 96) is providing faculty

with the ability to control when and what students are exposed to during the semester.

6.2 Intermediate Student Considerations

To review, students at the intermediate level are learning how to develop basic Java driven applications. They have learned the fundamentals of Java and are learning how to make their applications interact with multiple external entities. This interaction typically comes from communication with flat files, user input from GUIs and output to those GUIs.

During this developmental period, ESIDE expands the student's knowledge base and mindset by providing data sterilization materials. Based on input validation and output encoding, these materials go into greater depth than those available to early students.

Also during this period, students learn how to create and interact with various classes. To facilitate this development in a secure manner, ESIDE also provides code generation with instructional comments. The educational intent is to help the student begin to understand the process of implementing classes that are specifically written for the purpose of security such as ESAPI (The OWASP Enterprise Security API) [48].

6.2.1 Intermediate Findings and Design Implications

A few students wanted to know more about regular expressions, which I did not present upon or offer information in the touchpoints for simplicity sake. Future material on the educational pages will need to include this material in a non-confusing manner in a layered fashion.

Overall, the intermediate students found the warning icon noticeable yet non-

intrusive on their workspace. The students expressed an understanding that something bad could happen if an external source entered harmful characters into a program. The educational materials based on Path Traversal were somewhat difficult for many of the students to understand. As I explored this topic with the students, I discovered that most navigated their computers with icons and never really paid attention to the directory structure (some operating systems do not make this obvious). As I discussed directory structure and potential problems, the students did grasp the concept. The design implication is that available content must ensure that students understand a classroom principle prior to introducing a secure coding concept.

Similar to early student findings, there was the interplay between a student's understanding of the course lessons (e.g., access a file in a directory) and that of ESIDE's secure coding concepts (e.g., path traversal vulnerabilities). It appeared as if some students were confused on how secure coding concepts related to functionality. For these students, it was necessary to explain that if their code functioned then the addition of ESIDE's suggested practices would not break functionality yet if their code did not work, it would not work if they applied code shared by ESIDE. Design resolutions for this confusion are the same as the early student - further explanations on functionality and the ability for the instructor to control the timing of available ESIDE resources.

6.3 Advanced Student Considerations

Students at the advanced level have mastered the basics and intermediates of Java. Their programming is in the process of becoming more intricate and technical as they learn to implement robust interactions in their web applications. Not only does this

implementation help the student develop advanced Java skills and techniques, but it also helps them come to understand the process needed to store and retrieve database information using SQL communications.

6.3.1 Advanced ESIDE Interaction

During this technical developmental period, students are just below the introductory professional level. Therefore, ESIDE provides a full assortment of support features that range from auto-generated data filtration code (Figure 12 on page 93) to output encoding remediation to prepared SQL statements. Because the students are developing code that potentially has real-world vulnerabilities, ESIDE provides information on real world attacks like Cross Site Scripting (CSS). This exposure continues to expand the student's security mindset by conveying the need to consider potential issues at every level.

```
    /* NOTE: If the following ASIDE generated code detects a problem
     * (e.g., malicious characters entered by user) an exception is thrown.
     * Doing so will skip the rest of the try block code and go directly to
     * (execute) one of the generated catch blocks below.
     * */
    String SubmittedUserName = ESAPI.validator().getValidInput("replace ME with validation context",
            request.getParameter("userName"), "SafeString", 200, false);

    return macAddress;
} catch (ValidationException e) {
    /* This catch block is executed when ASIDE finds input that did
     * not match validation rules (e.g., bad user input). When this happens,
     * the developer should maintain usability in this catch block
     * (e.g., response.sendRedirect("Login.jsp"); with an accompanying
     * unsuccessful attempt indication presented to the user).
     * */

     // Note: Default return generated by ASIDE
    return null;
} catch (IntrusionException e) {
    /* This catch block will be executed when advanced
     * intrusion behavior is detected in ASIDE's try block
     * statement. This exception should also be handled by the
     * developer similar to the ValidationException block.
     * */

     // NOTE: default return generated by ASIDE
    return null;
}
```

Figure 12: Advanced generated code

6.3.2 Advanced Content

As with most students, advanced students are concerned about completing the assignment in the given time. In an attempt to make the interaction as fast and simple as possible, I reduced the amount of remediation choices available to the student (Figure 13 on page 94) from the amount available in previous ASIDE studies [82, 83]. I also included a few remediation options that did not directly relate to any course coding in an attempt to expose students to the larger picture of secure coding and, consequently, help them develop a more robust mindset.

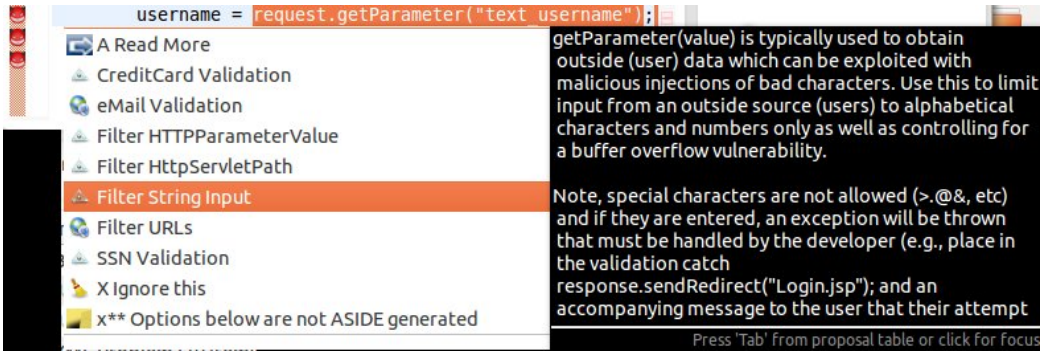

Figure 13: Advanced ESIDE remediation options

I also expanded the codeGen instructional materials. To review, codeGen takes the students code and modifies it in a secure manner. When this occurs, comments are included to explain the purpose of each piece of code generated. These comments also use the generated try and catch statements as an instructional opportunity by explaining the reason behind their presence and the need to handle the catch statements. In doing so, the codeGen materials provide students with the opportunity to either learn something new or reinforce their knowledge base.

Previous ASIDE research revealed the need to simplify materials [82, 83] for ad-

vanced users. To this end, I developed two instructional pages [19, 21] in a layered fashion where messages were simplistic, contextual and straightforward. Examples were written to complement the code used in advanced java application development courses (Figure 14 on page 95) as well as having specific points highlighted.

The following submitted email address is vulnerable and needs to be validated through a whitelist filter.

```
String eMail = request.getParameter("Submitted_eMail");
```

ASIDE takes this code, implements validation routines provided by ESAPI (open source library for secure applications), and surrounds it with a try catch.

```
try {
        String eMail = ESAPI.validator().getValidInput("replace ME with validation context",
                        request.getParameter("Submitted_eMail"), "Email", 200, false);
} catch (ValidationException e) {
        return;
} catch (IntrusionException e) {
        return;
}
```

Figure 14: Advanced code example

In addition, examples became available only after the student expanded the example option. In this manner, students can explore information that is relevant to them and not be overwhelmed with too much information.

6.3.3 Advanced Findings and Design Implications

Students reported a drive to make their application functional and feared that any code change might cause their application to break. However, with time they became more aware of their own vulnerable code practices even if they chose not to alter their vulnerable code writing behavior. To overcome this issue, an educational component will be necessary to explicitly address concerns about how the incorporation of secure coding will not cause an application to break.

Second, some students are very task oriented and want to be efficient with their time

spent on getting their application to function correctly. To that end, future interaction design and materials need to be simple and to the point, so that the student who feels the time pressure of completion will not stray away from the educational opportunity.

6.4 Faculty Input

I had designed ESIDE to be as autonomous as possible in an attempt to minimize any major commitment by faculty members. Over the course of this research, I worked with faculty on the development and deployment of the ESIDE plugin. During the time we worked together, we discussed their impression of the ESIDE plugin. Overall, they felt as if they deployment was simple and straightforward as the only requirement for use was to place the pair of ESIDE plugins in a single Eclipse folder. As the available content for the student interaction was influenced by faculty input, they found the interaction appropriate and acceptable.

The following are faculty recommendations and rationales for improvement:

1. Move ESIDE's menu to Eclipse's main menu bar - the need to right click the project then find ESIDE is cumbersome and relies too much on recall.
2. Consolidate the different versions of ESIDE (i.e., early, intermediate, advanced) into one plugin - distribution would be easier and students could chose their own level based on their need and abilities.
3. Provide a means to customize what code patterns ESIDE searches for - this would allow faculty to control what and when students are exposed to support.

ESIDE can be improved by moving away from the autonomous model and including faculty with the availability of ESIDE support. By providing this option, the students would receive a consistent teaching style throughout the semester and only be exposed

to new concepts when the instructor felt they were ready for them.

6.5 ESIDE Design Developments

Findings from user studies and faculty input have revealed a few areas that ESIDE can be improved. As discovered from the advanced studies (Section 4.4.2 on page 54), the icon is the same for each potential vulnerability category. There exists no means for the user to know if or when supportive content has changed based on new vulnerability identifications. The effect is a desensitization by the student because they believe they have already reviewed available information. To help resolve this issue, ESIDE now uses three different icons to indicate vulnerability issues with input validation, output encoding, and SQL database communications.

Earlier versions of ESIDE required the student to remember where the ESIDE menu was located (i.e., right click the project > scroll down to ESIDE menu). To simplify menu access, I placed the ESIDE menu on the main menu bar of Eclipse. The result is a placement that facilitates recognition over recall.

Over the course of these studies, I had created three versions of ESIDE. In the short term, different versions were manageable. To continue offering three versions would be burdensome to manage by the faculty and the students. In resolution of this issue, I have consolidated all versions into one ESIDE plugin. Depending on the student's and/or faculty's need, each level of ESIDE can be turned on or off from within the ESIDE menu.

The timing of exposure to ESIDE was found to be critical (Section 4.4.2 on page 54 and Section 5.5.1 on page 83). My original intent was to have ESIDE be as autonomous as possible. I discovered that ESIDE could be much more effective if the

instructor could control when and what information was available to the student. To this end, I have included the feature for students to load external XML rules files either locally or by URL. By doing so, the instructor can push out supportive files when students are ready for them (Figure 15 on page 98).

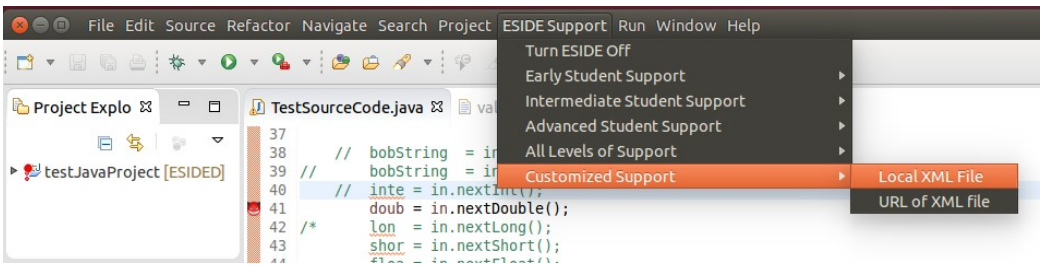

Figure 15: ESIDE external XML file

6.6 Conclusion and Future Direction for ESIDE

During the development of ESIDE's resources, I kept in mind that one of my goals is to enhance secure programming in education. My focus was to create a resource that increases exposure to secure coding instruction at the appropriate academic level, engages students and does not detract from instructor time nor course progression. Every iteration of ESIDE moves my approach to secure code instruction toward my overall goal.

For the early and intermediate versions, it was important to address available content and include material that helped the student understand functionality vs. secure. Such a content change was also needed for the advanced student as they were concerned that ESIDE might break the functionality of their application. After learning how focused students were on completing their project on time, I also streamlined supportive materials by including fewer menu items and java script driven help pages that are easily scanned for information.

Of importance was the development of a means to indicate different and new vulnerable code issues. This has been resolved with different icons for different categories of vulnerabilities. Future iterations could include more specific indications. All of which will need to be researched with user studies.

The timing of exposure plays a critical part in the teaching and learning of new materials. Students who were not ready to learn ESIDE materials could potentially become more confused with their respective classroom lessons. To this end, I added feature to ESIDE that allows students to locally or remotely load XML files that their faculty provides to them. In this manner, the faculty can control when their students become exposed to new materials.

One large advantage of the ESIDE approach is its ability to instruct the students in their own work environment on their own schedule. The success of the ESIDE approach is reliant on how well the supportive materials are aligned with the learning process. I believe that each iteration of ESIDE moves toward a better secure coding educational experience for students.

CHAPTER 7: DISCUSSION AND CONCLUSION

In this chapter I will summarize key findings within this research, highlight primary contributions, inform theory, and put forth implications for design of IDE-based instruction.

7.1 Research Summary and Contributions

My intention for this dissertation was to explore the effectiveness of the ESIDE approach throughout the academic experience. At each academic level, I examined the receptiveness to ESIDE support, the appropriateness of the interaction with ESIDE materials, and the effect of knowledge gained and practices implemented due to said interactions.

I proposed the following research questions at the start of this dissertation:

1. Can ESIDE positively influence a student's secure coding mindset?
2. Can ESIDE motivate and incentivize students to learn about secure programming (vulnerabilities, coding, etc)?
3. Can ESIDE motivate and incentivize students to implement secure programming practices?

I primarily addressed the research questions in Chapters 4 and 5.

7.1.1 Chapter 4 Summary and Contributions

In Chapter 4, I presented two advanced student field studies and related findings. The first study was over the period of a 14 day assignment and the second was over a whole semester. The semester long study also included a focus group. Data collected was drawn from interviews, a focus group, interaction logs, code written, and pre-post-test data. Key findings from Chapter 4 include:

- The ESIDE icon, menus and content were found to be unobtrusive, easily accessible, simple to navigate, and explained topics in an understandable nature.
- Interviews revealed a shift in students' mindset and a better understanding of their insecure code writing behavior. The icon started to serve as an important educator itself as it raised student awareness of behaviors. Students reported that they were starting to think about their own coding practices and how those practices influence the security of their overall application.
- Students also shared why they were hesitant to allow ESIDE to modify their functional code. When students acquired code functionality, they felt as if they accomplished something and wanted to move on to the next task. They felt that any alteration would slow them down by breaking functionality or requiring them to learn how to properly utilize the modification.
- Students who had ESIDE running more than five days in the semester study and four days in the assignment study performed significantly better on the post-test than those who chose to interact less frequently.

A major implication of the two advanced student studies is that ESIDE led to an increase in students' awareness that their code has security implications. Students reported being able to predict when the warnings would occur, and frequently used some of the language from ESIDE's explanations as I talked with them. These findings support ESIDE's effectiveness in increasing the security mindset of students (RQ1).

I also found a significant, but small, increase in secure coding knowledge based on the pre- and post-tests. Students who chose to have ESIDE running as little as 4 different days in the short study and 5 different days in the semester study experienced test score gains while the other group saw no noticeable gain. This is encouraging as there was no requirement to utilize ESIDE beyond personal interest so ESIDE served as its own means to motivate and incentivize students to learn about secure programming (RQ2).

The previous finding is promising as ESIDE autonomously had a positive effect on secure coding knowledge gained. It is possible for gains to be much higher if instructors implemented ESIDE as part of their instructional process. To this end, future studies will need to be developed that explore the educational impact that ESIDE has when faculty actively support and require its use in their courses.

Where I did not see any notable effectiveness was in modifying the students' behaviors - to incorporate secure coding practices into their assignments (RQ3). These two studies revealed two significant lessons that previous researchers did not encounter in their initial laboratory study [83]. Major contributors as to why students did not modify their coding behaviors include a functionality first mindset and the timing of exposure to ESIDE support.

7.1.1.1 Advanced Students - Functionality First

Students in both of the advanced studies maintained a task completion mindset that was primarily concerned with the core functionality of their code. They were motivated by the goals of the course which rewarded them with points as based on functionality. Consequently, when they realized that ESIDE did not create or enhance functionality, active interaction diminished.

Students found functionality so important that they feared ESIDE's recommended secure code alterations as they might cause a break in functionality. Once they got their code working, they felt it was too risky to attempt something that would not offer any point-based gain. Consequently, even after students became aware of their own insecure coding behaviors, they would not modify them to create a more secure application.

As with other programming concepts, it is important to understand both the concepts of secure programming but also have the ability to practice and employ those concepts. The current points driven model that rewards functionality makes it difficulty to have students independently modify their coding behaviors. To overcome this barrier, ESIDE will need to be integrated more directly into course goals if students are expected to change their coding behavior. Much like the real world, if security is not valued and rewarded at least in some way, those writing code will spend their limited time and efforts on the aspects that are.

7.1.1.2 Timing Exposure Amongst Advanced Students

Students in the assignment study interacted with ESIDE at a rate that was much higher than the students in the semester-long study. This finding is paramount to the success of a resource such as ESIDE. A major difference between the studies was when students were first exposed to ESIDE. During the assignment study, students were exposed to ESIDE during the same period that they were implementing the types of functionality that caused real world vulnerabilities. In doing so, there was a direct connection between the principles and practices that ESIDE provided education on and the student's code.

The semester long group were first exposed to ESIDE when their early semester place-holder code triggered an ESIDE interaction. Interaction with ESIDE revealed support for future coding concepts and practices which created a disconnect between ESIDE support and student coding practices. The result of which might have caused a desensitization which potentially diminished interaction. Consequently, when students introduced new coding vulnerabilities later in the semester (e.g., sql code) they were unaware that a different layer of support was available as the icon looked the same as it did when they originally explored ESIDE in the beginning of the semester. Therefore, the timing of ESIDE exposure was too soon in the semester study which caused students to be unaware as to when they were capable of understanding and implementing ESIDE's support.

Results indicate that students might become desensitized to a tool that is always available and, at times, unuseful. They will not return again and again to check if

the supportive content has changed. This finding has led to two important design lessons. First is to introduce ESIDE support to the student at the time it will be most effective. Second is to provide a means to indicate when supportive content has changed based on vulnerabilities discovered.

My results indicate that ESIDE can increase the advanced students' awareness and knowledge of secure programming, but to be most effective, instructors may need to incentivize its use through in-class methods and careful timing of its introduction.

7.1.2 Chapter 5 Summary and Contributions

In Chapter 5, I presented four early and one intermediate student studies and related findings. Two of the studies were conducted in a single setting lab environment, where students interacted with ESIDE as little or as much as they wanted as they worked on a lab project. One study with a focus group, another was a walk through study, and the fifth was a field study performed over a week long assignment where one of the three groups was offered points to include secure coding practices. Data I collected and presented in Chapter 5 was drawn from interviews, interaction logs, code written, and pre- post-test data. In total, 149 students participated in the five studies. Key findings from Chapter 5 include:

- ESIDE's interface provides a simple yet robustly supportive interaction. Most understood available content and the warning icon was effective.
- Those who chose to implement ESIDE's code recommendations (about 2/3 of the participants) appeared to understand why secure coding is important and how the recommendations would make their code more secure.
- Students have been conditioned by the one of the instructors to ignore Eclipse

warnings as they are confusing to early and intermediate students.

- Students who did not have a solid grasp of the classroom concepts expected ESIDE to provide instruction on those topics.
- Students who understood classroom concepts spent more time on interacting with ESIDE and asked questions as to what it did, how it was accomplished, and why that was important.
- Students who have secure coding requirements as part of their assignment tend to apply more secure coding techniques, increase their secure coding awareness and software security awareness at a higher rate than students in the same environment (i.e., same instructor, different section) without secure coding requirements.
- Having ESIDE educate on how secure code is built on top of functional code reduced confusion about how ESIDE can support the student.

7.1.2.1 Early and Intermediate Student Awareness and Use

On the whole, students appreciated ESIDE and liked the idea of having it available to them all the time. They felt as if ESIDE's icon identified code issues in a manner that discretely reminded them to explore why the icon was present. As they explored for an explanation, they felt as if the navigation was simple and the content was understandable.

An important finding amongst the early and intermediate studies is that ESIDE use did influence an increase in the security mindset of the students (RQ1). Most students reported that they were unaware of the security issues related to the code they were writing. After briefly working with ESIDE they became more aware of the

larger security picture "it shows me that though my code will run with proper user inputs, its still vulnerable" (p134).

Eclipse is a difficult IDE for early and intermediate students to use and understand because Eclipse provides interactions and warnings that are designed for professional level users. To remediate this issue, faculty conditioned their students to ignore Eclipse warnings. Despite this conditioning, students actively engaged with ESIDE.

7.1.2.2 Secure Coding Incentives and Motivations

Most students in my studies revealed a task completion mindset which can be expected from CS students [35]. To help answer my second and third research questions, I tested if minimal secure coding requirements on a week long assignment would influence the coding process and the students awareness. I found that students who had no incentive to learn about or write secure code (Section 2) performed considerably less than their counterparts in the other two sections. Their interaction with ESIDE and their awareness scores (RQ2) were noticeably lower than the other two sections.

The evident motivator for Section 1 was the requirement to write secure code as part of their assignment. The higher level of interaction over Section 2 can be attributed to this incentive. This requirement also had an effect on coding behavior as 11 out of the 14 completed assignments included secure coding (RQ3). In addition, their secure coding awareness scores greatly increased while their software security awareness only had a minimal increase. This finding relates back to functionality. To expand, the initial layers of ESIDE support are related to the identification and remediation of secure code issues while the last layer goes into depth why this is an issue which is where most of the software security awareness information is found.

So, students spent time learning about the practices needed to make their code work as required and then they moved on.

Section 3 interacted with ESIDE at a level similar to Section 1 (points incentive) and also experienced a large increase in their secure coding awareness scores (RQ2). I have attributed this finding to an unforeseen incentive which can be attributed to students using the assignment to study for the midterm exam which occurred the same day the assignment was due. When the students were provided with the assignment, the instructor informed them of its purpose of being a review activity. To this end, we postulate that they spent more time exploring ESIDE for information concerning classroom concepts. Once they did not find said information, they moved on. This behavior is supported in their post-test scores as their secure coding awareness increased at a much greater rate than their software security awareness scores which would have only dramatically increased if they had spent time exploring the final layer of ESIDE's support.

These findings reveal the importance of the instructor's role in guiding the student's education. While I designed ESIDE to be as autonomous as possible, it appears as if faculty will need to highlight the importance of secure coding by incentivizing its understanding and use.

7.1.2.3 Timing Exposure Amongst Early and Intermediate Students

Similar to advanced students, the timing of ESIDE exposure plays a critical role in the effectiveness of ESIDE. Too soon and the students might become confused and desensitized to future use. Too late, and crystallized habits and understandings will need to be re-learned which can be more difficult than learning a topic correctly the

first time. Because of the importance of timing, I designed ESIDE studies to align with expected student competency based on course progression.

Out of my studies I discovered two important findings. The first was a variation between institutions. One institution implemented an experiential approach to learning while the second was more lecture-based learning. The experiential group was able to readily implement ESIDE support into their code. However, the lecture group weren't as proficient at writing code and looked to ESIDE for help with converting classroom principles into practice. Such an interaction could have a dampening affect on interaction with ESIDE as students were unable to resolve their issues with ESIDE support. This group will need become more comfortable with the code writing process prior to being exposed to topics that build upon written code.

The second discovery was the influence that student preparedness has on the student's expectations of available ESIDE support. Those who used lab time to understand classroom materials tended to explore ESIDE for explanations to classroom materials rather than secure coding materials. While ESIDE interaction and materials did not necessarily confuse the student, they were unable to obtain instruction on the topic that they needed.

Initially, I conceptualized ESIDE to be continuously available to students as they learn to code over the course of their education. However, these findings reveal the potential of abandonment as students might view ESIDE as being non-helpful. In light of this new knowledge, we will need to explore how to control for timing. The resolution could be from one of three design areas. The first is to have ESIDE ascertain the appropriate time to initiate interaction. The second is to provide the instructor

with the ability to control when and what ESIDE provides the students and the third is a means for students to assess when they are ready for support.

7.2 Implications on Future Designs

Many of my findings have an implication on future designs of ESIDE. To begin, the icon serves as an unobtrusive indicator of a coding issue. As students progress through the semester they introduce new vulnerabilities which have a different set of educational materials. However, there existed no indication that the educational content is different as the icon stayed the same. To overcome this issue, ESIDE now provides a different icon for input and output issues. However, this is very general. Future designs will need to consider what granularity in needed for indicating the category of supportive content (e.g., different icon for every issue vs one icon for all issues).

A second finding that must be considered in future designs is the timing of ESIDE support. It would appear as if the advanced students in the semester long study became desensitized to the icon as when it first made itself available was on placeholder code. The students were not ready to implement ESIDE's recommendations because the placeholder code was not functional and students had yet to learn about the concepts driving the placeholder code. The students were unaware when they were ready to implement ESIDE's support as there was no indication that anything had changed. Unfortunately, the most opportune time to introduce an educational moment might have been missed. This condition was similar with some of the early and intermediate students who were not ready to work with ESIDE as they did not fully comprehend the classroom principles that ESIDE used to build upon. Future

designs will need to take into consideration how to inject ESIDE support into the learning process at the moment the support will be the most effective. This could be accomplished with an automated code review that waits until students successfully implement code that ESIDE is complementing or provide a means for the instructor to control when students start receiving support. Another means to control for timing is to provide a means for students to indicate when they are ready for complementary support.

Through faculty interviews I discovered a need for instructors and students to control what code patterns ESIDE monitors. To this end I created an interface that allows the user to choose a level of support they would like (e.g., early, intermediate, advanced) or use their own rule set. Future designs could provide even more control by providing a central repository that ESIDE continuously accesses for directions on what to monitor, when to start monitoring, and the content that complements the monitored code. The repository design would need to be a simple interface that allows the instructor to indicate code patterns, when they should be searched, and complementing menus and content.

Students are inherently motivated to complete a task rather than explore complementing topics [35]. In order for them to step out of this paradigm, they need an incentive to stimulate the motivation to learn and implement a new topic. While students in my studies appreciated ESIDE and learning did occur, little coding behavior changed without the incentive of points built into activities. To be effective, ESIDE will need a stimulating mechanism to motivate a behavior change amongst students. The simplest means is to have instructors actively integrate ESIDE into

course offerings. Another avenue is to explore a more engaging interaction experience with ESIDE that overcomes the task completion mindset barrier.

7.3 Theoretical Implications

I framed my work with the Integrative Learning Theory which describes how the blending and synthesization of multiple perspectives helps students make connections across the curriculum. ESIDE incorporates the Integrative Learning Theory by introducing secure coding concepts in a manner that complements classroom coding concepts. I discovered that a connection between secure coding and classroom concepts was being made as indicated by an increase in knowledge (RQ1) and awareness (RQ2) amongst those who actively engaged with ESIDE. This finding supports the implementation of the Integrative Learning Theory in settings where topics can complement each other such as code learning and secure code learning.

Considering how well the Integrative Learning Theory framed my research, I would encourage encourage others to further the application and development of these two paired theories. By doing so, these paired theories might provide insights and explanations as to why adult learners interact with the learning process as they do.

7.4 Research Implications and Future Directions

The instruction of computer science topics (i.e., fundamentals through advanced) has been and continues to be accomplished in a segregated fashion. Students receive pieces to the bigger CS picture as they work through their core courses and electives. Eventually, students assemble these pieces during a senior project or similar task. Unfortunately, this process misses out on the opportunity to facilitate strong connections amongst topics during the learning process.

My work has drawn from the medical community and the Integrative Learning Theory and initiated a new CS instructional approach by facilitating the connection between segregated topics in the student's IDE. While each of my studies summarized above provides insight into instructional support in the IDE for specific environments, the bigger picture they reveal is that it is possible to facilitate the connection between topics that are typically presented separately. By incorporating such an approach, students can holistically develop an understanding of the bigger picture of CS as they go through the learning process rather than assembling all the pieces at the end of their academic track. The result is a group of students who perceive CS as a whole rather than pieces needing to be assembled. Consequently, they are better prepared to create and maintain diverse technologies that are more robust in nature.

With this new approach I foresee many opportunities for future directions and research. Content from many elective courses (e.g., accessible design) can be included in the support available from ESIDE. However, this introduces the potential for cognitive overload. At what point does the available complementing content saturate the student's ability to learn new topics? Once the threshold is discovered, what should be done with the topics above said threshold. How should the topics be ranked to ensure students are provided with a well rounded education? Who should make that decision?

While I developed the ESIDE approach to complement the CS classroom with secure coding instruction, it is possible to modify ESIDE so that it supports classroom topics in the IDE. To this end, ESIDE could serve as a primary component of any given CS course. Instructors could build in their own materials for pre-defined code

they know their students struggle with. Through research, it would be possible to discover how well ESIDE could facilitate the learning of class materials.

An additional opportunity for future research is the application of the ESIDE approach to online learning environments such as distance education classes or Massive Open Online Courses (MOOCs). As many students collaborate through electronic resources such as Google Hangout, is it possible to have students develop their own rules that are shared to a public repository? In this manner, students serve as contributors to a supportive knowledge base.

7.5 Conclusion

I began my research by presenting the following thesis statement:

> The ESIDE approach to secure code education complements the code learning process by integrating real-time secure code instruction into the IDE. When ESIDE is deployed, it is possible to enhance a student's secure programming mindset, knowledge, and abilities with ESIDE.

As I explored the application of ESIDE in an educational environment I kept in mind my thesis and how the overall goal of ESIDE is to serve as an effective means to educate students at every level on the principles and practices of secure coding throughout their educational experience. To this end, I explored the usability and effectiveness of ESIDE in early, intermediate, and advanced student environments at multiple academic institutions. I discovered that ESIDE can serve as an effective means to enhance the student's mindset, secure coding knowledge and abilities. Advanced students reported an appreciation for the real-time support, as it raised their awareness of their own insecure coding practices and helped them learn remediation

techniques. Intermediate students actively interacted with ESIDE and integrated secure coding techniques (e.g., string input validation) into their lab work. Early students were receptive to ESIDE and readily implemented basic secure coding techniques (e.g., character checks).

As new generations of coders enter the process of learning how to become computer science professionals, it is imperative that educators strive to motivate their students to become the best creators of our world's future technologies. To this end, it is the educator's responsibility to develop and deploy educational approaches and resources that provide an engaging learning environment that reflects societal needs. One such approach and resource is ESIDE which effectively contributes to the enhancement of the student learning experience.

REFERENCES

[1] Adobe. Adobe secure software engineering team (ASSET) blog: Training secure software engineers, part 1. https://blogs.adobe.com/asset/2013/05/training-secure-software-engineers-part-1.html, 2013.

[2] S. Azadegan, M. Lavine, M. O'Leary, A. Wijesinha, and M. Zimand. An undergraduate track in computer security. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, ITiCSE '03, pages 207–210, New York, NY, USA, 2003. ACM.

[3] M. Bishop and D. Frincke. Teaching secure programming. *IEEE Security Privacy*, 3(5):54–56, Oct 2005.

[4] D. Bradbury. Microsoft's new window on security. *Computers & Security*, 25(6):405 – 407, 2006.

[5] British Intelligence Agenccy: Government Communications Headquarters (GCHQ). Uk universities awarded academic centre of excellence status in cyber security research. http://www.gchq.gov.uk/press/pages/cyber-security-research-centres-of-excellence.aspx, 2013.

[6] B. J. Brown Leonard. *Integrative learning as a developmental process: A grounded theory of college students' experiences in integrative studies*. ProQuest, 2007.

[7] D. L. Burley and M. Bishop. Summit on education in secure software: Final report. Technical Report CSE-2011-15, Department of Computer Science, University of California at Davis, Davis, June 2011.

[8] G. W. Bush Administration. National security presidential directives (NSPD) 54: Cyber security and monitoring - Comprehensive National Cybersecurity Initiative (CNCI). http://www.fas.org/irp/offdocs/nspd/index.html, 2013.

[9] R. N. Caine and G. Caine. *Making connections: teaching and the human brain*. Association for Supervision and Curriculum Development, 1991.

[10] Centers of Academic Excellence (CAE) Team: NSA/DHS sponsored. National Centers of Academic Excellence in Information Assurance / Cyber Defense (IA/CD) Knowledge Units. http://www.cisse.info/pdf/2014/2014%20CAE%20Knowledge%20Units.pdf, June 2013.

[11] CERT. The CERT Oracle secure coding standard for java. https://www.securecoding.cert.org/confluence/display/java/The+CERT+Oracle+Secure+Coding+Standard+for+Java, 2013.

[12] CERT. Secure coding standards. https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards, 2013.

[13] CERT. Top 10 secure coding practices. https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices, 2013.

[14] CISCO. Cyber security: Superhero series. http://www.cisco.com/web/AP/partners/cyber_security.html, 2013.

[15] J. Davis and M. Dark. Teaching students to design secure systems. *Security & Privacy, IEEE*, 1(2):56–58, 2003.

[16] N. Davis, W. Humphrey, J. ST Redwine, G. Zibulski, and G. McGraw. Processes for producing secure software: Summary of US National Cybersecurity Summit Subgroup Report. *Security & Privacy, IEEE*, 2(3):18–25, 2004.

[17] ESIDE Research Team. Directory traversal / path traversal. http://hci.uncc.edu/tomcat/ElonSp13ASIDE/DirectoryTraversal.jsp, 2013.

[18] ESIDE Research Team. Input validation: Don't trust anything!!! http://hci.uncc.edu/tomcat/ElonSp13ASIDE/InputValidation.jsp, 2013.

[19] ESIDE Research Team. Input validation: Don't trust anything!!! http://hci.uncc.edu/tomcat/ASIDE/InputValidation.jsp, 2013.

[20] ESIDE Research Team. Output encoding. http://hci.uncc.edu/tomcat/ElonSp13ASIDE/OutputEncoding.jsp, 2013.

[21] ESIDE Research Team. Output encoding. http://hci.uncc.edu/tomcat/ASIDE/OutputEncoding.jsp, 2013.

[22] K. Evans and F. Reeder. A human capital crisis in cybersecurity: Technical proficiency matters. Technical report, Center for Strategic and International Studies (CSIS), 2010.

[23] Frost, Sullivan, $(ISC)^2$, and B. A. Hamilton. The 2013 $(ISC)^2$ global information security workforce study. https://www.isc2.org/workforcestudy/Default.aspx, January 2013.

[24] George Mason University: Department of Computer Science. Special Topics in Computer Science: Secure Coding: Cse 8990-01. http://www.cs.gmu.edu/~simon/cs499-simon-r.html, 2013.

[25] Hewlett-Packard. Hp fortify static code analyzer. http://www8.hp.com/us/en/software-solutions/software.html?compURI=1338812#.UfAJbt9QGY4, 2013.

[26] IBM Academic Initiative. Building skills for a smarter planet. http://www-03.ibm.com/ibm/university/academic/pub/page/academic_initiative, 2013.

[27] C. E. Irvine and S.-K. Chin. Integrating security into the curriculum. *Computer*, 31(12):25–30, 1998.

[28] S. Kaza, B. Taylor, H. Hochheiser, S. Azadegan, M. O'Leary, and C. Turner. Injecting security in the curriculum–experiences in effective dissemination and assessment design. In *Proceedings of the The Colloquium for Information Systems Security Education (CISSE)(Baltimore, MD, 2010)*, 2010.

[29] H. R. Lipford and B. Chu. NSF Grant #1044745 - Collaborative research: Supporting secure programming education in the ide. http://www.nsf.gov/awardsearch/showAward?AWD_ID=1044745, 2011.

[30] S. McConnell. Gauging software readiness with defect tracking. *IEEE Software*, 14(3):136,135, 1997.

[31] T. McGee and P. Ericsson. The politics of the program: ms word as the invisible grammarian. *Computers and Composition*, 19(4):453–470, dec 2002.

[32] Microsoft. Microsoft security development lifecycle. http://www.microsoft.com/security/sdl/default.aspx, 2013.

[33] Microsoft. Office, check spelling and grammar. http://office.microsoft.com/en-us/word-help/check-spelling-and-grammar-HP010117963.aspx, 2013.

[34] Mississippi State University: Department of Computer Science and Engineering. Special Topics in Computer Science: Secure Coding: Cse 8990-01. http://www.cse.msstate.edu/~allen/cse8990sp07/syllabus.htm, 2013.

[35] K. Nance, B. Hay, and M. Bishop. Secure coding education: Are we making progress? In *Proceedings of the 16th Colloquium for Information Systems Security Education*, 2012.

[36] K. Nance and B. Taylor. Teaching high school students to code responsibly. In *Proceedings of the 16th Colloquium for Information Systems Security Education*, pages 111–115, jun 2012.

[37] National Audit Office. The UK cyber security strategy: Landscape review. http://www.nao.org.uk/wp-content/uploads/2013/03/Cyber-security-Full-report.pdf, February 2013.

[38] National Institute of Standards and Technology (NIST). Computer Security Division: Computer Resourcce Center. http://csrc.nist.gov/, 2013.

[39] National Institute of Standards and Technology (NIST). Glossary of key information seccurity terms. http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf, 2013.

[40] National Institute of Standards and Technology (NIST). National Initiative for Cybersecurity Education: Training & professional development. http://csrc.nist.gov/nice/training.htm, 2013.

[41] National Security Agency (NSA) Central Security Service (CSS). Centers of academic excellence institutions. http://www.nsa.gov/ia/academic_outreach/nat_cae/institutions.shtml, 2013.

[42] W. H. Newell. Professionalizing interdisciplinarity: Literature review and research agenda. *Interdisciplinarity: Essays from the literature*, pages 529–563, 1998.

[43] Oracle. Oracle University. http://education.oracle.com, 2013.

[44] OWASP Foundation. Data validation. https://www.owasp.org/index.php/Data_Validation, 2013.

[45] OWASP Foundation. Injection theory: Escaping. https://www.owasp.org/index.php/Injection_Theory#Escaping_.28aka_Output_Encoding.29, 2013.

[46] OWASP Foundation. Input validation. https://www.owasp.org/index.php/Category:Input_Validation, 2013.

[47] OWASP Foundation. OWASP ASIDE project. https://www.owasp.org/index.php/OWASP_ASIDE_Project, feb 2013.

[48] OWASP Foundation. OWASP enterprise security api. https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API, 2013.

[49] OWASP Foundation. Secure coding practices - quick reference guide. https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide, 2013.

[50] OWASP Foundation. Secure coding principles. https://www.owasp.org/index.php/Secure_Coding_Principles, 2013.

[51] OWASP Foundation. Top ten project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, jun 2013.

[52] L. F. Perrone, M. Aburdene, and X. Meng. Approaches to undergraduate instruction in computer security. In *Proceedings of the American Society for Engineering Education Annual Conference and Exhibition, ASEE*, 2005.

[53] R. Potter and D. Fuller. My new teaching partner? Using the grammar checker in writing instruction. *English Journal*, pages 36–41, 2008.

[54] V. Pournaghshband. Teaching the security mindset to cs1 students. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 347–352, New York, NY, USA, 2013. ACM.

[55] Review Taskforce: CS2008. Computer science curriculum 2008: An interim revision of CS 2001. http://www.acm.org//education/curricula/ComputerScience2008.pdf, 2008.

[56] SAFECode. SAFECode: About SAFECode. https://training.safecode.org/about_safecode, 2013.

[57] SAFECode. SAFECode: Free on-demand training courses. https://training.safecode.org/courses, 2013.

[58] SAFECode. SAFECode Publications: Security Engineering Training Paper. https://training.safecode.org/about_safecode, 2013.

[59] SAFECode.org. Fundamental practices for secure software development. http://www.safecode.org/publications/, 2013.

[60] B. Taylor and S. Azadegan. Threading secure coding principles and risk analysis into the undergraduate computer science and information systems curriculum. In *Proceedings of the 3rd annual conference on Information security curriculum development*, InfoSecCD '06, pages 24–29, New York, NY, USA, 2006. ACM.

[61] B. Taylor and S. Azadegan. Using security checklists and scorecards in cs curriculum. In *National Colloquium for Information Systems Security Education*, pages 4–9. Citeseer, 2007.

[62] B. Taylor and S. Azadegan. Moving beyond security tracks: Integrating security in cs0 and cs1. *SIGCSE Bull.*, 40(1):320–324, mar 2008.

[63] B. Taylor, H. Hochheiser, S. Azadegan, and M. Leary. Cross-site security integration: Preliminary experiences across curricula and institutions. In *Proceedings of the Colloquium for Information Systems Security Education (CISSE)*, 2009.

[64] 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*, ITiCSE '11, pages 3–7, New York, NY, USA, 2011. ACM.

[65] The Times of India. Cybersecurity to be part of India's college, university curriculum. http://articles.timesofindia.indiatimes.com/2013-01-17/education/36393726_1_cybersecurity-security-scenario-information-security, January 2013.

[66] Towson University. Security injections at towson. http://cis1.towson.edu/~cssecinj/, 2013.

[67] Towson University. Security injections at towson: Input validation - "all input is evil" - cs0. http://cis1.towson.edu/~cssecinj/modules/cs0/input-validation-cs0-java/, 2013.

[68] UNC Charlotte: Department of Software and Information Systems. Course Listings: ITIS 4221. Secure Programming and Penetration Testing. http://sis.uncc.edu/academics/course-listings, 2013.

[69] UNCC Department of Software and Information Systems: Concentration Areas. Security injections at towson. http://sis.uncc.edu/academics/graduate-msit/concentration-areas, 2013.

[70] R. B. Vaughn Jr. Application of security to the computing science classroom. In *ACM SIGCSE Bulletin*, volume 32, pages 90–94. ACM, 2000.

[71] Veracode. State of software security report. http://www.veracode.com/reports/index.html, 2013.

[72] M. Viveros and D. Jarvis. Cybersecurity education for the next generation: Advancing a collaborative approach. Technical report, IBM Center for Applied Insights, April 2013.

[73] G. White and G. Nordstrom. Security across the curriculum: Using computer security to teach computer science principles. In *Proceeding of the 19th National Information Systems Security Conference*, pages 483–488, 1996.

[74] M. E. Whitman and H. J. Mattord. Designing and teaching information security curriculum. In *Proceedings of the 1st annual conference on Information security curriculum development*, InfoSecCD '04, pages 1–7, New York, NY, USA, 2004. ACM.

[75] M. Whitney, H. R. Lipford, and B. Chu. Embedding secure coding instruction into the ide: A field study in an advanced cs course. In *Proceeding of the 46th ACM technical symposium on Computer science education*, SIGCSE '15, New York, NY, USA, 2015. ACM.

[76] Wikipedia. Integrative learning. http://en.wikipedia.org/wiki/Integrative_learning, 2013.

[77] Wikiversity. Andragogy: Learning theories in practice: Integrative — Wikiversity. http://en.wikiversity.org/wiki/Learning_theories_in_practice/Integrative, 2013.

[78] World Wide Web Consortium (W3C). Character model for the World Wide Web 1.0: Fundamentals: Escaping. http://www.w3.org/TR/charmod/#sec-Escaping, 2013.

[79] J. Xie, B. Chu, and H. R. Lipford. Idea: Interactive support for secure software development. In . Erlingsson, R. Wieringa, and N. Zannone, editors, *ESSoS*, volume 6542 of *Lecture Notes in Computer Science*, pages 248–255. Springer, 2011.

[80] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton. ASIDE: IDE support for web application security. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 267–276, New York, NY, USA, 2011. ACM.

[81] J. Xie, H. Lipford, and B. Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 161–164, sep 2011.

[82] J. Xie, H. Lipford, and B.-T. Chu. Evaluating interactive support for secure programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 2707–2716, New York, NY, USA, 2012. ACM.

[83] J. Zhu, H. R. Lipford, and B. Chu. Interactive support for secure programming education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 687–692, New York, NY, USA, 2013. ACM.