







An Approach to Teaching Secure Programming in the .NET Environment

Sifiso Bangani¹ , Lynn Fletcher¹  , and Johan van Niekerk² 

¹ Department of IT, Nelson Mandela University, Port Elizabeth, South Africa
lynn.fletcher@mandela.ac.za

² Faculty of Computing, Noroff University College, Kristiansand, Norway
johan.vanniekerk@noroff.no

Abstract. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures and loss of information. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. Many software developers are not competent in secure programming. This leads to risks that are caused by vulnerabilities in the application code of software applications. Although various techniques for writing secure code are known, these techniques are rarely fundamental components of a computing curriculum. This paper proposes the teaching of secure programming through a step-by-step approach. Our approach includes the identification of application risks and secure coding practices as they relate to each other and to basic programming concepts. We specifically aim to guide educators on how to teach secure programming in the .Net environment.

Keywords: Programming education · Secure coding practices · Secure programming

1 Introduction

As the world advances in technology by creating new software applications, so does the need to protect these software applications as their vulnerabilities and associated risks also increase. Software applications have become integral to many people as they use them on a day-to-day basis for working with top-secret enterprise intellectual property, sharing personal information, making bank transactions, or sharing pictures with family and friends [12].

Although software plays an important role on a day-to-day basis, it often has associated risks as a result of vulnerabilities in the application layer [29]. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures, and loss

Financially supported by the National Research Foundation (NRF), NMU Research Capacity Development (RCD) and BankSETA, South Africa.

of information [18]. Failure to secure software solutions can have more serious effects than just a temporary interruption to a service. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. The human factor, which includes the programmer, has a major impact on the success and failure of efforts to secure and protect the business, services, and information [17]. According to [30], the main cause of software application failure is human error in application programming, which happens during the coding process.

Software developers are typically equipped with relevant programming knowledge and skills to develop innovative software [25]. However, software developers are rarely equipped with secure programming knowledge and skills from the undergraduate level [19,31]. According to [7], *“students graduating from technical programs such as information technology often do not have the attributes to fill the needs of industry”*. Fundamental programming principles are often introduced to students without an understanding of their security implications. This leads to non-adherence to secure programming [31]. For example, arrays and loops are introduced and explained without the mention of buffer overflows that could occur due to lack of adherence to secure programming.

The purpose of this paper is to help educators to teach secure programming in the .Net environment. Secure programming is an important part of information security education [32], whereby relevant topics of information security ought to be taught to some extent in all modules of the main curriculum throughout all years of study [15]. In this context, the contribution of this paper is five-fold:

1. It identifies relevant application risks in the .Net environment.
2. It identifies secure coding practices to be taught to undergraduate computing students.
3. It determines the basic programming concepts taught in the .Net environment in South African undergraduate computing curricula.
4. It maps the basic programming concepts to relevant application risks.
5. It maps the relevant application risks to the identified secure coding practices.

These mappings help us to understand how to teach secure programming in undergraduate computing curricula. By ‘computing curricula’ we mean university courses that teach programming for Computer Science or BSc IT degrees.

2 Related Work

As much as new software technologies are needed and are being developed, the industry increasingly demands software developers that possess relevant security knowledge, skills and abilities [7]. Advancements in technology also increase the security risks associated with those technologies. This leads to a ‘gap’ of outdated knowledge and skills for industry and academia [14]. In cybersecurity, according to [7], *“although jobs are and will be available, employers find it increasingly difficult to find qualified people to fill them. Students graduating from technical programs such as information technology often do not have the attributes to*

fill the needs of industry". Software security is becoming every company's norm and concern as a result of the rising trend of software application vulnerabilities [12, 14, 26]. This is the driving force behind the demand for software developers with security knowledge and skills. This security skill demand results in industry's need to hire developers experienced in secure programming. These developers must have the knowledge, skills, and abilities of secure programming that enables them to implement security-related solutions.

The security skills scarcity often forces companies to enrol their employees in secure programming certifications such as, IBM's Application Security Analyst Mastery Award, or Microsoft's Software Development Fundamentals course. These certifications are an attempt to make software developers competent in secure programming, as they often lack the required knowledge [9, 24]. However, the knowledge acquired with such certifications is not sufficient to be productive in secure programming without the necessary skills, as there should be a balance between knowledge and skills [28]. Secure programming certifications and training focus on two primary factors, namely: awareness of a specific security threat, and having adequate training in the use of the security counter-measure to such a threat [1]. However, these certifications and training do not guarantee a change in human behaviour [1, 17], as human behaviour requires more than just awareness of a specific security threat. For software developers to be competent in secure programming, they must be trained on the requisite skills of secure programming at an undergraduate level.

Producing competent software developers should therefore begin in universities and colleges where students are being educated in understanding and applying learned concepts and the ability to work in a team environment [6, 14]. Universities are responsible for providing a hands-on teaching approach for undergraduate students, which includes lectures, computer laboratory practicals, and experiments [28, 29]. The fundamentals of computing are introduced to students at university level. The learning outcomes of university curricula are used to show what the students will know and be able to demonstrate after the completion of that course [14]. They are key to the shift of focus in education from a paradigm of providing instructions to a paradigm of producing learning [3].

Various computing curricula guidelines, such as the Association for Computing Machinery's (ACM), state that Information Assurance and Security (IAS) belong to an advanced level of a four-year computing program. However, many students in three-year computing courses graduate and leave university before completing a fourth year of study [15]. This implies that many undergraduate computing students are often not adequately exposed to secure programming. Since students can only apply what they have been taught, the behaviour of a student in a certain area such as secure programming can be improved by providing students with the requisite knowledge [20]. This can be done through software security education in computing at universities.

Software security is the idea implemented to protect software to ensure that it functions correctly under malicious attacks [16]. Furthermore: *"Software security is about building secure software: designing software to be secure, making sure*

that software is secure, and educating software developers, architects, and users about how to build secure things" [16]. Software security is not simply implemented by installing an anti-virus software to a computer or mobile device, as hackers steal or get access to top secret enterprise information, or even damage the behaviour of software applications. Hackers can damage software through embedding malicious software or scripts in the code. Software applications without proper security built-in can be vulnerable to various computer attacks such as Cross-Site Scripting, SQL Injections, Session Hijacking, Cross-Site Request Forgery, and Denial of Service attacks [29]. The only way to avoid such attacks is by practising good secure programming techniques [2,30].

Secure programming is the manner of writing code to minimise software security vulnerabilities, as many problems faced by users nowadays are caused by vulnerabilities resulting from flaws in application code. Various techniques for writing secure code are known [23,27,31]. Although these techniques exist, they are rarely fundamental components of a computing curriculum, but rather treated as secondary topics that are briefly discussed in programming courses [31]. To maintain security in software applications, students must have the necessary skills and knowledge. According to [5], *"the ability to write secure code should be as fundamental to a university computer science undergraduate as basic literacy"*. The following section briefly describes computing education in the South African context.

3 Computing Education in the South African Context

Institutions of higher learning in South Africa are divided into public and private universities [11]. In this paper our focus is on the public universities. South African public universities are divided into three categories, namely: traditional universities, universities of technology, and comprehensive universities. The South African public universities are overseen by the government's Department of Higher Education and Training (DHET) which is responsible for post-school education and training.

South African universities offer three-to-four-year degree qualifications depending on the type of university [10]. Comprehensive universities and universities of technology offer three-year diploma qualifications for workplace readiness [15]. For a student in such universities to obtain a degree, the student would be required to advance the diploma qualification by completing a fourth year of study. Such a fourth year of study is considered to be 'advanced', wherein students can be introduced to higher-level topics [14]. In the case of programming qualifications, the fourth year of study would typically include an introduction to security and secure programming basics [14,15]. Traditional universities with three-year degree qualification teach programming basics in the undergraduate level. In traditional universities the fourth year of study (a.k.a. 'honours' in South Africa) is also considered to be an 'advanced' level wherein students are introduced to higher-level topics.

South African universities offer semester courses as well as year courses. Semester courses are usually carried out over a period of six months, whereas year

courses are carried out throughout the academic year [11]. Both types of courses in various universities offer fundamentals of programming. However, secure coding practices are rarely explicitly taught to undergraduate students. They are rather treated as secondary topics that are briefly discussed in those programming courses [31]. Examples of such courses include: Programming Fundamentals, Computing Fundamentals, Development of Software, Applications Development, Mobile Computing, Technical Programming, or Web Systems.

In this paper the focus is on applications development in the .Net environment, since most South African universities teach programming with it (whereby Microsoft promotes free product usage by university students). Nonetheless our proposed approach to teaching secure programming practices in undergraduate computing curricula is suitable for other development environments and frameworks as well.

4 Research Approach

A preliminary investigation and content analysis were conducted to determine whether South African universities incorporate secure programming in their undergraduate computing curricula. This preliminary investigation was conducted on South African universities through a thematic content analysis by reviewing the Prospectus or Study-Guides of various universities, where relevant themes and topics relating to secure programming were sought. The purpose of the investigation was to determine whether secure programming is already included in the teaching of programming concepts, as writing secure code ought to be fundamental to an undergraduate computing student [5]. A content analysis is typically conducted to make replicable and valid inferences from texts and data [13,20]. For this paper, content analysis was applied w.r.t. the above-mentioned matters.

Figure 1 sketches our research process which led to a step-by-step approach for teaching secure programming in undergraduate computing curricula with the following six steps; (for additional recommendations see [19]).

- Step 1:** Identification of relevant application risks in the .Net environment that are important for teaching secure programming.
- Step 2:** Identification of secure coding practices that should be taught to computing students as requisite knowledge for secure programming.
- Step 3:** Identification of basic programming concepts typically taught to undergraduate students (in the .Net environment).
- Step 4:** Mapping application risks to programming concepts to demonstrate the need for teaching application risks along with programming concepts.
- Step 5:** Mapping basic programming concepts to identified secure coding practices in order to highlight the need for, and relevance of, integrating secure coding practices to programming concepts taught.
- Step 6:** Mapping application risks to identified secure coding practices in order to show the relationship between application risks and secure coding practices to highlight the importance of secure programming.

Each step is described in further detail in the following sub-sections.

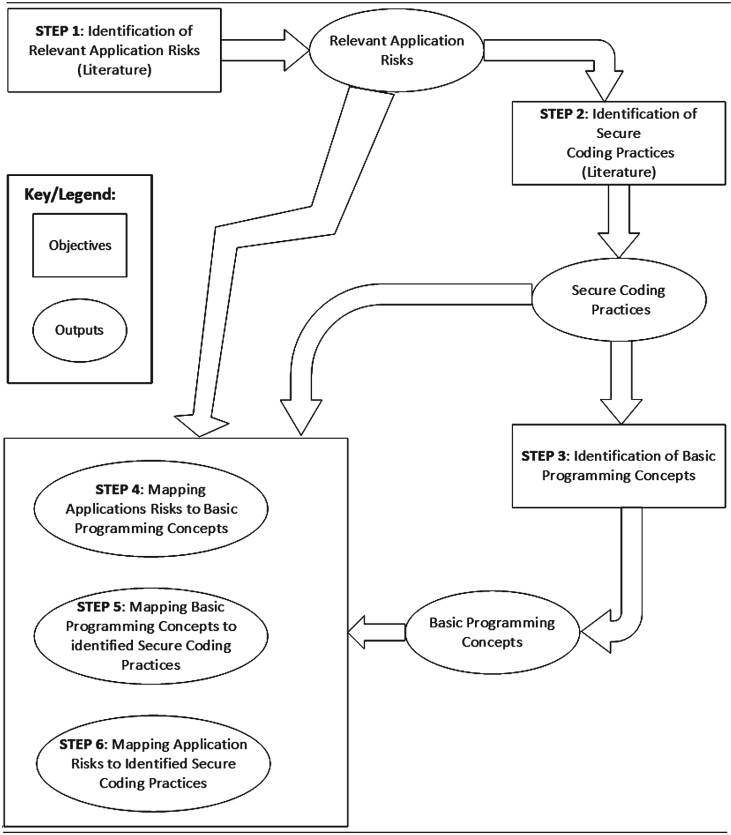


Fig. 1. Research process

4.1 Step 1: Identification of Relevant Application Risks (ARs)

An initial literature review identified application risks that can affect software applications developed in the .Net environment. The Open Web Application Security Project (OWASP) was used as a source of software application security guidance. OWASP is an international not-for-profit group dedicated to helping organisations to develop, purchase, and maintain good software applications [22]. OWASP is known for providing free documents of application risks, including a ‘Top 10 Application Risks’ document for awareness in *web application security* [7]. This document represents a broad consensus about the most critical security risks to web applications [23]. Although this OWASP list is mostly relevant to web applications, it can also be used for other software applications during application development, testing, and maintenance. The examples provided in this paper relate to web applications as they are often deemed the most vulnerable software applications.

Table 1. OWASP top 10 application risks for the year 2017

AR 1	Injection
AR 2	Broken authentication and session management
AR 3	Sensitive data exposure
AR 4	XML external entities
AR 5	Broken access control
AR 6	Security misconfiguration
AR 7	Cross-site scripting
AR 8	Insecure deserialisation
AR 9	Use of components with known vulnerabilities
AR 10	Insufficient logging and monitoring

Table 1 shows the OWASP Top 10 Application Risks (ARs) ordered by their severity. This list can also be used in the development of other software solutions that are not .Net-based, to guide and test for well-known vulnerabilities, as these application risks can affect most applications regardless of the development environment. In the identification of these application risks, the SANS Top 25 Programming Errors list [8] was used to compare the current application risks to well-known programming errors, to determine the extent to which the errors could cause the risks listed in OWASP’s list of Top 10 Application Risks. Some errors in the SANS Top 25 Errors list are no longer critical, as there have been changes in the security of development platforms and frameworks. This also resulted in the change in OWASP’s list of the Top 10 Application Risks, with cross-site scripting dropping from rank 2 in the list of 2013 to rank 7 in the list of 2017 [8, 23].

4.2 Step 2: Identification of Secure Coding Practices (SPs)

To identify the secure coding practices, a literature review identified principles, techniques, and practices of secure programming. The literature review of fundamental secure coding practices was conducted to understand what software developers need to be competent in w.r.t. secure programming [6].

The Secure Coding Practices Checklist recommended by OWASP was used for the identification of secure coding practices. The OWASP Secure Coding Practices Checklist can be used to mitigate most common software application vulnerabilities [22]. This checklist addresses the application risks listed in Table 1 and is used later in this paper to map with application risks and basic programming concepts. Table 2 shows OWASP’s Secure Coding Practices Checklist: the encoding identifier is SP followed by its rank number in the list.

In the ongoing investigation of secure coding practices to be taught to undergraduate students, the concept map by the University of California’s Davis Secure Programming Clinic [4] was reviewed for the identification and classification of programming practices. The identification and classification of the

Table 2. OWASP secure coding practices checklist

SP 1	Input validation
SP 2	Output encoding
SP 3	Authentication and password management
SP 4	Session management
SP 5	Access control
SP 6	Cryptographic practices
SP 7	Error handling and logging
SP 8	Communication security
SP 9	System configuration
SP 10	Database security
SP 11	File management
SP 12	Memory management
SP 13	General coding practices

secure coding practices was supported by the OWASP Secure Coding Practices Checklist [22]. This was done to assess the validity of the guidelines and principles in the current secure programming clinic.

4.3 Step 3: Identification of Basic Programming Concepts (PCs)

Having identified the secure coding practices, the globally published curricula guidelines for undergraduate computing programs were reviewed to determine the extent to which secure programming should be addressed in Computer Science (CS) and Information Technology (IT) qualifications. The focus was on the ACM curricula documents, as the ACM updates and adapts curricula recommendations quickly to the rapidly changing landscape of computer technology. Although the ACM curricula guidelines mention security as part of computing curricula, the guideline documents for CS and IT do not have adequate guidance on how secure programming can be taught to enable a graduate software developer to be competent in secure programming. The key to educating and training software developers is typically found in the Prospectus or Study-Guides pertaining to each university [14, 28].

To understand the state of programming in the undergraduate level, a thematic content analysis was conducted on undergraduate computing curricula in South Africa. The content analysis was done to determine basic programming concepts taught in the .Net environment, across different public universities in South Africa. The Prospectus and Study-Guides that are available on the universities' websites were used for this purpose.

Table 3 provides a list of basic programming concepts (PCs) that are typically taught in South African universities. The list does not provide any 'correct order' in which the concepts are taught; it only outlines the fundamentals of

programming for beginners developing in the .Net environment. Each PC item in the list is followed by its position number. Although PC8 and PC9 are not ‘concepts’ in the strict sense of the term, they were deemed essential enough by most universities to be taught to beginners developing in the .NET environment.

Table 3. Basic programming concepts for beginners in the .Net environment

PC 1	Variable declaration (with data types)
PC 2	Conditional structures
PC 3	Arrays (including search and update)
PC 4	Collections (arrays, lists, stacks, queues)
PC 5	Loops (while, for, until)
PC 6	Database connection
PC 7	File operations
PC 8	Basic HTML and XML
PC 9	JavaScript
PC 10	Web control
PC 11	Data binding
PC 12	Error handling
PC 13	Validation
PC 14	State management
PC 15	Master pages and layouts

4.4 Step 4: Mapping Application Risks (ARs) to Basic Programming Concepts (PCs)

After consolidating findings about what programming concepts should be included when teaching secure programming in South African universities, we created mapping links of how application risks can be taught when teaching programming concepts. The purpose of these mapping links is to demonstrate the need for, and relevance of, teaching application risks along with programming concepts. The mapping links in the content analysis results were given an impact value (**I**). **I** is based on how many times a programming concept (PC) was linked to an application risk (AR). **I** can also be seen as a way of prioritising important links. Figure 2 shows the mapping links between the identified programming concepts and the OWASP Top 10 Application Risks.

The mapping of programming concepts to application risks shows the relationship that the programming concept has directly to the application risk. A programming concept can have a number of application risks associated with it, and an application risk can occur due to poorly written programming concepts. A programming concept that links with many application risks receives a high impact value (**I**), where an impact value of 4 and above would mean that the link needs special attention. Therefore, educators of programming courses could

Programming Concept	OWASP Top 10 Application Risks 2017										(<i>I</i>)
	AR1	AR2	AR3	AR4	AR5	AR6	AR7	AR8	AR9	AR10	
PC1: Variable Declaration		X									1
PC2: Conditional Structures *	X	X			X			X			4
PC3: Arrays	X										1
PC4: Collections	X		X								2
PC5: Loops	X										1
PC6: Database Connection *	X	X	X	X	X						5
PC7: File Operations	X			X		X					3
PC8: Basic HTML & XML *	X			X		X	X		X		5
PC9: JavaScript		X					X		X		3
PC10: Web Controls *	X					X	X		X		4
PC11: Data Binding	X										1
PC12: Error Handling *	X	X	X		X	X				X	6
PC13: Validation *	X	X	X		X		X	X			6
PC14: State Management		X									1
PC15: Master Pages & Layouts		X			X						2
Impact (<i>I</i>)	11	8	4	3	5	4	4	2	3	1	(<i>I</i>)

Fig. 2. Mapping of basic programming concepts to OWASP Top 10 application risks

prioritise the lecture time taken for each link according to its impact value. In this paper, the programming concepts with a high impact value will be used to highlight the importance of considering application risks when teaching programming.

For example, the programming concept Error Handling (PC12) links to many application risks and, thus, it received a high impact value (**I**) of 6. Error handling is the last defence in a software application when written code statements do not execute as expected [22,30]. To educate students on how to program securely, the associated application risks must be taught after the introduction of the programming concept. The introduction of these application risks should begin from the first year of study, and should be taught in parallel with programming concepts as they occur in the syllabus. In an attempt to ensure that students adhere to secure programming and avoid these application risks, students must implement a means that recovers from errors (e.g. try-catch) [31].

Similarly, the programming concept Validation (PC13) received a high impact value (**I**) of 6 which shows its importance to software applications. Applications without proper validation of data can be vulnerable to various applications risks [23]. Students must therefore be encouraged to always use input validation to avoid the application risks such as Injection (AR1) associated with Validation (PC13) in Fig. 2 [9]. Encouraging students to do Validation (PC13) would require educators to examine the students’ adherence by setting laboratory practicals that require input validation. Students should be assessed and their work graded by reviewing the code they develop.

In Fig. 2, Injection (AR1) is the first in the OWASP Top 10 Application Risks, which shows how critical this risk is to software applications [23]. This application risk should be introduced and taught in parallel with the associated programming concepts to avoid this risk from occurring. To avoid Injections (AR1), students must be taught how they relate to each of the associated programming concepts (i.e. PC2, PC6, PC8, PC10, and PC12–13).

4.5 Step 5: Mapping Basic Programming Concepts (PCs) to Secure Coding Practices (SPs)

After understanding the application risks that must be taught to undergraduate computing students, we created mapping links of how secure coding practices can be taught in basic programming concepts. The purpose of the mapping links is to demonstrate the need for, and relevance of, integrating secure coding practices to basic programming concepts. The mapping links identified in the content analysis results were given an impact value (**I**); see Fig. 3. **I** is based on how many times a programming concept (PC) was linked to a secure coding practice (SP) horizontally according to the programming concept (PC), and how many times a secure coding practice (SP) was linked to a programming concept (PC) vertically. **I** can thus be seen as a way of prioritising important links that need special attention.

Programming Concept	OWASP Secure Coding Practices Checklist													I
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	
PC1: Variable Declaration	x			x									x	3
PC2: Conditional Structures	x		x	x	x		x	x					x	7
PC3: Arrays	x													1
PC4: Collections	x						x					x		3
PC5: Loops												x		1
PC6: Database Connection	x		x	x	x					x				5
PC7: File Operations	x	x					x		x		x	x		6
PC8: Basic HTML & XML		x	x						x					3
PC9: JavaScript	x										x		x	3
PC10: Web Controls	x						x						x	3
PC11: Data Binding							x			x		x	x	4
PC12: Error Handling	x	x	x	x	x	x	x		x	x	x	x	x	11
PC13: Validation	x	x	x	x	x	x	x	x	x	x	x	x	x	13
PC14: State Management	x				x		x							2
PC15: Master Pages & Layouts									x				x	2
Impact I	11	4	5	5	5	2	8	2	5	4	4	6	7	I

Fig. 3. Mapping of basic programming concepts to secure coding practices

The mapping link between programming concepts and secure coding practices shows a direct relationship between them. A programming concept can have a number of secure coding practices that can be associated with it, and a secure coding practice can be applied to a number of programming concepts. A programming concept that links with many secure coding practices receives a high impact value (**I**), whereby an impact value of ≥ 4 indicates that such a link ought to be given special attention.

Figure 3 shows that the programming concepts Error Handling (PC12) and Validation (PC13) in this mapping are most important, as they received the highest impact values (**I**) of 11 and 13 respectively. (Similarly, in Fig. 2, PC12 and PC13 achieved high impact values of 6.) Most application failures are a result of lack of Error Handling (PC12) and poor Validation (PC13). For Input Validation (SP1) to work effectively, it is mostly used with Conditional Structures (PC2) to avoid errors that might occur due to a lack of Error Handling (PC12) and Validation (PC13). When Input Validation (SP1) is not properly implemented, an application can be vulnerable to many application risks as shown in Fig. 2. When educators teach these programming concepts, they should therefore pay specific

attention to the impact caused by the association, and try to keep a balance between the programming concept and its associated secure coding practices.

4.6 Step 6: Mapping Application Risks (ARs) to Identified Secure Coding Practices (SPs)

After understanding the secure coding practices that must be taught to undergraduate computing students, we created mapping links that show the relationships between application risks and secure coding practices. As above, the mapping links in the content analysis results were given an impact value (**I**). Here, **I** is based on how many times an application risk (AR) was linked to a secure coding practice (SP) horizontally, and how many times a secure coding practice (SP) was linked to an application risk (AR) vertically; see Fig. 4. Again, **I** can be seen as a way of prioritising important links that need special attention.

Application Risk	Secure Coding Practices													(<i>I</i>)
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	
AR1: Injection	x	x					x			x	x	x	x	6
AR2: Broken Authentication	x		x	x	x	x	x		x				x	8
AR3: Sensitive Data Exposure						x			x	x	x			5
AR4: XML External Entities		x	x	x					x					4
AR5: Broken Access Control	x		x	x	x	x	x							6
AR6: Security Misconfiguration		x		x			x	x	x	x	x	x	x	9
AR7: Cross-Site Scripting	x	x					x		x		x			5
AR8: Insecure Deserialization	x	x					x		x					4
AR9: Using components with known vulnerabilities		x											x	2
AR10: Insufficient Logging and Monitoring							x							1
Impact (<i>I</i>)	5	6	3	4	4	2	7	1	6	3	4	2	4	(<i>I</i>)

Fig. 4. Mapping of identified application risks to secure coding practices

The mapping links between application risks and secure coding practices show a direct relationship between them. An application risk can have a number of secure coding practices that address it, and a secure coding practice can be applied to mitigate a number of application risks. An application risk that links with many secure coding practices receives a high impact value (**I**), where (**I**) of ≥ 4 would mean that such a link ought to receive special attention. Therefore, educators must not teach application risks and secure coding practices in isolation; the secure coding practices in Table 2 are used to prevent or mitigate the application risks in Table 1. Broken Authentication (AR2) in Fig. 4 links with many secure coding practices; thus it receives a high impact value (**I**) of 8. Software applications without a properly structured authentication mechanism can be vulnerable to privilege escalation [22,30]. The application risks Broken Authentication (AR2) and secure coding practice Authentication and Password Management (SP3) provide an example that can be used to teach students not to hard-code passwords, nor to leave plaintext passwords in the configuration files, as that can enable attackers to bypass access controls [30]. Error Handling and Logging (SP7) received a high impact value (**I**) of 7, which shows the importance that error handling and logging has in avoiding application risks such as

Sensitive Data Exposure (AR3). When error handling and logging is properly used in an application, default errors that show critical information such as server details are avoided by showing a custom error created by the programmer [21, 30]. To avoid Security Misconfiguration (AR6), students must be taught to avoid insecure default configurations and verbose error messages containing sensitive information. For ASP.Net applications, students can avoid Security Misconfiguration (AR6) by being taught to properly configure the `.config` file in the solution. A typical example of configuring the `.config` file would be to enable `customErrors`, so that default error messages will not be displayed.

5 Discussion and Conclusion

For (under)graduate software developers to be competent in secure software development they should be equipped with relevant and necessary secure programming knowledge in the curriculum. The literature shows that secure coding practices and techniques already exist [23, 27, 31]. However, they are rarely used as fundamental components of computing curricula, but are rather treated as secondary topics which are merely briefly discussed in programming courses [31].

Universities are responsible for educating undergraduate computing students where fundamentals of computing are introduced to students who are guided through practical classes in the computer laboratories [28]. Although many universities teach programming, often only little attention is given to secure programming, which results in incompetent undergraduate software developers. A university's Prospectus or Study-Guides are key to teaching undergraduate students, as these documents show what the student will know and be able to do apply after completion of the course.

Computing curricula reports such as the various ACM curricula guidelines recommend the teaching of secure programming in undergraduate computing courses. However, these guidelines do not provide adequate guidance on how secure programming can be integrated into the curriculum to enable a graduate software developer to be competent in secure programming.

The step-by-step approach proposed in this paper can be used at various levels for preparing a computing curriculum. Our approach can be used in setting up the Prospectus and Study-Guides, to ensure that relevant application risks and secure coding practices are appropriately considered. The steps proposed by this paper go hand-in-hand and cannot be addressed in isolation, as isolating these steps may lead to vulnerabilities that can affect a software application.

In addition, the mappings presented in this paper show the relationships between the programming concepts taught to undergraduate students with the identified application risks and secure coding practices. These mappings serve as a guide for how the application risks can be addressed by considering secure coding practices relating to basic programming concepts. Secure coding practices must be explicitly taught in undergraduate computing curricula to ensure that students will be competent in secure software development.

This paper proposes that secure coding practices be taught throughout the undergraduate programming modules, from the first year of study throughout

to the final year of study. This approach would not only impact the competence of graduate software developers, but it would positively influence the security of software applications developed by these graduate software developers for the benefits of society.

The main limitation of this paper is that the approach and the mappings suggested in this paper have not yet been thoroughly validated. This will be part of future research as well as the actual implementation of this approach at various universities in South Africa. In the next-following paper of this CCIS volume we address the ‘pervasive integration’ of secure coding principles into the entire computer science curriculum [19].

References

1. Aytes, K., Conolly, T.: A research model for investigating human behavior related to computer security. In: Proceedings of the 9th Americas Conference on Information Systems, pp. 1–6 (2003)
2. Aziz, N.A., Shamsuddin, S.N.Z., Hassan, N.A.: Inculcating secure coding for beginners. In: Proceedings of the ICIC International Conference on Informatics and Computing, pp. 164–168. IEEE (2016)
3. Barr, R.B., Tagg, J.: From teaching to learning - a new paradigm for undergraduate education. *Change Mag. High. Learn.* **27**(6), 12–26 (2012)
4. Bishop, M.: A clinic for secure programming. *IEEE Secur. Priv. Mag.* **8**(2), 54–56 (2010)
5. Bishop, M., Frincke, D.A.: Teaching secure programming. *IEEE Secur. Priv.* **3**(5), 54–56 (2005)
6. Buoncristiani, M., Buoncristiani, P.: How People Learn (2014). <https://doi.org/10.4135/9781483387772.n2>
7. Burley, D., Bishop, M., Buck, S., Ekstrom, J., Fitcher, L., Gibson, D.: Joint Task Force on Cybersecurity Education (2017). <http://www.csec2017.org/>
8. Christey, S., Martin, B.: CWE-2011 CWE/SANS Top 25 Most Dangerous Software Errors (2011). <http://cwe.mitre.org/top25/#CWE-209>
9. Cotler, J., College, S., Mathews, L., College, S., Hunsinger, S.: Information systems applied research 2015 AITP education special interest group (EDSIG) board of directors. *Inf. Syst. Appl. Res.* **8**(1), 1–65 (2015)
10. Department of Education: Creating Comprehensive Universities in South Africa: a Concept Document. Rep. of South Africa (2004)
11. Department of Education: Regulations for the Registration of Higher Education. Rep. of South Africa (1997)
12. Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., del Cuvillo, J.: Using innovative instructions to create trustworthy software solutions. In: Proceedings of the HASP 2013 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, p. 1 (2013)
13. Krippendorff, K.: Content Analysis: An Introduction to its Methodology (1985). <https://doi.org/10.1103/PhysRevB.31.3460>
14. Lunt, B.M., Ekstrom, J.J., Lawson, E.: Curriculum guidelines for undergraduate degree programs in information technology (2008)
15. Mabece, T., Fitcher, L., Thomson, K.L.: Towards using pervasive information security education to influence information security behaviour in undergraduate computing graduates. In: Proceedings of the CONFIRM 2016, p. 14 (2016)

16. McGraw, G.: Software security. *EEE Secur. Priv.* **2**(2), 80–83 (2004)
17. Metalidou, E., Marinagi, C., Trivellas, P., Eberhagen, N., Skourlas, C., Giannakopoulos, G.: The human factor of information security: unintentional damage perspective. *Procedia Soc. Behav. Sci.* **147**, 424–428 (2014)
18. Mumtaz, H., Alshayeb, M., Mahmood, S., Niazi, M.: An empirical study to improve software security through the application of code refactoring. *Inf. Softw. Technol.* **96**, 112–125 (2018)
19. Ngwenya, S., Fitcher, L.: A framework for integrating secure coding principles into undergraduate programming curricula. In: Tait, B., et al. (eds.) *SACLA 2019. CCIS*, vol. 1136, pp. 50–63 (2020)
20. van Niekerk, J.F., von Solms, R.: Information security culture: a management perspective. *Comput. Secur.* **29**(4), 476–486 (2010)
21. OWASP: Secure Coding Practices Checklist (2016). <https://www.owasp.org/>
22. OWASP: Secure Coding Practices Quick Reference Guide (2010)
23. OWASP: Top 10 2017: The Ten Most Critical Web Application Security Risks (2017). <https://www.owasp.org/>
24. Perrone, L.F., Aburdene, M., Meng, X.: Approaches to undergraduate instruction in computer security. In: *Proceedings of the ASEE Annual Conference and Exhibition*, pp. 651–663 (2005)
25. Rajlich, V.: Teaching developer skills in the first software engineering course. In: *Proceedings of the ICSE*, pp. 1109–1116 (2013)
26. Ramachandran, M.: Software security requirements management as an emerging cloud computing service. *Int. J. Inf. Manag.* **36**(4), 580–590 (2016)
27. Singhal, A., Winograd, T., Scarfone, K.: Guide to secure web services. NIST Special Publication 800–95 (2007)
28. The Joint Task Force on Computing Curricula: Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology. ACM/IEEE (2017)
29. Uskov, A.V.: Hands-on teaching of software and web applications security. In: *Proceedings of the IEDEC 3rd Interdisciplinary Engineering Design Education Conference*, pp. 71–78 (2013)
30. Veracode: State of Software Security (2017)
31. Whitney, M., Lipford, H.R., Chu, B., Thomas, T.: Embedding secure coding instruction into the IDE: complementing early and intermediate CS courses with ESIDE. *J. Educ. Comput. Res.* **56**(3), 415–438 (2018)
32. Wu, D., Fulmer, J., Johnson, S.: Teaching information security with virtual laboratories. In: Carroll, J.M. (ed.) *Innovative Practices in Teaching Information Sciences and Technology*, pp. 179–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-03656-4_16