# Integrating Secure Coding Principles into Undergraduate Programming Modules

**Full Paper**

**SACLA 2019**



Sandile Ngwenya [1[0000-0002-6766-224X]] and Lynn Futcher [2[0000-0003-0406-8718]]

[1, 2] Nelson Mandela University, Port Elizabeth, South Africa
{s215161033, lynn.futcher}@mandela.ac.za



**Abstract.** The rise of the use of the internet has led to significant growth in software applications for conducting business, entertainment and socialising, which in turn has led to a higher rate of attacks on software applications. This problem has led to the Information Technology industry requiring that software developers be skilled in developing software in a secure manner. The challenge that industry faces is that many software development graduates requiring employment do not have the requisite knowledge regarding secure programming. The need is therefore for academia to address the needs of industry by integrating secure coding principles into undergraduate programming modules. This paper highlights some secure coding principles that could be integrated into such modules. In addition, it discusses the challenges of, and various approaches to, integrating these principles into programming modules. Finally, it presents a framework for integrating secure coding principles into undergraduate programming modules to assist university departments in integrating these principles into their undergraduate programming modules.

**Keywords:** Secure Software Applications, Secure Coding Principles, Undergraduate Programming Modules, Secure Programming.

## 1 Introduction

Secure coding, also known as secure programming, is the practice of developing software programs that are safe from attacks [1]. This definition immediately introduces us to the practical aspect of the application of security to software. Over the past decade, the software development industry's focus has grown to include security. This change in focus signifies the need for software developers who can develop secure software applications. This need has resulted in a demand for continuing secure coding education as was stated over a decade ago [2] and has been further compounded due to the increase in the number of application vulnerabilities prevalent in software applications today.

[3] assert that many undergraduate programming classes teach some elements of secure coding, such as good program structure, basic input validation, checking bounds for array references and checking that pointers are non-null. However, many programming classes tend to focus more on the skills required for developing functional and user-friendly applications than on the security aspects. Without a conscious learning effort from both programming lecturers and students towards secure coding, software applications developed by many programming graduates will remain prone to vulnerabilities and susceptible to attacks.

Furthermore, [4] state that teaching secure coding has never been more important, suggesting that the need for the formal inclusion of secure coding principles into undergraduate programming modules must be effected. These secure coding principles should be assessed both theoretically and practically.

In order to address the problem of the lack of formal inclusion of secure coding principles into many undergraduate programming modules, this paper firstly highlights the need for teaching secure coding principles in undergraduate computing curricula as determined by various ACM curricula reports. Secondly, it presents some secure coding principles to be considered when teaching secure coding. Furthermore, it reports on the various challenges and approaches to integrating secure coding principles into programming curricula. Finally, it proposes a three-phased approach to integrate secure coding principles into undergraduate programming modules in the form of a framework.

## 2 Secure Coding in Computing Curricula

The Association for Computing Machinery (ACM) is the largest computing society in the world [5]. As part of its education initiative, the ACM produces and updates curricular recommendations in computer science, computer engineering, information systems, information technology, and software engineering that are trusted resources utilised by computing programs the world over [6]. The ACM periodically publishes curricular reports for the computing programs that it provides recommendations for [7, 8, 9].

As the field of IT continues to change and grow rapidly, IT curricula must follow suit. Inculcating modern skills into the IT curriculum prepares students for professional practice upon graduation [7]. Furthermore, [7] states that it is vital to include professional preparedness into IT curricula because graduates of IT programs will be faced with real-world problems within their workplaces. The report [7] delineates the technical IT Domains that constitute the education an IT undergraduate must learn. The Cyber Principles Domain is considered essential by the report [7].

In addition, [7] specifically includes "*A focus on implementation, operation, analysis, and testing of the security of computing technologies*" as part of its scope. As part of the Integrated Systems Technology Domain it requires undergraduate programming students to be able to "*Illustrate the goals of secure coding and show how to use these goals as guideposts in dealing with preventing buffer overflow, wrapper code, and securing method access*". This report therefore confirms that secure coding education must be part of undergraduate programming modules.

Similarly, [8] describes its Information Assurance and Security Knowledge Area where it specifies the need for security education in undergraduate Computer Science curricula. The report states that "*Information assurance and security as a domain is the set of controls and processes both technical and policy intended to protect and defend information and information systems by ensuring their confidentiality, integrity, and availability, and by providing for authentication and non-repudiation*" [8]. The mention of the technical aspect of security suggests that the practical aspects of security such as secure coding are relevant in undergraduate programming modules. Security education therefore includes all efforts to prepare graduates with the needed knowledge, skills, and abilities to develop information systems and attest to the security of processes and data [8].

[9] is in agreement and states that developers must ensure that their software is designed to meet security requirements [9], therefore further emphasising the need to educate undergraduate programming students in secure coding so as to meet the security requirements of industry.

The support for the inclusion of security in undergraduate programming modules by various ACM curricula reports [7, 8, 9] serves to show that attention to secure coding is mandatory for the education of computing students. This should ensure they have the ability to develop software applications that can perform the vital function of consistently securing critical data and information.

## 3 Secure Coding Principles

Secure coding refers to a software product's robustness against accidental or malicious unexpected behaviour causing a problem [10]. Coding responsibly means knowing how to develop secure code which is essential for the implementation of modern software systems [11]. It is therefore important that undergraduate students learn how to code responsibly by being taught the theoretical and practical aspects of various secure coding principles.

The implementation of secure coding principles is key in the development of software applications that adhere to security requirements. The list of secure coding principles defined below provides an example of secure coding principles that can be taught in undergraduate programming modules as identified by OWASP [12]. [9] also references the same principles within its Software Security Knowledge Area. It is important to note that the list is not all encompassing, but contains some fundamental secure coding principles that can be formally included in undergraduate programming modules.

- **Input Validation** refers to the process of ensuring that data that is entered as input in applications is filtered and without 'unclean' data [13].
- **Authentication and Password Management** is defined by [14] as the process of a server deciding whether a user should be allowed to login or not. Authentication controls must be validated on a trusted system (e.g. a server).
- **Session Management** tracks the activity of a user as they interact with a website across sessions. Its most widespread use is the login functionality, but it is also used to track other types of interactions users may have with a website [15].

- **Access Control** ensures that restricted areas of a software system are only accessed by authorised users. A user would have to provide credentials to be granted access [16].
- **Cryptographic Practices** involve the conversion of data into a format that is unreadable to users that are not authorised to do so [17].
- **Error Handling and Logging** entails the recovery of an application from an error condition using recovery responses. Error handling anticipates the possibility of error conditions, detects errors, and provides a resolution of an error to maintain the execution of an application [18].
- **Data Protection** aims to keep private data only available for business purposes or to maintain data privacy rights [19].
- **Database Security.** Information stored in today's databases is highly valuable and confidential. As such, secure coding must play an active role in the protection of that information from unauthorised access to it [20].
- **File Management.** Files typically store information within them. File types include text files, data files, binary and graphic files [21]. File upload controls should support anti-malware and anti-virus capabilities to avoid the upload of files that can compromise the application or database where they would get to be saved [22].

Table 1 depicts the above-mentioned secure coding principles and the related content that can be taught within programming modules.

**Table 1.** Secure Coding Principles and Related Content [12].

| **Secure Coding Principle** | **Related Secure Coding Content** |
|---|---|
| Input Validation | • Input validation should be processed on a trusted system (e.g. a server)<br>• Input that fails validation should be rejected.<br>• Expected data types must be validated.<br>• The data range of input must be validated. |
| Authentication and Password Management | • All authentication should be processed on a trusted system (e.g. a server)<br>• Passwords stored on a database must be encrypted.<br>• Password complexity must be enforced.<br>• Minimum password length must be validated. |
| Session Management | • Logout functionality should be protected by authorisation across all pages where it is used in a website.<br>• A session must be configured to expire within a reasonably short amount of time<br>• Session identifiers must not be exposed such as on URL headers. |
| Access Control | • Access controls should fail securely<br>• Only authorised users must be granted access to services. |

| | |
|---|---|
| Cryptographic Practices | • All cryptographic functions used should be processed on a trusted system (e.g. a server)<br>• Highly sensitive data must be encrypted |
| Error Handling and Logging | • Do not disclose sensitive information in error responses, including system details, session identifiers or account information.<br>• Feedback must be in the form of generic error messages and custom error pages must be used. |
| Data Protection | • Feedback must be in the form of generic error messages and custom error pages must be used.<br>• URL headers must not include sensitive information. |
| Database Security | • Parameterised queries must be used for the interaction between the application and a database.<br>• Connection strings should be stored and encrypted separately in a configuration file on a trusted system. They must not be hard coded. |
| File Management | • Files should never be uploaded without requiring authentication.<br>• Only allow the uploading of file types that are required for business purposes. |

Programming lecturers can use the related content in Table 1 as a basis for determining the learning outcomes that can be taught for each secure coding principle. This content should be aligned with the relevant undergraduate programming modules as seen fit by the educators of such computing courses.



## 4 Challenges and Approaches to Integrating Secure Coding Principles into Undergraduate Programming Modules

There are several challenges that affect undergraduate programming modules in integrating learning outcomes that are specifically aimed at teaching secure coding principles. Although there might be a genuine desire and interest in teaching secure coding, the reality is that it is not an easy task to accomplish. [10] mention three challenges that hinder progress when it comes to implementing the teaching of secure coding, including:

- The lack of room in the software development curricula;
- The focus of introductory software development courses; and
- Teaching in a manner that does not promote the application of learnt programming techniques.

The first challenge is due to the notion that software development classes that include secure coding principles must be separate classes. This is an assumption as introductory coding classes already teach some of the basics of secure coding such as validating

inputs and the handling of exceptions [4]. There are various ways of adding secure coding principles to undergraduate programming modules. The common approach would be to add lectures and practical classes that include the teaching of secure coding [23].

The second challenge stems from the primary focus of introductory undergraduate programming modules which is to teach programming logic. For many university faculties, the challenge is knowing where to infuse the secure coding principles within the sequence of the primary focus. Undergraduate programming classes mainly focus on algorithmic and programming language issues rather than that of secure coding. This focus therefore tends to be mostly on good program structure and logic. The norm is that undergraduate students do not need to know the secure coding concepts in detail [10]. This leads to students not being taught secure coding principles.

The third challenge speaks to the practical application of what students learn when they are taught how to code and how their work is graded. It is assumed that students can be introduced to basic secure coding techniques and that they apply the principles of secure coding. Students tend to focus on what they are being proactively taught and assessed on. The result is that students do not apply the principles of secure coding, as most often the grading also focuses on good program structure and whether their programs work. This leaves students with the implicit belief that security is not as important as functionality [10].

These challenges can be overcome if there is an extensive effort by faculty staff to integrate secure coding principles into undergraduate programming modules. An important factor in the drive to include comprehensive secure coding principles is to create a security mindset amongst faculty members as well as students. Many experts consider this approach to be one of the most important strategies in making progress towards improving the implementation of secure coding education in undergraduate programming courses [4].

This research acknowledges the need for a structured or logical method of integrating secure coding principles into undergraduate programming modules. [24] defines an approach for integrating a pervasive theme into undergraduate IT curricula. The pervasive theme approach provides a structured method for assisting in achieving this. Pervasive themes are defined as topics that are addressed multiple times from different perspectives [24]. This means that the teaching of similar content within undergraduate programming modules can exist between modules. Teaching in this manner is necessary for a topic such as secure coding because students need to learn secure coding as part of all programming modules holistically and repetitively. The suggestion is that of considering the use of pervasive themes across undergraduate programming modules.

A further approach to be considered is the pillars-first approach which provides students with the core understanding of foundational software development concepts before integrating pervasive themes into the curriculum. The main disadvantage of the pillars-first approach is that it tends to present each core concept in an isolated manner [24]. This challenge can be addressed by a joint effort by faculty staff by lecturing in a manner that enables students to understand how the various core concepts are interlinked.

This research proposes the use of the pillars-first approach as it allows introductory concepts to be understood by students first before integrating various secure coding concepts across the several modules taught within an institution's undergraduate programming curriculum. This approach also enables lecturers to gradually integrate secure coding into the curriculum without diminishing the focus of the core programming concepts [24]. The content for teaching students can then be taught by the lecturer at a rate they consider feasible for the given module. The importance of repeatedly addressing secure coding across several programming modules in computing curricula ensures that students understand that secure coding is an integral part of software development and not just a subset of it.

## 5 A Phased Approach for Integrating Secure Coding Principles into Undergraduate Programming Modules

The following sub-sections outline three phases that specify how secure coding principles can be integrated into undergraduate programming modules. The phases are: the Identification Phase, the Buy-In Phase and the Implementation Phase as discussed in the following sub-sections.

### 5.1 The Identification Phase

The Identification Phase as shown in Fig. 1 is the first phase and it is initiated by the requirements of industry that need academia to teach undergraduate programming students secure coding. These requirements may change over time so it would be essential for universities to keep up to date with these changes. Similarly, the standards and best practices relating to secure coding and related vulnerabilities may also change.

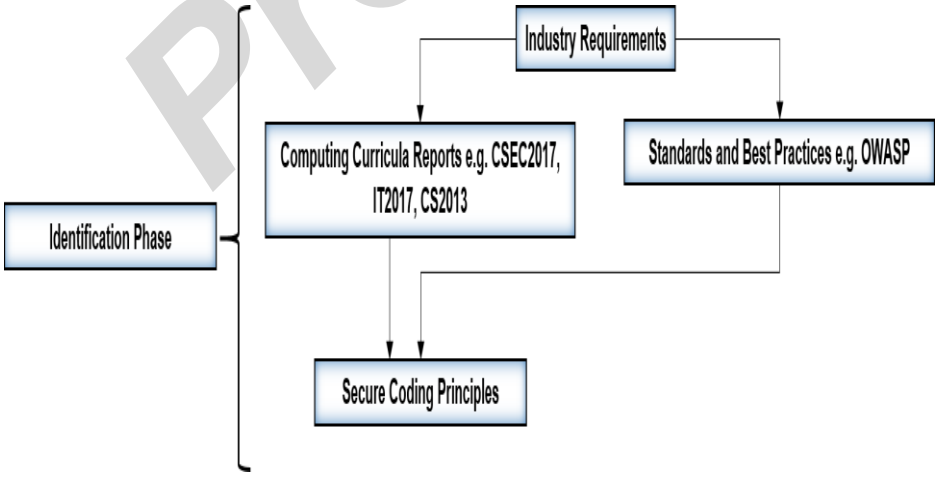


**Fig. 1.** The Identification Phase.

It is therefore necessary that universities take cognisance of this when identifying which secure coding principles to integrate into their programming modules. Currently, the secure coding principles as defined by [7, 8, 9, 12] are deemed to be reputable. Furthermore, universities should refer to the corresponding ACM curricular reports depending on what qualifications the university offers (for example, Computer Science, Information Technology, Information Systems). Fig. 1 depicts the Identification Phase encompassing these key elements, resulting in the identification of relevant secure coding principles.

### 5.2 The Buy-In Phase

The Buy-In phase is the second phase and it requires staff in university computing faculties to take the initiative in choosing the secure coding principles to be used as part of the learning outcomes of undergraduate programming modules. The initiative would need to be both a top down approach and also a collaborative one as it requires the cooperation and buy-in of departmental leadership and the lecturers.


**Fig. 2.** Direct Control in a Typical University Faculty [25].

The initiative to integrate secure coding principles into the various programming modules must be supported and initiated by computing faculty leadership within each university. The buy-in of leadership is crucial for the successful integration of secure coding principles into undergraduate programming modules. [25] states that in the university context, the Director of School would typically be at the strategic level, with the Head of Department as part of the leadership at a tactical level and lecturers at the operational level as shown in Fig. 2.

Furthermore, faculty staff need a mechanism for determining which secure coding principles should be integrated into which programming modules and at the appropriate year of study. The understanding of pervasive themes would guide a faculty by the understanding that each secure coding principle can be taught repeatedly across different course levels but with a different focus at each level. Table 2 shows an example of how a university department can create a checklist for formally integrating secure coding principles into undergraduate programming modules in each year of study.

**Table 2.** Checklist for Integrating Secure Coding Principles into Various Years of Study.

| **Secure Coding Principle** | **1st Year Modules** | **2nd Year Modules** | **3rd Year Modules** |
|---|---|---|---|
| Input Validation | ✓ | ✓ | ✓ |
| Authentication and Password Management | | ✓ | ✓ |
| Session Management | | ✓ | ✓ |
| Access Control | | ✓ | ✓ |
| Cryptographic Practices | | | ✓ |
| Error Handling and Logging | ✓ | ✓ | ✓ |
| Data Protection | | ✓ | ✓ |
| Database Security | | | ✓ |
| File Management | | ✓ | ✓ |

A secure coding checklist forms part of this phase due to the need for a formal structure for determining learning outcomes for each level of study within undergraduate programming modules. Each department would need to customise such a checklist according to prior knowledge of their students as well as the structure of their specific curriculum. To use the Input Validation as an example, the JavaScript code segment below shows part of what can be taught to students when teaching Input Validation [26].

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

Fig. 3 depicts the Buy-In Phase and how Direct Control in university faculties can be used to encourage the use of a secure coding principles checklist and the development of learning outcomes for each secure coding principle identified. The secure coding principles used in this phase would be sourced from the Identification Phase as agreed by the various faculty stakeholders. It would then be necessary to determine appropriate learning outcomes (LOs) for each level at which each secure coding principle will be taught as explained in the Implementation Phase.

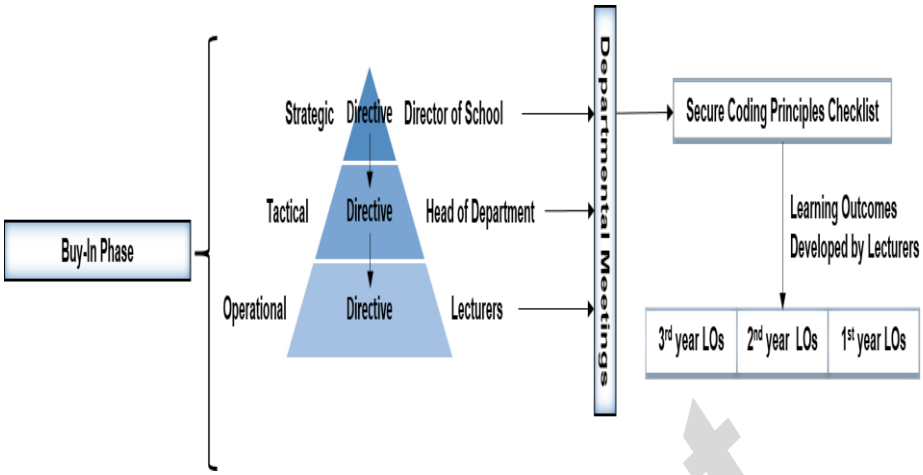


**Fig. 3.** The Buy-In Phase.

### 5.3 The Implementation Phase

The revised version of Bloom's Taxonomy [27] defines this taxonomy as a multi-level model arranged according to six cognitive levels of complexity of classifying thinking. For the sake of simplicity, this paper uses the bottom three levels of Bloom's Taxonomy to illustrate how secure coding principles can be aligned to each of the three years of study in terms of the cognitive levels of the content to be assimilated. [28] state that *"When each topic is presented with a Bloom-level of mastery, the instructor is better informed as to what level of mastery is expected; thus he will be able to determine the required time and necessary instruction to help the student achieve the proper knowledge level for each topic*". This suggests that Bloom's Taxonomy can be used to determine learning outcomes for undergraduate programming modules within each year of study as shown in Fig. 4.

The three bottom levels are defined as follows [27]:

- **Remembering**: Refers to retrieving, recognising, and recalling relevant knowledge from long term memory.
- **Understanding**: Refers to constructing meaning from oral, written and graphic messages through interpreting, exemplifying, classifying, summarising, inferring, comparing, and explaining.
- **Applying**: Refers to carrying out or using a procedure through executing or implementing.

The bottom three levels of Bloom's taxonomy can assist lecturers with devising learning outcomes that can be relatively understandable for undergraduate programming students to grasp. The complexity of learning secure coding principles might become too difficult for students within the scope of a three year undergraduate degree as the cognitive levels of Bloom's Taxonomy reach the three highest levels.

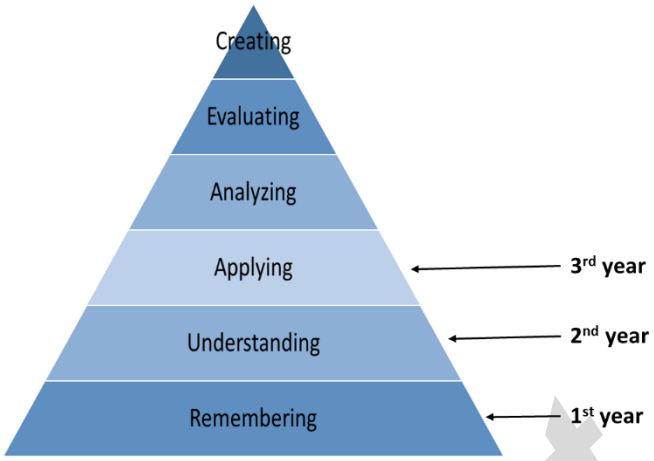


**Fig. 4.** Bloom's Taxonomy [27].

These levels could be addressed in fourth year and postgraduate studies. Table 3 highlights the relationship between the bottom three levels of Bloom's taxonomy and the possible learning outcomes that can be implemented in each undergraduate year of study.

**Table 3.** Input Validation Learning Outcomes Using Bloom's Taxonomy.

| **Bloom's Taxonomy Level** | **Secure Coding Principle – Input Validation Learning Outcomes** |
|---|---|
| 1st year – Remembering | **LO1**. State why input validation is important.<br>**LO2**. Define and provide examples of input validation. |
| 2nd year – Understanding | **LO1**. Determine the types of input validation output in various code segments<br>**LO2**. Explain how input is validated in various C# code segments. |
| 3rd year – Applying | **LO1**. Write C# code segments to validate input.<br>**LO2**. Modify C# code to suit Input Validation secure coding principles. |

Table 3 provides an example (Input Validation) to illustrate how the learning outcomes for each identified secure coding principle could be determined by using appropriate levels of Bloom's Taxonomy. Since the complexity of secure coding principles varies at each level, the assessment at each level of teaching must also differ. Although this paper does not map out all detail relating to all secure coding principles, it uses Bloom's Taxonomy to demonstrate the need for teaching and assessing secure coding at different levels according to the year of study, taking into account the context and prior knowledge of the students.

The integration of secure coding principles through the pillars-first approach allows students to learn secure coding principles in classroom settings and be assessed through periodic assignments and tests. However, the practicality of secure coding requires that the assessment of a student's learning progress be also assessed outside the formal classroom setting. According to [7], the IT curriculum must provide students with a capstone experience that gives them a chance to apply their skills and knowledge to solve a challenging problem. The capstone project in undergraduate programming courses provides an opportunity for students to practically apply secure coding principles they would have been taught. Fig. 5 depicts the Implementation Phase as it relates to the actual teaching and assessment of the identified secure coding principles. This study therefore proposes using Bloom's Taxonomy and the capstone project for the integration and assessment of secure coding principles respectively. The capstone project developed in the third year of study provides an opportunity for the practical assessment of secure coding principles learnt during the first, second and third years of study.

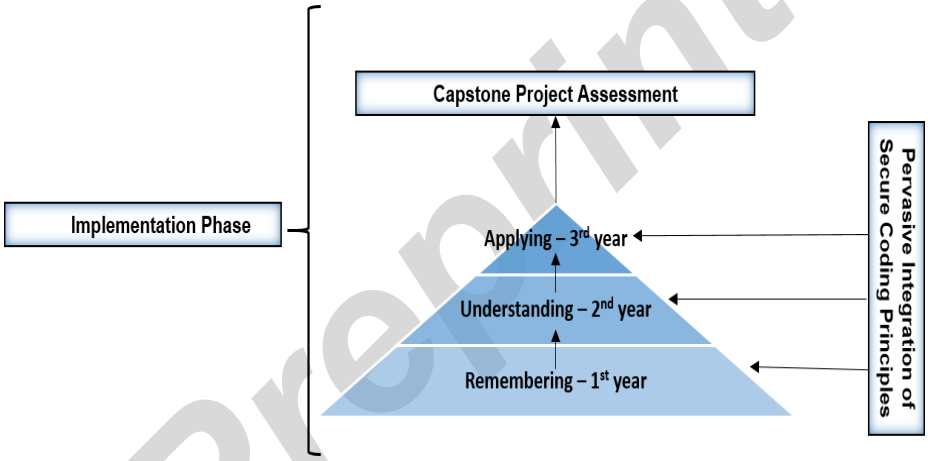


**Fig. 5.** Implementation Phase.



Fig. 6 presents the proposed framework comprising all three phases as discussed in this section. The proposed framework shows how the three phases are linked to each other within the context of integrating secure coding principles into undergraduate programming modules. From this figure it is clear that the secure coding principles identified in the Identification Phase are approved and agreed upon by various computing faculty memebers at the Buy-In Phase. The lecturers of the various programming modules are responsible for determining the learning outcomes related to each secure coding principle according to the level of study at which they teach. These learning outcomes are then used as a basis for teaching and assessment during the Implementation Phase, with the capstone project at third year providing the final assessment of secure coding.

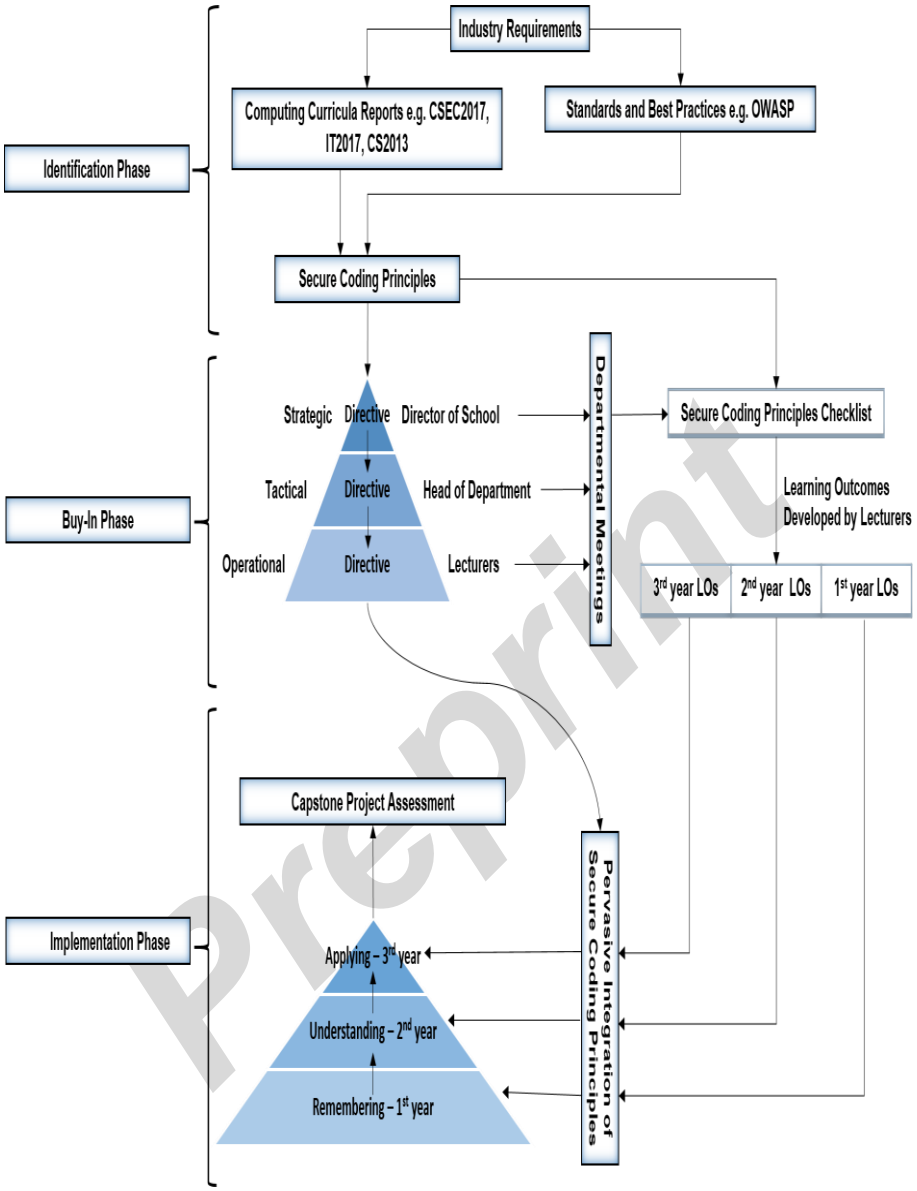




**Fig. 6.** Proposed Framework for Integrating Secure Coding Principles into Undergraduate Programming Modules.

## 6 Discussion and Conclusion

There is extensive evidence from literature that indicates that secure coding in undergraduate programming modules is still not extensively taught. This literature emphasises the need for computing faculties in universities to consider integrating secure coding into their undergraduate programming modules to ensure that, upon graduation, these students can meet the software security needs of industry. The various ACM Computing Curricula reports [7, 8, 9, 24] affirm this by further emphasing the need for secure software development.

The problem addressed by this paper is the general lack of formal inclusion of secure coding into undergraduate programming modules. The proposed solution to this problem is the development of a framework to assist computing faculty members in this regard. The proposed framework incorporates a three-phased approach including: an Identification Phase, a Buy-In Phase and an Implementation Phase.

The Identification Phase of the framework suggests that the ACM Computing Curricula reports can be used to justify the need for teaching secure coding principles in undergraduate programming courses since they consider the changing needs of industry. This phase specifically assists with the identification of current standards and best practices related to secure coding by referring to the work of organisations such as OWASP.

The Buy-In Phase depicts how multiple computing faculty stakeholders must be considered to ensure buy-in at strategic, tactical and operational levels within the faculty. It includes the use of checklists to ensure that the identified secure coding principles are addressed in multiple programming modules at various levels of study.

The Implementation Phase ensures the pervasive integration of secure coding principles into undergraduate programming modules. The use of Bloom's Taxonomy in this phase ensures that learning outcomes are appropriate for the relevant year of study. In addition, it is suggested that the capstone project be used to ensure that undergraduate programming students learn to practically apply secure coding principles as well as being assessed on them.

The purpose of the framework is to guide the integration of secure coding principles into undergraduate programming modules by illustrating the key components and role players to be considered. The current limitation of this research is that it has not yet been validated through the actual implementation of the framework. Further research will consider implementing this framework at various universities within South Africa to encourage the effective integration of secure coding principles into undergraduate programming modules.

## References


1. Aziz, N., Shamsuddin, S., Hassan, N.: Inculcating Secure Coding for Beginners. In: International Conference on Informatics and Computing (ICIC). Mataram, Indonesia (2016).
2. Ingham, K.: Implementing A Successful Secure Coding Continuing Education Curriculum For Industry: Challenges and Successful Strategies. In: Proceedings of the 19th Conference on Software Engineering Education and Training Workshops. Washington D,C (2006).
3. Dark, M., Ngambeki, I., Bishop, M., Belcher, S.: Teach the Hands, Train the Mind. A Secure Programming Clinic. In: Proceedings of the 19th Colloquium for Information System Security Education. Las Vegas, NV (2015).
4. Taylor, B., Bishop, M., Hawthorne, E., Nance, K.: Teaching secure coding: the myths and the realities. In: Proceeding of the 44th ACM technical symposium on Computer science education. (2013).
5. ACM, About the ACM Organization, https://www.acm.org/about-acm/about-the-acm-organization, last accessed 2019/03/12.
6. ACM, Key ACM Education Activities, https://www.acm.org/education/about-education. last accessed 2019/03/12.
7. IT2017 Task Group.: Information Technology Curricula 2017. ACM (2017).
8. The Joint Task Force on Computing Curricula.: Computer Science Curricula 2013. ACM (2013).
9. Burley, D., Bishop, M., Buck, S., Ekstrom, J., Futcher, L., Gibson, D.: Joint Task Force on Cybersecurity Education 2017. ACM, IEEE (2017).
10. Dark, M., Stuart, L., Ngambeki, I., Bishop, M.: Effect of the secure programming clinic on learners' secure programming practices. UC Davis, California (2016).
11. Agama, E., Hongmei, C.: A framework for teaching secure coding practices to STEM students with mobile devices. In: Proceedings of the 2014 ACM Southeast Regional Conference, Kennesaw, Georgia (2014).
12. OWASP.: OWASP Secure Coding Practices Quick Reference Guide. (2010).
13. Techopedia, Data Validation, https://www.techopedia.com/definition/10283/data-validation, last accessed 2019/03/05
14. Choudhury, A., Kumar, P., Sain, M.: A Strong User Authentication Framework for Cloud Computing. In: Asia -Pacific Services Computing Conference, Jeju Island, South Korea (2011).
15. Visaggio, C., Blasio, L.: Session Management Vulnerabilities in Today's Web. IEEE Security and Privacy 8, 48-56 (2010).

16. Techopedia, Access Control, https://www.techopedia.com/definition/5831/access-control, last accessed 2019/03/05.
17. Duong, T., Rizzo, J.: Cryptography in the Web: The Case of Cryptographic Design Flaws in ASP.NET. In: IEEE Symposium on Security and Privacy, Berkeley, CA, USA (2011).
18. Techopedia, Error Handling, https://www.techopedia.com/definition/16626/error-handling, last accessed 2019/03/05.
19. Sadeghi, A., Wachsmann, C.: Security and privacy challenges in industrial Internet of Things. In: 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA (2015).
20. Kindy, D., Pathan, A.: A Survey on SQL Injection: vulnerabilities, attacks, and prevention techniques. In: 15th International Symposium on Consumer Electronics, Singapore, Singapore (2011).
21. Techopedia, File, https://www.techopedia.com/definition/7199/file, last accessed 2019/03/05.
22. Aratyn, T., Kzerooni, S.: Secure Web Application Framework Manifesto. Security Compass (2010).
23. Whitney, M., Richter, L., Chu, B., Jun Z.: Embedding secure coding instruction into the IDE: A field study in advanced CS Course. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, Missouri, USA (2015).
24. Joint Task Force on Computing Curricula.: Curriculum Guidelines for Undergraduate Degree Programs in Information Technology. ACM, IEEE, (2008).
25. Gomana L.: Towards a Framework for the Integration of Information Security into Undergraduate Computing Curricula. Nelson Mandela University, Port Elizabeth (2017).
26. W3schools, JavaScript Form Validation, https://www.w3schools.com/js/js_validation.asp, last accessed 2019/03/11.
27. Forehand M.: Bloom's Taxonomy - From emerging perspectives on learning, teaching and technology. Georgia (2011).
28. Starr, C. W., Manaris, B., Stalvey R.: Bloom's Taxonomy Revisited: Specifying Assessable Learning Objectives in Computer Science. In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE, Portland (2008).