Training for Security: Planning the Use of a SAT in the Development Pipeline of Web Apps

Sabato Nocera, Simone Romano, Rita Francese, Giuseppe Scanniello
University of Salerno, Italy
{snocera, siromano, francese, gscanniello}@unisa.it

***Abstract*—We designed a prospective empirical investigation to study our *STW* (*Software Technologies for the Web*) course with respect to the training of bachelor students in the context of software security when developing e-commerce Web apps. To that end, we devised the following steps: *(i)* studying the state of the students enrolled in the STW course in the a.y. (academic year) 2021-22; *(ii)* defining a training plan for the a.y. 2022-23; and *(iii)* acting the plan and measuring the differences (if any) between the students of the a.y. 2021-22 and 2022-23. In this idea paper, we present the results of the former two steps, as well as the evaluation strategy of the proposed training plan. We observed that security concerns are widespread in the code of the Web apps the students of the STW course (a.y. 2021-22) developed. Therefore, we plan (second step) to ask the students of the STW course (a.y. 2022-23) to use in their development pipeline a Static Analysis Tool (SAT) to detect security concerns.**

***Index Terms*—Software security, Static analysis tool, Web app**

I. Introduction

Security concerns (aka *security issues* and *software vulnerabilities*) are flaws in the computational logic of software systems that, if exploited, would negatively impact software security in terms of confidentiality, integrity, and/or availability. The economic impact of security concerns can be catastrophic: for example, *eWEEK* estimated that *Heartbleed*—a software vulnerability in the popular *OpenSSL* cryptography library—has costed at least 500 million USD to companies [1]. While the introduction of some security concerns in the source code is unavoidable (*e.g.,* because a certain vulnerability is still unknown), others can be fixed before the delivery of software systems (or even developers could avoid their introduction). Web apps are not immune to security concerns and very often are not tested for security before being deployed (less than 50% are tested) [2].

Nowadays, supporting tools are available to help developers to identify and then fix security concerns before the delivery of software systems. Depending on the kind of analysis performed by these supporting tools, we can distinguish between dynamic and Static Analysis Tools (SATs). The former analyze software systems at run-time, while the latter examine the code of software systems without executing them. There are three scenarios in which developers can leverage SATs to identify and then fix security concerns [3]: *(i)* while developers code, by highlighting the presence of security concerns directly in IDEs (Integrated Development Environments); *(ii)* within a Continuous Integration (CI) pipeline, which could make a build fail if the code is not compliant with given security rules (*e.g.,* the committed code must not contain critical vulnerabilities); and *(iii)* during a code review.

In this idea paper, we propose a training plan, whose main goal is to provide evidence on the following question: *Does the use of a SAT help bachelor students in Computer Science (CS) to better deal with software security in the development pipeline of Web apps?* To answer this question, we defined a prospective empirical investigation consisting of the following steps: *(i)* studying the behavior of bachelor students in CS enrolled in the *STW* (*Software Technologies for the Web*) course in the a.y. (academic year) 2021-22 with respect to software security when developing Web apps; *(ii)* defining a training plan, based on the use of a SAT, to allow bachelor students enrolled in the STW course (a.y. 2022-23) to improve their behavior towards software security when developing Web apps; and *(iii)* experimenting that training plan and measuring the differences (if any) in students' behavior towards software security between the a.y. 2021-22 and 2022-23. In this paper, we present the results of the former two steps, as well as the evaluation strategy of the proposed training plan. We observed that security concerns are widespread in the code of the Web apps the students developed as mandatory projects of the STW course (a.y. 2021-22). Based on this finding, we can postulate that the students are not adequately equipped to develop secure Web apps. Therefore, we plan a training intervention in which the bachelor students of the STW course (a.y. 2022-23) are equipped with *SonarLint*, a SAT plugged into their IDE, to identify security concerns in their Web apps and then fix them as soon as possible in the development pipeline.

Paper Structure. In Section II and Section III, we outline related work and SonarLint, respectively. We present our contribution in Section IV, while we show and discuss the results in Section V. Final remarks conclude the paper.

II. Related Work

Recruiting people skilled in software security is a very hard task due to the shortage of these skills: the workforce gap is around 402,000 in North America and 199,000 in Europe, including 33,000 in the UK [4]. According to a study by McAfee, only 7% of the top universities (in the eight countries studied) offer an undergraduate major in cybersecurity [5].

Lam *et al.* [6] observed through a survey (involving 21 CS students from two US universities) that secure programming is not a mandatory skill: many security concerns can be easily detected and do not require security experts to be fixed. Therefore, a chip and easy solution to the skill shortage in software security is to teach security topics across all courses of a CS program [7]. In this way, secure programming may be inserted into the CS curriculum since the beginning classes [8]. Almansoori *et al.* [9] reported that this rarely happens, and students graduate with scarce or no secure programming knowledge. Also, Taeb and Chi [10] highlighted the gaps that exist in the current cybersecurity education and training landscape and proposed a framework to guide future professionals in developing secure software.

Web apps are particularly subject to attacks that can exploit, for example, flaws in the code or poorly-configured Web servers [11]. The problem could become even more relevant for e-commerce Web apps, which ask bank accounts and and personal information. If an attack to an e-commerce Web app succeeds, the effect on the reputation and credibility of such a Web app could be dramatic and the economic effects catastrophic. Thus, there is a need to integrate security practices into the development pipeline of Web apps and to have developers adequately trained to develop secure Web apps. Due to the shortage of security skills, it is desirable to train the next generation of Web developers by considering security not only as a simple add-on. To fill this gap, Zhu *et al.* [12] proposed and then experimented, in a Web development course, the use of *ASIDE*, a plug-in for Eclipse providing instant security warnings together with a description of the corresponding security concerns. The authors also conducted a preliminary observational study involving 20 students and the results provided some evidence that ASIDE could potentially help secure programming in the context of programming assignments. Similarly, Tabassum *et al.* [13] proposed an Eclipse plug-in, named *ESIDE*, to provide instant security warnings while coding. To compare ESIDE against a similar approach utilizing person-to-person feedback on security concerns (referred to as a security clinic), the authors conducted two studies in which took part 36 students. The participants in the studies accomplish programming tasks, which took 15 to 20 minutes, and then filled out a survey. The results indicated that both ESIDE and security clinic suffered from challenges in incentivizing students to incorporate secure programming techniques into their code.

We based our training plan on past evidence (*e.g.,* [6], [9], [10], [12]). Specifically, we postulated that there is a gap in the education and training landscape with respect to the development of secure software. We gather further evidence on this respect and design a plan to fill this gap in the context of Web-app development. As for the differences between our proposal and past research, the most remarkable ones follows: *(i)* we focus on e-commerce Web apps developed with Java-based technologies; *(ii)* we consider a large number of e-commerce Web apps (*i.e.,* 45) developed in teams by a total of 120 bachelor students; *(iii)* we plan a training intervention to fill the shortage of software security skills in Web-app development; and *(iv)* we plan to use in our training intervention a SAT, SonarLint, popular among developers.

III. SonarLint

SonarLint is a popular SAT (*e.g.,* on the Eclipse marketplace, SonarLint appears in the top ten of the most installed plug-ins of all time [14]). It allows developers to verify the compliance of their code against a pre-defined set of rules defined for 29 languages, including Java and JavaScript. When the analyzed code violates a rule, SonarLint generates an issue. SonarLint's issues are classified according to three quality characteristics: reliability, maintainability, and security—in our idea paper, we are interested in the security quality characteristic only. Security concerns—*i.e., hotspots* (security-sensitive code) and *vulnerabilities* (security concerns requiring immediate action)—are the kind of issues that SonarLint uses to rate the security quality characteristic. A severity level is also assigned to any kind of issue, security concerns included. From the least to most severe, these severity levels are: minor, major, critical, and blocker.

Developers can leverage SATs in three scenarios: *(i)* while coding in the IDE, *(ii)* within a CI pipeline, and *(iii)* during a code review. SonarLint is conceived to support the scenario *(i)* since it is available as a plug-in for the most popular IDEs (*e.g.,* Eclipse, IntelliJ IDEA, Visual Studio, *etc.*). Once SonarLint is plugged into an IDE (*e.g.,* Eclipse), it highlights vulnerabilities and hotspots (and other kinds of issues, which can be disabled) in real time. SonarLint also provides a detailed description of the identified security concerns as well as tips on how to fix them. In our training intervention, we plan to use SonarLint because: *(i)* we want to push students to fix security concerns like vulnerabilities or hotspots as soon as possible in the development pipeline; *(ii)* due to its popularity, SonarLint is likely to be used by CS students in the future (*i.e.,* once graduated); *(iii)* SonarLint is continuously updated to support the detection of new security concerns.

IV. Study Design

To plan our study, we followed the guidelines for experimentation in Software Engineering (SE) [15], [16]. In the following of this section, we describe the study design.

A. Goal

We formalized the goal of our study, through the GQM (Goal/Question/Metric) template [17], as follows:

> **Analyze** the use of a SAT (integrated into an IDE) **for the purpose of** evaluating its effect **with respect to** software security **from the point of view** of educators, researchers, and practitioners **in the context of** bachelor students in CS, who develop Web apps with Java-based technologies.

Based on the above-mentioned goal, we formulated the following Research Questions (RQs).

RQ1. *Are students equipped to manage the challenges associated with software security when developing Web apps?*

This RQ aims to study whether, and to what extent, bachelor students take care of software security when developing Web apps by using Java-based technologies. If the code of these Web apps contains security concerns, we can postulate that students are not equipped to manage the challenges associated with software security, and thus corrective actions are needed.

RQ2. *To what extent does the use of a SAT in an IDE affect software security when developing Web apps?*

This RQ aims to study whether, and to what extent, the use of a SAT (SonarLint), integrated into an IDE (Eclipse), improves the security of Java-based Web apps. If we observe that the number of security concerns is inferior when using a SAT (with respect to not using it), we can speculate that there is an improvement in the students' equipment to manage software security in the development of Web apps. This RQ has been defined here, but we will answer it when the last part of our training plan is concluded.

B. Context

The context of our study is represented by bachelor students in CS taking the STW course in the a.y. 2021-22 and 2022-23. The course is scheduled in the second semester of the second year of the CS program. It alternates theoretical and practical lessons, and covers the following topics: HTTP(S); HTML and CSS; Java-based technologies for Web development like Servlets, JavaBeans, and JSPs; MVC pattern for Web apps; authentication and access control; JavaScript and DHTML; XML and JSON; and AJAX. The reference IDE of the course is Eclipse. To pass the exam of STW, the students are asked to pass a written test and then carry out a project. To carry out the project, the students can choose to work in a team (composed of two to four members) or alone (*i.e.,* one-person team). The project consists of an e-commerce Web app. Each team is free to choose the e-commerce Web app to be implemented; however, each implemented e-commerce Web app must at least implement the following functional requirements:

- allowing customers and administrators to log-in and -out;
- allowing customers to sign-up, navigate the catalogs of items (*i.e.,* products or services), add items to the cart, buy items, and check own orders;
- allowing administrators to manage the catalogs of items (*e.g.,* modifying items) and customers' orders.

Further functional requirements are welcome especially for larger teams. Also, each e-commerce Web app must use the MVC pattern, interact with a database, and use the technologies presented in the course (*e.g.,* Servlets, DHTML, *etc.*). Before implementing the Web app, each team is asked to design that app and write a document, including: *(i)* high-level description of the Web app; *(ii)* market study of possible competitors; *(iii)* functional requirements; *(iv)* database schema; *(v)* navigation schema; and *(vi)* page template, theme and color pallet.

To answer RQ1, we use the projects delivered by the students of the STW course in the a.y. 2021-22, namely 45 e-commerce Web apps developed in teams by a total of 120 students. As for RQ2, we plan to use the projects delivered by the students enrolled in that course for the a.y. 2022-23.

C. Intervention and Measurements

The intervention of our study—needed to study RQ2—consists of providing the students (of the STW course) of the a.y. 2022-23 with a SAT, SonarLint, highlighting security concerns directly in the IDE. The training intervention includes lessons on SonarLint and on how to fix the security concerns—*i.e.,* hotspots and vulnerabilities—this SAT identifies. Therefore, these students are asked to use SonarLint when developing the mandatory project of the STW course (*i.e.,* an e-commerce Web app). On the contrary, the students of the a.y. 2021-22 did not use any SAT to carry out their project.

In both RQ1 and RQ2, we planned to quantify software security in terms of the security concerns—*i.e.,* hotspots and vulnerabilities—*SonarQube* identifies in the Web apps developed by the students as mandatory projects to pass the STW course. SonarQube is a popular SAT among developers and it has been receiving an increasing interest from the SE research community (*e.g.,* [18]–[21]). Both SonarQube and SonarLint are developed by *SonarSource*, and share the same security-concern detector—*i.e.,* SonarQube and SonarLint detect the same hotspots and vulnerabilities. While SonarLint works within the IDE, SonarQube is conceived to be integrated into a CI pipeline or to support code reviews. We used SonarQube to quantify the software security construct because, unlike SonarLint, it is possible to automatize the extraction process of security hotspots and vulnerabilities. To deal with threats to construct validity, we will not disclose to the students of the a.y. 2022-23 any information about our study goal, including the measurements based on SonarQube. Similarly, the students of the a.y. 2021-22 did not know such information.

D. Design and Experimental Procedure

We designed a *before-after* prospective study. We first measured the security hotspots and vulnerabilities the students introduced in their e-commerce Web apps at the time **T0**—*i.e.,* before the intervention and with bachelor students enrolled in the STW course, a.y. 2021-22—and then at the time **T1**—*i.e.,* after the intervention and with bachelor students enrolled in the STW course, a.y. 2022-23.

The experimental procedure consists of the following steps.

1) The students enrolled in the STW course (a.y. 2021-22) had to design and develop, in teams, e-commerce Web apps using Java-based technologies. We informed the students that the gathered data would be treated confidentially and anonymously shared for research purposes only.
2) The data gathered at the step 1 revealed that the students (a.y. 2021-22) did not take care of security hotspots and vulnerabilities when developing their e-commerce Web apps. Therefore, we defined our training intervention, consisting of providing the students (a.y. 2022-23) with SonarLint (as previously described in Section IV-C).
3) The students of STW (a.y. 2022-23) will be asked (to design and) develop e-commerce Web apps (using Java-based technologies) with the support of SonarLint. This (mandatory) teaching activity has a two-fold pedagogical goal: let the students familiarize with both the presented technologies

TABLE I: Some descriptive statistics.

Metric	Mean	SD	Min	Median	Max	Total
KLOC[*]	13.48	17.18	1.08	9.01	103.33	606.5
#JavaFiles	39.33	17.49	10	37	100	1,770
#JSPFiles	22.47	7.85	3	22	37	1,011
#JavaScriptFiles	5.31	6.32	0	3	31	239
#CSSFiles	8.91	9.23	1	6	49	401
#SecurityHotspots	41.78	25	6	38	103	1,880
#Vulnerabilities	0.04	0.3	0	0	2	2

[*] It stands for Kilos of Lines of Code, excluding comments and white spaces.

and the challenges associated to software security. The students who will take part in this step are different from those who took part in the step 1 and, therefore, we will inform them that their data would be confidentially treated and anonymously shared.

While developing the Web apps, the students of the a.y. 2021-22 were encouraged to push the local code changes to their remote Git repository. We will ask the same to the students of the a.y. 2022-23. The step 1 of the procedure corresponds to T0, while the step 3 to T1. In this idea paper, we present the results at T0 and introduce the corrective actions of the step 2 of our procedure. This is to say that T1 is needed to evaluate our training proposal (defined at the step 2).

V. Results and Discussion

Our preliminary results and implications are below, as well as possible threats to validity. Raw data are online [22].

A. Analysis of the Results

In Table I, we report some descriptive statistics—*i.e.,* mean, Standard Deviation (SD), minimum (min), median, maximum (max), and total—of the 45 Web apps the students of STW developed in the a.y. 2021-22. These descriptive statistics concern the size of these apps (*e.g.,* KLOC or #JSPFiles) and the number of detected security hotspots and vulnerabilities.

The Web apps contain a total of 1,882 security concerns (1,880 security hotspots plus 2 vulnerabilities). The average number of security hotspots is approximately equal to 42, while the median value is 38; however, taking into account that the SD value is 25, the number with which security hotspots appear in the Web apps is quite variable. The max and min values recorded in the apps are 103 and 6, respectively. Since security hotspots are code fragments that may pose critical security threats, it is necessary to carefully analyze them to understand to what extent they can cause actual threats to the security of a Web app. To that end, we show in Table II the complete list of violated security rules along with the number of security concerns (hotspots and vulnerabilities) and their severity level. We can observe that the security concern most present is: *delivering code in production with debug features activated*. This security concern has a minor severity level and would allow attackers to easily acquire detailed information on the running system, app, and users. For example, *Throwable.printStackTrace()* prints a *Throwable* and its stack trace to *System.Err* (by default), which can expose sensitive information. Other security concerns largely present in the code of the analyzed apps are: *formatting SQL queries*, *disabling resource integrity features*, and *using slow regular expressions*. These security concerns regard, respectively, the formatting of SQL strings that can lead to SQL injection, the absence of integrity checks on the external resources used (*e.g.,* CDNs), and the use of regular expressions with nonlinear complexity that can be exploited to cause a Denial-of-Service (DoS) of the Web app. The severity of these security concerns ranges from minor to critical. A more comprehensive description of security hotspots and vulnerabilities can be found in the SonarQube documentation [23].

TABLE II: Summary of the violated security rules.

Security Concern	#	Severity	Type
Delivering code in production with debug features activated	973	Minor	Security Hotspot
Formatting SQL queries	297	Major	Security Hotspot
Disabling resource integrity features	283	Minor	Security Hotspot
Using slow regular expressions	149	Critical	Security Hotspot
Hard-coded credentials	78	Blocker	Security Hotspot
Using pseudorandom number generators	62	Critical	Security Hotspot
Authorizing an opened window to access back to the originating window	31	Minor	Security Hotspot
Using clear-text protocols	4	Critical	Security Hotspot
Dynamically executing code	2	Critical	Security Hotspot
Encryption algorithms should be used with secure mode and padding scheme	1	Critical	Vulnerability
Cipher algorithms should be robust	1	Critical	Vulnerability
Using weak hashing algorithms	1	Critical	Security Hotspot

TABLE III: Total number of violated rules by severity level.

Minor	Major	Critical	Blocker
1,287	297	220	78

In Table III, we show the total number of violated rules aggregated by severity level. We can observe that blocker and critical security concerns account for, respectively, 4% and 12% of the total. The 78 security concerns, classified as blocker, are all *hard-coded credentials* (see Table II). This means that intruders can easily extract sensitive information from source or binary code. The results shown in Table III also indicate that security concerns (even those blocker and critical) are widespread in the analyzed Web apps. In Figure 1, we show a line plot of one of the studied Web apps depicting how the number of security concerns changes across the commit history. We observe that the number of security concerns increases with the number of commits. This trend is nearly the same for all 45 Web apps. A postulation for this trend (at time T0) is that either students were not accustomed to developing secure Web apps or there is a gap in the education and training landscape with respect to the development of secure software. To get some insights on this postulation, we conducted an unstructured interview with a few students enrolled in the STW course (a.y. 2021-22). We randomly sampled six students among those who participated in the 45 projects (at most one student for each project), then we asked the sampled students the following question: *The code of the Web app you developed included some security concerns, why did you not pay attention to this aspect in your development pipeline?* Based on the gathered

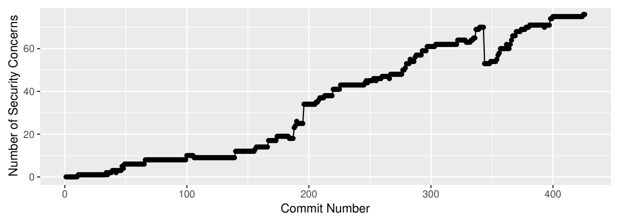

Fig. 1: Security concerns per commit of one project.

answers, we can sketch the following outcomes: *(i)* students did not use SATs in their development pipeline (*e.g.,* IDE) to detect security concerns; *(ii)* the development of secure Web apps was not considered as a mandatory requirement; and *(iii)* there is a will to better approach security in the development of Web apps. To deal with some threats to the validity of these qualitative outcomes we: *(i)* conducted the interviews after the students had passed the STW exam so that they were encouraged to answer sincerely; *(ii)* did not impose any time limit to answer the question and provide their perspective on the development of secure Web app; *(iii)* asked for clarifications only when strictly needed. The number of interviewees might pose another threat to validity. However, Johnson *et al.* [24] observed that a sample of six interviews may be sufficient to allow the development of meaningful themes and useful interpretations.

B. Threats To Validity

To determine the threats that might affect the validity of our study, we followed the guidelines by Wohlin *et al.* [16].

Threats to Internal Validity. We could not monitor, at T0, the students while accomplished the projects since developing e-commerce Web apps requires several working hours. However, we checked the delivered Web apps against plagiarism and no issue emerged; also, each team had to discuss its project at the delivery time with the course lecturers. We plan to do the same at T1. There might also be a *selection* threat due to the natural variation of the involved students at T0 and T1.

Threats to Construct Validity. There might be a threat of *restricted generalizability across constructs* at T1—*i.e.,* while we might find a positive effect of using a SAT on software security, there might be side effects on unconsidered constructs. Finally, although we did not disclose (and will not disclose) our research goal to the students, they might try to guess it and adapt their behavior based on their guess (threat of *hypotheses guessing*).

Threats to Conclusion Validity. We planned to involve students taking the same course (albeit in two different academic years), so having similar backgrounds and skills. Therefore, we mitigate a threat of *random heterogeneity of participants*. Also, the students at T1 will undergo a training to make them as more homogeneous as possible in terms of SonarLint usage and behavior toward software security. A threat of *reliability of treatment implementation* might occur at T1—*e.g.,* some participants might follow the tips to deal with security concerns more strictly than others.

Threats to External Validity. The kind of participants (at both T0 and T1) poses a threat of *interaction of selection and treatment*—*i.e.,* the results might not be generalized to any kind of student (or developer). The kinds of projects (at both T0 and T1) might represent another threat to external validity: *interaction of setting and treatment*. However, e-commerce Web apps are really widespread nowadays. Finally, we acknowledge that our results might not be generalized to a SAT different from SonarLint.

C. Implications

We observe a lack of skills in software security among the bachelor students of the a.y. 2021-22, and this is consistent with past evidence [5], [9], [12]. To deal with this lack, bachelor programs have been including security courses, but very often these courses do not provide practical means to let students deal with security concerns in their code [10]. This is what we learned from the first part of our study. We acknowledge this gap and will act as follows: *(i)* training students on how to deal with security concerns and *(ii)* promoting the use of a SAT, SonarLint, to identify their presence. Our outcomes are clearly relevant to educators, but also to researchers interested to develop *linters* that better detect security concerns and possibly suggest how to deal with them in the IDE. Researchers could be also interested to study to what extent the use of these tools actually improves the students' behaviors with respect to software security.

Bachelor students are not accustomed to developing secure Web apps. This result might be of interest to practitioners. In particular, they could be interested in knowing that junior developers (new hires) need to be adequately trained before being put in the production pipeline of Web apps. The second stage of our training plan goes in that direction and aims to share with the community an estimation of how this gap can be filled from a quantitative perspective when using SonarLint. We also plan to gather qualitative data from the participants (*i.e.,* students of STW, a.y. 2022-23) to identify strengths/limitations concerning the SonarLint adoption in the development of secure Web apps.

VI. Conclusion

We present an idea paper whose goal is to first understand if our Computer Science bachelor students, enrolled in the Software Technologies for the Web (STW) course, were equipped to handle security concerns in the Web apps they developed. Our outcomes suggest that a training plan is required because students do not pay attention to security concerns when developing Web apps. Therefore, we devised (as the second stage of our research) a training plan for the students of the next academic year of the STW course in order to improve students' behaviors towards security concerns. This training plan is based on the use of SonarLint—a Static Analysis Tool (SAT) that detects security concerns—and tips to fix security concerns. In the third stage of our research, we will implement and then evaluate our training plan. We believe that this plan will allow having new hires better trained on software security, and easier to insert into the job market.

References

[1] eWeek. (2020) Heartbleed ssl flaw's true cost will take time to tally. [Online]. Available: http://archive.today/IMt3t

[2] Ponemon Institute LLC. (2015) The cost of web application attacks. [Online]. Available: https://www.openreality.co.uk/wp-content/uploads/2015/08/2015-ponemon-institute-the-cost-of-web-application-attacks.pdf

[3] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir. Softw. Eng.*, vol. 25, no. 2, pp. 1419–1457, 2020.

[4] P. Muncaster. (2021) Global security skills shortage falls to 2.7 million workers. [Online]. Available: https://www.infosecurity-magazine.com/news/global-security-skills-shortage/

[5] Center for Strategic and International Studies. (2016) Hacking the skills shortage a study of the international shortage in cybersecurity skills. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-hacking-skills-shortage.pdf

[6] J. Lam, E. Fang, M. Almansoori, R. Chatterjee, and A. G. Soosai Raj, "Identifying gaps in the secure programming knowledge and skills of students," in *Proceedings of Technical Symposium on Computer Science Education V. 1*. ACM, 2022, pp. 703–709.

[7] G. White and G. Nordstrom, "Security across the curriculum: using computer security to teach computer science principles," in *Proceedings of National Information Systems Security Conference*, 1996, pp. 483–488.

[8] C. E. Irvine and S.-K. Chin, "Integrating security into the curriculum," *Computer*, vol. 31, no. 12, pp. 25–30, 1998.

[9] M. Almansoori, J. Lam, E. Fang, K. Mulligan, A. G. Soosai Raj, and R. Chatterjee, "How secure are our computer systems courses?" in *Proceedings of Conference on International Computing Education Research*. ACM, 2020, pp. 271–281.

[10] M. Taeb and H. Chi, "A personalized learning framework for software vulnerability detection and education," in *Proceedings of International Symposium on Computer Science and Intelligent Controls*. IEEE, 2021, pp. 119–126.

[11] T. Srivatanakul and F. Annansingh, "Incorporating active learning activities to the design and development of an undergraduate software and web security course," *Comput. Educ. J.*, vol. 9, no. 1, pp. 25–50, 2022.

[12] J. Zhu, H. R. Lipford, and B. Chu, "Interactive support for secure programming education," in *Proceeding of Technical Symposium on Computer Science Education*. ACM, 2013, pp. 687–692.

[13] M. Tabassum, S. Watson, B. Chu, and H. R. Lipford, "Evaluating two methods for integrating secure programming education," in *Proceedings of Technical Symposium on Computer Science Education*. ACM, 2018, pp. 390–395.

[14] Eclipse. (2022) Eclipse marketplace metrics. [Online]. Available: https://marketplace.eclipse.org/metrics

[15] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[16] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer, 2012.

[17] V. R. Basili and H. D. Rombach, "The tame project: Towards improvement-oriented software environments," *IEEE Trans. Softw. Eng*, vol. 14, no. 6, p. 758–773, 1988.

[18] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, "How do developers fix issues and pay back technical debt in the apache ecosystem?" in *Proceedings of International Conference on Software Analysis, Evolution and Reengineering*, 2018, pp. 153–163.

[19] D. Marcilio, R. Bonifácio, E. Monteiro, E. Canedo, W. Luz, and G. Pinto, "Are static analysis violations really fixed? a closer look at realistic usage of sonarqube," in *Proceedings of International Conference on Program Comprehension*. IEEE, 2019, pp. 209–219.

[20] M. T. Baldassarre, V. Lenarduzzi, S. Romano, and N. Saarimäki, "On the diffuseness of technical debt items and accuracy of remediation time when using sonarqube," *Inf. Softw. Technol.*, vol. 128, p. 106377, 2020.

[21] D. Pina, A. Goldman, and C. Seaman, "Sonarlizer xplorer: A tool to mine github projects and identify technical debt items using sonarqube," in *Proceedings of the International Conference on Technical Debt*. ACM, 2022, p. 71–75.

[22] Anonymous. (2022) Raw data of training for security. [Online]. Available: https://figshare.com/s/1b9359e7b4da866edfff

[23] SonarSource. (2022) Sonarqube rules by language. [Online]. Available: https://rules.sonarsource.com/

[24] G. Johnson, "How many interviews are enough? an experiment with data saturation and variability." *Field methods.*, vol. 18, no. 1, 2006.