Exercise Perceptions: Experience Report From A Secure Software Development Course[⋆]

Akond Rahman[1][0000-0002-5056-757X], Shahriar Hossain[2], and Dibyendu Brinto Bose[3]

[1] Tennessee Technological University, Cookeville, TN, USA, arahman@tntech.edu
[2] Kennessaw State University, Kennessaw, Georgia, USA, hshahria2012@gmail.com
[3] Reeve Systems, Dhaka, Bangladesh, brintodibyendu@gmail.com

Abstract. The ubiquitous use of software in critical systems necessitates integrating cybersecurity concepts into the software engineering curriculum so that students studying software engineering have adequate knowledge to securely develop software projects, which could potentially secure critical systems. An experience report of developing and conducting a course can help educators to gain an understanding of student preferences on topics related to secure software development. We provide an experience report related to the 'Secure Software Development' course conducted at Tennessee Technological University. We discuss student motivations, as well as positive and negative perceptions of students towards exercises. Our findings show cybersecurity-related topics can also help students gain skills related to software engineering, e.g., we observe exercises related to taint tracking can help students to be better at program comprehension.

Keywords: devsecops · education · experience report · secure software

1 Introduction

With the emergence of the fourth industrial revolution [4] the use of software is becoming pervasive in critical systems, such as energy, health care, and transportation [4]. Security weaknesses in software used in critical systems can create serious consequences, such as creating large-scale outages, as it happened for Americold, a U.S.-based cold storage company [5]. Examples of cybersecurity attacks similar to that of Americold, highlight the need of educating software developers about cybersecurity concepts. Educators have also acknowledged to bring in cybersecurity research concepts into the curriculum of software engineering so that students gain knowledge about the cybersecurity concepts [18].

To strengthen the computer science curriculum at Tennessee Technological University (TnTU), a faculty at the Department of Computer Science (CS) introduced the 'Secure Software Development' course in Fall 2020. The purpose

[⋆] Partially funded by the U.S. National Science Foundation (NSF) award # 2026869.
[4] https://jia.sipa.columbia.edu/fourth-industrial-revolution-shaping-new-era
[5] https://threatpost.com/food-supply-americold-cyberattack/161402/

of this graduate-level course was to provide students with fundamental knowledge and training on secure software development. The course focused on using a hands-on approach where students will learn about cybersecurity and software engineering concepts via class lectures as well as by solving programming exercises.

We present an experience report of the exercises that were conducted as part of the 'Secure Software Development' course. Our reported experience related to exercises can be helpful for other educators who want to adopt secure software development as a course into their CS curriculum. Furthermore, our experience report can provide clues for researchers on how to better integrate cybersecurity into software engineering.

We answer the following research questions:

– ***RQ1****: What are students' motivations for enrolling in the 'Secure Software Development' course? Based on student feedback, which components of the 'Secure Software Development' are aligned with student motivations?*
– ***RQ2****: What is the performance of students in exercises conducted as part of the 'Secure Software Development' course?*
– ***RQ3****: What are the positive perceptions of exercises conducted as part of the 'Secure Software Development' course?*
– ***RQ4****: What are the negative perceptions of exercises conducted as part of the 'Secure Software Development' course?*

We answer the research questions by analyzing grade books and survey results collected from a graduate course titled 'Secure Software Development', which was introduced for the first time at TnTU. To synthesize students' positive and negative perceptions we apply open coding [17], a qualitative analysis technique to generate high-level categories from text input. Prior to conducting the survey and analysis we obtain Internal Review Board (IRB) approval from TnTU (IRB#2316).

Our contributions are listed as follows:

– A list of positive perceptions expressed by students regarding exercises conducted in the 'Secure Software Development' course;
– A list of negative perceptions expressed by students regarding exercises conducted in the 'Secure Software Development' course;
– A list of students' motivations to enroll in the 'Secure Software Development' course; and
– A publicly-available repository of materials used to conduct exercises in the 'Secure Software Development' course [2].

2 Overview of the Course and Exercises

The course is titled 'Secure Software Development', which was introduced in the graduate curriculum in the Department of Computer Science (CS) at TnTU for the first time. The pre-requisite of this course for students was to be enrolled

Table 1. Students' Experience in Cybersecurity and Software Engineering

Experience	Cybersecurity	Soft. Engg.
< 1 year	8	4
$1 - 2$ years	2	1
$3 - 4$ years	2	4
> 4 years	0	3

as a graduate student in the Department of CS at TnTU. Prior to conducting the course, the syllabus of the course was shared amongst all graduate students through e-mails in April 2020. A total of 12 students enrolled in the course. The instructor of the course conducted an initial survey of students' experience in software engineering and cybersecurity. The students' reported academic and professional experience in software engineering and cybersecurity is presented in Table 1. The course included three components: class lectures, exercises, and a semester long project assigned individually to each student.

The course included eight exercises that discussed eight topics related to secure software development. Before assigning each exercise necessary theoretical concepts were covered by the instructor. Each of the exercises maps to a knowledge unit (KU) recommended by the U.S. National Center of Academic Excellence in Cyber Defense Education (CAE-CD) [13]. KUs are CS-related topics deemed essential or recommended by the U.S. National Center of Academic Excellence to develop a curriculum related to cyber defense education. We describe each of the exercises below:

Exercise#1 - Git Hooks for Automated Security Static Analysis: The purpose of this exercise was to help students learn how to integrate security using a single example of Git hook [6]. Automated security static analysis is considered as a good practice to integrate security into software development workflows. If a software repository uses Git, then using Git-based utilities, such as Git Hooks, automated security static analysis can be performed. As part of this exercise, students were asked to learn about Git hooks, and how to create a Git hook so that upon committing a file, a security static analysis tool can run and scan all files in the repository. To perform security static analysis the students used `cppcheck`, a security static analysis tool for C/C++ code [7].

Exercise#2 - Logging Location: The purpose of this exercise was to identify locations where logging needs to be enabled for machine learning projects. In this exercise the students were asked to inspect machine learning code implementation in Python and identify locations where logging needs to be enabled but is not. Before assigning this exercise the students were exposed to concepts related to security-related logging provided by prior work [9] [5].

Exercise#3 - Privacy Violations in Software Projects: The purpose of this exercise was to make students aware of how implementation of a software project can violate privacy properties of individuals using the software. As part of

[6] https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
[7] http://cppcheck.sourceforge.net/

this exercise, *first*, the students read a scientific paper [15] and identified personally identifiable identifiers (PIIs), i.e., what utilities of the Android development kit are susceptible to leaking information that can identify an individual. Examples of PIIs include permissions used in Android, such as `ACCESS_FINE_LOCATION` and `GET_ACCOUNTS`. *Second*, the students had to identify if these permissions are used in a set of 50 Android applications. *Finally*, the students were required to report which of the identified permissions from the first step were used in the source code of the collected Android applications.

Exercise#4 - Security Requirements Validation: The purpose of this exercise was to help students understand how security requirements can be translated to test cases and observe if a given piece of software satisfy the specified requirements. Security requirements are a specific sub-category of software requirements that are related to ensuring confidentially, integrity, or availability [7]. As part of this exercise the students conducted test driven development, where they *first* wrote test cases for a simple calculator to satisfy the following requirements: (i) the calculator must be able to multiply and divide, (ii) all methods related to mathematical operations should sanitize input, (iii) all methods related to mathematical operations should handle division-by-zero exceptions, and (iv) all methods related to mathematical operations should be fast. *Second*, following the practice of test-driven development, the students were required to write code so that the test cases written in the first step are satisfied.

Exercise#5 - Security Smells: The purpose of this exercise was to allow students to apply their knowledge related to security smells gathered in the lecture and apply it to SaltStack [8] scripts. Security smells are recurring coding patterns that are indicative of security weaknesses [16]. SaltStack scripts are used to implement the practice of infrastructure as code (IaC), the practice of managing system configuration automatically using dedicated programming languages and by applying recommended software engineering best practices [16]. As part of this exercise the students were asked to perform two tasks: *first*, the students were asked to manually inspect three SaltStack scripts to identify security smells. *Second*, the students were asked to build an automated program to detect the identified security smell instances.

Exercise#6 - Security Static Analysis for Adversarial Machine Learning: The purpose of this exercise was to help students learn about how security static analysis can be conducted for machine learning projects at scale. Adversarial machine learning focuses on securing implementation of machine learning projects to protect against adversaries. Practitioners consider application of security static analysis as an important practice to protect machine learning projects against adversarial attacks [10]. As part of the exercise, *first*, the students were asked to use `bandit`, a static analysis tool for Python [9], and apply it automatically for 175 machine learning projects collected from the Model-

[8] https://www.saltstack.com/
[9] https://bandit.readthedocs.io/en/latest/

Zoo repository [10]. *Second*, the students were asked to automatically filter static analysis results that are of 'low' severity as reported by Bandit.
Exercise#7 - Taint Analysis: The purpose of this exercise was to give students hands-on experience about taint analysis. Taint analysis is the technique of tracking a potential security weakness in the source code for the software of interest [8]. Taint analysis can help to reduce false positives during security static analysis and also help understand which parts of the software are affected by the security weaknesses. As part of this exercise, the students had to inspect one Python file and perform two tasks: *first*, they had to report the complete flow of a taint, i.e., hard-coded password in the Python file. *Second*, they had to mine abstract syntax tree of the Python file to automatically report the complete flow of the taint.
Exercise#8 - White-box Fuzzing: The purpose of this exercise was to help students get hands-on experience on white box fuzzing and understand how white box fuzzing can help find faults in software. White-box fuzzing is the technique of providing malicious input by inspecting the source code of software artifacts and identify faults within the software [1]. In this exercise the students were asked to craft malicious input semi-automatically for an Ansible script. Ansible is a tool to implement the practice of IaC [16], which compiles and executes Ansible scripts to automatically provision cloud computing resources.

3 RQ1: Student Motivations

In this section we provide the methodology and findings for **RQ1: *What are students' motivations for enrolling in the 'Secure Software Development' course? Based on student feedback, which components of the 'Secure Software Development' are aligned with student motivations?***

3.1 Methodology to Answer RQ1

We collect student responses through an online survey that was deployed at the beginning of the semester. The purpose of this survey was to understand the experience level of students with software engineering and cybersecurity. As part of the survey we asked: "*What were your motivations to enroll in the 'Secure Software Development' course*"? The question was open-ended.

We apply a qualitative analysis technique called open coding [17] to generate categories from the text responses to the question. The derived categories of student perceptions are susceptible to rater bias as the categories are all derived by the first author. We mitigate this limitation by allocating another rater who is the last author of the paper. The last author provided a mapping for the obtained responses to the identified categories. The agreement rate is 100% with a Cohen's Kappa [6] of 1.0.

[10] https://modelzoo.co/

3.2 Answer to RQ1

We identify three motivation categories for enrolling in the 'Secure Software Development' course. We describe these categories below. The name of each category related to student motivations is followed by the count of students who mentioned the identified category:
Motivation#1 - Academic Requirements (2 out of 12): Two students enrolled in the course to satisfy course requirements: "*this class will be helpful for my masters thesis and professional career*".
Motivation#2 - Career Development (7 out of 12): Students were motivated by the fact that the content of the course could help in their career pursuits. As reported in Table 1, the enrolled students' experience in software engineering and cybersecurity varied, yet majority of the students perceived the course content to advance their professional career. One student stated "*I will be pursuing a cybersecurity related position, but I think that it [the course] will be something that will serve me well whether I choose to stay in a government position, go into private industry, or in academia*". Strengthening software engineering skills was also a motivating factor as one student stated "*software development is not my strong suit and I want to gain knowledge on how I can develop software applications in a more robust way considering security*".
Motivation#3 - Gain Research Background (3 out of 12): Students mentioned the focus and the content of the course may help them to conduct their research projects. One student mentioned "*I enjoy working in software security and I will be doing my course project consistent with my research work*".

We also asked students about which course component helped them to satisfy their motivations. The question was presented as a survey and all students participated. As shown in Table 2 we observe students to perceive exercises to be most aligned with their motivations to enroll in the course, followed by the semester-long project.

Table 2. Exercises are Perceived to be Best Suited with Enrollment Motivations

Experience	**Respondent count**
Exercise	11
Semester-long Project	9
Lectures	8

4 RQ2: Student Performance in Exercises

In this section, we provide the methodology and results for **RQ2:** ***What is the performance of students in exercises conducted as part of the 'Secure Software Development' course?***

4.1 Methodology to Answer RQ2

We answer RQ2 by using information related to percentage of task completed obtained from the course gradebook. Once the deadline for each exercise passed the instructor inspected and graded the submission materials. Grades were assigned based on the amount completed and correctness of the provided solution.

4.2 Answer to RQ2

We answer RQ2 by reporting summary statistics for grades obtained for each exercise. The summary statistics for grades is provided in Table 3. From the statistics presented in Table 3, we observe students to perform the worst for taint analysis. Students performed the best for security requirements validation.

Table 3. Summary Statistics of Grades For Eight Exercises

Exercise Name	**Stats** (Min., Median, Max.)
Git Hooks for Automated Security Static Analysis	(30%, 100%, 100%)
Logging Location	(71%, 100%, 100%)
Privacy Violations in Software Projects	(50%, 95%, 100%)
Security Requirements Validation	(85%, 100%, 100%)
Security Smells	(45%, 65%, 100%)
Security Static Analysis for Adversarial Machine Learning	(80%, 100%, 100%)
Taint Analysis	(20%, 30%, 70%)
White-box Fuzzing	(45%, 100%, 100%)

5 RQ3: Positive Perceptions of Exercises

In this section we provide the methodology and results for **RQ3:** ***What are the positive perceptions of exercises conducted as part of the 'Secure Software Development' course?***

5.1 Methodology to Answer RQ3

For each exercise the students were required to participate in a survey that asked two questions: *(i) Survey_Q1: What are the positive aspects of the exercise?, and (ii) Survey_Q2: What are the negative aspects of the exercise?* We use the answers provided by the students for Survey_Q1 to answer RQ3. We apply open coding [17] to determine categories that express positive aspects of the students for each exercise. Our process of applying open coding was similar to that of deriving student motivations described in Section 3.1. Similar to RQ1, we conduct rater verification, where the last author provided a mapping for the obtained responses to the identified categories related to positive perceptions of students. The agreement rate between the first and last author for the obtained responses is 65% with a Cohen's Kappa [6] of 0.53.

5.2 Answer to RQ3

We identify six categories of positive perceptions. A mapping between each identified category and each exercise is presented in Table 4. The number of students who have reported the category for an exercise is presented in parenthesis. For example according to Table 4, skill set development was mentioned by six students for the exercise related to security smell detection. We describe each identified category related to positive perception below:

Table 4. Positive Perceptions and Corresponding Exercises.

Exercise Topic	**Reported Positive Perception**
Git Hooks for Automated Security Static Analysis	Skill Set Development (8), Practicality (7)
Logging Location	Skill Set Development (4), Lecture Reinforcement (4), Program Comprehension (1), Practicality (4)
Privacy Violations in Software Projects	Skill Set Development (7), Practicality (4)
Security Requirements Validation	Skill Set Development (4), Practicality (3)
Security Smells	Skill Set Development (6), Lecture Reinforcement (3), Practicality (5), Sense of Accomplishment (1)
Security Static Analysis for Adversarial Machine Learning	Practicality (4)
Taint Analysis	Skill Set Development (5), Program Comprehension (3), Practicality (2), Self Evaluation (1)
White-box Fuzzing	Skill Set Development (6), Lecture Reinforcement (2), Practicality (7), Sense of Accomplishment (2), Self Evaluation (1)

Positive Perception#1 - Lecture Reinforcement: The conducted exercises provided students the opportunity to get a better understanding of what was being taught in the class lectures. The exercises complemented the class lectures by providing students clarity, as noticed by one student for the security smell exercise "*[it] was nice to actually use what we learned in class and reinforce the material*". For the logging-related exercise one student stated "*I got to actually implement some of the concepts discussed in class which can be beneficial to future work I will perform*". One student found the exercises to be a better medium for learning the concepts taught as part of the lecture "*I always learn better from assignments that involve coding rather reading/studying the subject*".
Positive Perception#2 - Practicality: All exercises were perceived as practical by the students. For the exercise related to privacy violation one student stated "*practical knowledge of identifying personally identifiable information (PII) in Android project source code*". For the exercise related to security requirements validation one student stated "*I had been introduced to TDD before theoretically, and the process did not really make sense to me. With this [exercise] and actually going through the process with a practical, hands-on example*

was very helpful in understanding how it works and the usefulness of the practice; practical knowledge gain".

Positive Perception#3 - Program Comprehension: For exercises related to logging location, security smell detection, and taint analysis, students were required to inspect source code. As part of the assignment students navigated source code files written in Python and SaltStack, which helped them to better navigate source code. The exercises helped students to get better at program comprehension. For example, in the case of taint analysis one student stated "*The exercise of manually going through the code to track the tainted paths was a valuable and helpful exercise*".

Positive Perception#4 - Self Evaluation: Students mentioned how the exercises helped them to self-evaluate their programming skills. The exercise related to taint analysis required programming using the 'ast' library [11], which helped students to assess what they knew. One student stated that the exercises are helpful because: "*they are highly applicable and from my personal point of view they exposing my shortcomings in programming*".

Positive Perception#5 - Sense of Accomplishment: The exercises helped students to gain a sense of accomplishment. For the white-box fuzzing exercise one student was able to find a bug in the Ansible compiler, which the student perceived as an accomplishment: "*Being able to use fuzzing to test a production application and being able to cause a crash in that application*".

Positive Perception#6 - Skill Set Development: For multiple exercises the students mentioned that the assigned exercises help them to learn new tools and techniques needed in software engineering. For the security smell exercise one student stated exercises of this nature "*is highly appreciated as it helps to get a diverse skill set*". Completion of the fuzzing-related exercise required students to learn on how to parse YAML files, which one student perceived positively and stated "*it was cool to use pyyaml, I haven't done that before*". About the exercise that involved security requirement validation a student stated: "*I've never used the python unit test module, and I believe this [exercise] gave me exposure and a hands on experience on performing/creating unit tests in Python*". The idea of using Git hooks for secure software development came as a pleasant surprise for one student "*Very cool to learn about git hooks and realize how useful it could be for software projects. I was not aware that git provided this feature prior*".

6 RQ4: Negative Perceptions of Exercises

In this section, we provide the methodology and results for **RQ4: *What are the negative perceptions of exercises conducted as part of the 'Secure Software Development' course?***

[11] https://docs.python.org/3/library/ast.html

6.1 Methodology to Answer RQ4

We use the answers provided by the students for Survey_Q2 ('What are the negative aspects of the exercise?') included in our survey to answer RQ4. We apply open coding [17] to determine categories that express negative perceptions of students for each exercise. Our process of applying open coding was similar to that of RQ1 and RQ3. We also conduct rater verification, where the last author provided a mapping for the obtained responses to the identified categories related to negative perceptions of students. The agreement rate between the first and last author for the obtained responses is 83% with a Cohen's Kappa [6] of 0.62.

6.2 Answer to RQ4

We identify three categories of negative perceptions expressed by students for exercises. A complete mapping between the identified categories and the applicable exercise is provided in Table 5. In Table 5, the 'Reported Negative Perception' column states the negative perception category names and the count of students who stated the category enclosed within parenthesis. 'None' indicates that no negative perceptions were reported by students for a certain exercise. We describe each of the categories below:

Negative Perception#1 - Artifact Management: All artifacts i.e., datasets and scripts for each exercise was shared using a Docker image. The Docker image was available using the instructor's DockerHub account, which included all necessary dependencies to run certain programs needed to complete each exercise. While downloading the Docker images one student commented: "*seems unnecessary to download a docker image of some 800+ MB to work on a small python file*". Transfer of files back and forth between the Docker image and the development environment also created negative experience for one student: "*dev environment is in Windows ... Docker is in a virtual machine ... passing files back and forth is tedious*".

Negative Perception#2 - Lack of Background: Despite detailed written instructions, we observe students to express a lack of background for each of the eight exercises. For example, while identifying and detecting security smells in SaltStack scripts one student found comprehension of SaltStack scripts to be difficult: "*I think SaltStack scripts are hard to look through especially if your not familiar ... I spent a lot of time trying to look up and research how to get the scripts to parse*". For the logging-related exercise, one student was not familiar with machine learning, and stated "*I cannot really think of any negatives other than my limited experience with machine learning and zero experience with the Keras library. I struggled to know exactly what the code was doing in the* `doDeepLearning` *function.*". Even though the instructions on how to use the Docker image were given for each exercise, the students faced challenges: "*I didn't know that 'exiting' from the shell will destroy the running image, and when I rerun the Docker image all my work was gone*".

Negative Perception#3 - Limiting Documentation: Students expressed negative perceptions while following the instructions provided in the documentation of software libraries. For the taint analysis exercise one student found the documentation of the Python-based 'ast' library: "*Need to use the python library AST, which is difficult to understand from the documentation*". Such views were expressed by multiple students for the Python-based 'javalang' library that was needed to complete the exercise related related to privacy: "*Struggled to find good resources for the javalang library beyond the basic examples and just ran out of time to try to get it to work*". Another student stated "*Couldn't find good documentation for javalang. Figuring out how to use the package was mostly trial and error with the `dir()` function and interactive python console to learn how to get the needed information*".

Table 5. Negative Perceptions and Corresponding Exercises.

Exercise Topic	**Reported Negative Perception**
Git Hooks for Automated Security Static Analysis	None
Logging Location	Lack of Background (2), Artifact Management (1)
Privacy Violations in Software Projects	Limiting Documentation (4)
Security Requirements Validation	None
Security Smells	Lack of Background (2), Artifact Management (1)
Security Static Analysis for Adversarial Machine Learning	Artifact Management (1)
Taint Analysis	Limiting Documentation (2), Artifact Management (1)
White-box Fuzzing	Artifact Management (3)

7 Discussion

We discuss the lessons that we learned from our findings as follows:
Students Prefer Real-world Exercises: We observe students to positively perceive exercises that involve code snippets collected from real-world open source projects and usage of real-world tools that are well-known in industry. Based on our findings, we advocate cybersecurity educators to design exercises and exams using real-world projects for a secure software development course.
The Good and the Bad of Exercise Diversity: Topic-wise exercises in the 'Secure Software Development' course are diverse, which involved a diverse set of technologies, such as SaltStack, Ansible, Python-based TDD, Android applications, Git hooks, and machine learning code developed in Python. On one side we have observed positive aspects of such diversity, for example, students being exposed to a diverse set of tools and techniques that enhance their skill set. On the other hand, students face challenges as they do not have necessary background. Based on our experience, we urge educators to be aware of the possible and negative aspects for introducing a diverse set of exercises, and find a balance that is adequate for a secure software development course.

Documentation and Tool Challenges: For multiple exercises students mentioned existing documentation to be limiting. For example, while completing the exercise related to taint analysis the documentation for 'ast' was hard to comprehend. Similarly, for the privacy violation exercise, students found the 'javalang' documentation to be hard to follow. Our findings show that students face documentation-related challenges while completing exercises. We urge software engineering researchers to systematically investigate the pervasiveness of the reported documentation-related challenges and identify techniques to mitigate such challenges.

From our reported findings in Section 6.2, we observe that use of Docker image may be inappropriate for exercises as it incurs overhead with respect to computation time and storage. We urge CS education researchers to synthesize the best practices on sharing artifacts for students, which will ensure that necessary dependencies of a software artifact is installed with limited overhead.

The Curious Case of Taint Analysis: From Table 3, we observe majority of the students to not complete the exercise related to taint analysis. Even though the students expressed positive perceptions about the exercise itself, we observe a disconnect between their perceptions and their ability to complete the exercise. One possible explanation can be attributed to the documentation of ast, which students found lacking. Another possible explanation is that students were not previously exposed to compiler-related courses, which hindered the students to conduct the exercise. Till date, TnTU does not offer compiler-related course, which could have exposed students to concepts, such as parse trees and abstract syntax trees. The instructor used one class lecture to expose students to concepts, such as parse trees, liveness of variables, and recursion, but that may not have been sufficient to mitigate the deficiency of the students. The lesson learned from conducting the taint analysis exercise is that (i) not all graduate students may not be proficient in parse tree mining and/or recursion, and (ii) before assigning taint analysis exercises instructors should dedicate multiple lectures on program analysis and recursion.

Limitations of the Paper: Our derived categories related to perceptions are susceptible to rater bias, as they were derived by the first author. We mitigate this limitation by assigning another rater who mapped student response to the identified categories. We also acknowledge the identified findings are limited to the sample size: our findings may not be generalizable to other courses related to secure software development that are conducted at other universities. Furthermore, our findings are derived from a course that was conducted once.

8 Related Work

Our paper is closely related with prior publications related to cybersecurity education. Beach [3] surveyed 129 education institutions that offer cybersecurity programs and reported 62% of the surveyed institutions do not consider human factors while developing their cybersecurity curriculum. Wood and Raj [20] described how key-logger exercises can be integrated into cybersecurity education curriculum. Lukowiak et al. [11] reported that presenting the course materials in an incremental manner helped students to reinforce the content provided in class lectures. Veneruso et al. [19] described their experience of using 'CyberVR', a game that uses visual reality, to teach cybersecurity concepts to students. Mountrouidou et al. [12] described their experience in integrating cybersecurity concepts into the general curriculum of a liberal arts degree and reported that if cybersecurity modules are flexible, then they can be incorporated into a general education curriculum. Olano et al. [14] reported their experience of introducing 'SecurityEmpire' in an undergraduate course to teach cybersecurity concepts to students. They [14] reported SecurityEmpire to help increase awareness and engagement about cybersecurity amongst students. Veneruso et al. [19] reported CyberVR to be equally effective, but more engaging in teaching cybersecurity-related concepts, compared to that of textbook-based methods. Theisen et al. [18] documented their experience of conducting a massively online open course (MOOC) related to secure software development, and observed on-campus students to have higher quiz scores than that of MOOC students.

The above-mentioned description shows the prevalence of experience reports related to a wide range of cybersecurity education concepts, such as hardware device, gaming, MOOCs, virtual reality, and industrial control systems. However, we observe a lack of research that discusses the experience of conducting a course related to secure software development, which we address in this paper.

9 Conclusion

We have reported our experience in conducting a secure software development course for the first time at TnTU. We document multiple types of perceptions that express students' positive attitude towards the assigned exercises, such as self evaluation, skill set development, and practicality. Students reported three categories of negative perceptions too, namely, lack of background, limiting documentation, and artifact management. Based on our findings, we recommend educators to integrate real-world exercises into a secure software development course with careful consideration of tool documentation, balance in exercise diversity, and student background.

References

1. Ammann, P., Offutt, J.: Introduction to software testing. Cambridge University Press (2016)
2. Anonymous: Materials for the Secure Software Development Course (12 2020), `https://figshare.com/s/f40c6df28ab2a2b55165`
3. Beach, S.K.: Usable cybersecurity: Human factors in cybersecurity education curricula. Nat. Cybersecur. Inst. J **1**(1), 5–15 (2014)
4. Bures, T., Weyns, D., Schmer, B., Tovar, E., Boden, E., Gabor, T., Gerostathopoulos, I., Gupta, P., Kang, E., Knauss, A., et al.: Software engineering for smart cyber-physical systems: Challenges and promising solutions. ACM SIGSOFT Software Engineering Notes **42**(2), 19–24 (2017)

5. Chuvakin, A., Peterson, G.: How to do application logging right. IEEE Security Privacy **8**(4), 82–85 (2010). https://doi.org/10.1109/MSP.2010.127
6. Cohen, J.: A coefficient of agreement for nominal scales. Educational and Psychological Measurement **20**(1), 37–46 (1960). https://doi.org/10.1177/001316446002000104
7. Firesmith, D., et al.: Engineering security requirements. J. Object Technol. **2**(1), 53–68 (2003)
8. Gupta, M.K., Govil, M.C., Singh, G.: Static analysis approaches to detect sql injection and cross site scripting vulnerabilities in web applications: A survey. In: International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014). pp. 1–5 (2014). https://doi.org/10.1109/ICRAIE.2014.6909173
9. King, J., Pandita, R., Williams, L.: Enabling forensics by proposing heuristics to identify mandatory log events. In: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. HotSoS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2746194.2746200
10. Kumar, R.S.S., Nyström, M., Lambert, J., Marshall, A., Goertzel, M., Comissoneru, A., Swann, M., Xia, S.: Adversarial machine learning–industry perspectives. arXiv preprint arXiv:2002.05646 (2020)
11. Lukowiak, M., Radziszowski, S., Vallino, J., Wood, C.: Cybersecurity education: Bridging the gap between hardware and software domains. ACM Trans. Comput. Educ. **14**(1) (Mar 2014). https://doi.org/10.1145/2538029
12. Mountrouidou, X., Li, X., Burke, Q.: Cybersecurity in liberal arts general education curriculum. In: Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education. p. 182–187. ITiCSE 2018, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3197091.3197110
13. NIETP: NIETP About CAE Program. `https://www.iad.gov/nietp/CAERequirements.cfm` (2020), [Online; accessed 18-Dec-2020]
14. Olano, M., Sherman, A., Oliva, L., Cox, R., Firestone, D., Kubik, O., Patil, M., Seymour, J., Sohn, I., Thomas, D.: Securityempire: Development and evaluation of a digital game to promote cybersecurity education. In: 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14). USENIX Association, San Diego, CA (Aug 2014), `https://www.usenix.org/conference/3gse14/summit-program/presentation/olano`
15. Onik, M.M.H., Kim, C.S., Lee, N.Y., Yang, J.: Personal information classification on aggregated android application's permissions. Applied Sciences **9**(19), 3997 (2019)
16. Rahman, A., Rahman, M.R., Parnin, C., Williams, L.: Security smells in ansible and chef scripts: A replication study. ACM Trans. Softw. Eng. Methodol. **30**(1) (Jan 2021). https://doi.org/10.1145/3408897
17. Saldana, J.: The coding manual for qualitative researchers. Sage (2015)
18. Theisen, C., Williams, L., Oliver, K., Murphy-Hill, E.: Software security education at scale. In: Proceedings of the 38th International Conference on Software Engineering Companion. pp. 346–355 (2016)
19. Veneruso, S.V., Ferro, L.S., Marrella, A., Mecella, M., Catarci, T.: Cybervr: An interactive learning experience in virtual reality for cybersecurity related issues. In: Proceedings of the International Conference on Advanced Visual Interfaces. AVI '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3399715.3399860
20. Wood, C., Raj, R.: Keyloggers in cybersecurity education. In: Security and Management. pp. 293–299. Citeseer (2010)