Journal of Computer Science

Original Research Paper

A Framework for a Multi-Layered Security of an Automated Programming Code Assessment Tool

[1]Mohd Fadzli Marhusin, [1]Muhammad Firdaus Zul Kafli, [2]Rossilawati Sulaiman, [1]Shaharudin Ismail and [1]Zul Hilmi Abdullah

[1]Faculty of Science and Technology, Universiti Sains Islam Malaysia (USIM), Malaysia
[2]Faculty of Information Science and Technology, The National University of Malaysia, Malaysia

Article history
Received: 03-02-2014
Revised: 03-06-2014
Accepted: 16-09-2014

Corresponding Author:
Mohd Fadzli Marhusin
Faculty of Science and Technology, Universiti Sains Islam Malaysia, Malaysia
Email: fadzlimarhusin@gmail.com

Abstract: In a learning environment, a low student-lecturer ratio is considered a practical solution by many educational institutions. However, the number of students in information technology is increasing every year. This could lead to a significant increase in the workload of lecturers, who need to evaluate assignments, quizzes and projects. Hence, it is desirable that an automated assessment tool is used to lessen their workload. In the era where mobile devices are getting popularity, the high demand to execute suitable quality code is there and the cost is on the processing power of the CPU which has a direct implication on the power source or the battery used. With various implementations of cryptography algorithms available, many of them could satisfy different level of needs. In this research, we introduce the architecture for a multi-layered security of automated assessment of programming code. First, we review the existing research studies in the area. We describe the features of the tool, as part of a complex e-learning environment. We also discuss the implementation of security, to protect data transmission and storage used by the tool. Challenges the system might face and the potential solutions, are also described.

Keywords: Automatic Assessment, Automatic Grading, E-learning Security

Introduction

A low student-lecturer ratio is considered ideal in learning environments (Bloom, 1984). However, the number of students studying information technology is increasing each year and has a negative impact in terms of workload and efficiency (Jackson and Usher, 1997). If insufficient time is available to evaluate assignments, quizzes and projects, it will affect the quantity of knowledge that students are able to absorb. An automated assessment tool could alleviate the increased workload.

Hence, the motivation of work in this area is to assist lecturers to overcome the problems so that a more quality time could be spent by lecturers to their students and at the same time the quality of lecture and assessment could be improved. The assessment outcome could be delivered almost in an instant, thus, allowing immediate feedback to the students, enhancing their understanding on the particular subject they are learning. All the aforementioned objectives are possible to achieve via the use of Automatic Assessment System (AAS).

E-learning is gaining headway, in line with development and innovation of the World Wide Web and increased significance of the e-learning 2.0 phenomenon (Downes, 2005). There are currently some open-source e-learning systems, such as Moodle, course manager, class web 2.0 and Sakai 2.0 (Amant and Still, 2007). With advances in the user interface of web 2.0 and standard features in many e-learning systems, opportunities are opened up for developers and researchers to develop compatible tools.

Delivering a programming course via an e-learning system such as Moodle (2013), is quite a challenge. Although tests and quizzes in an objective-based format (true/false or multiple-choice questions) can easily be implemented, an evaluation and assessment of the format of programming code would require a more dynamic technique. This is due to the fact that the programming code applied by an individual student can be unique, as various code formats will produce the required response to the question posed and produce the anticipated output. Using this tool, a lecturer would not be affected by an increase in the number of students taking a programming subject. The tool would automatically evaluate assignments, quizzes and tests submitted by the students, by comparing the code submitted with an answer scheme, which could cover one or more possible solutions. A full report may contain student profiles, scores and feedback, to enable the students to receive an assessment more quickly and, improve their understanding of the subject area they are studying.

In this study, we discuss a framework for an agent-based multi-layered security of an automated assessment tool. We begin with a review of the background to the study. The architecture of the system, covering its essential components, is discussed. A detailed description is given of the major role of functional agents, which comprise several agents that represent operations involving users, as well as the key components of the system.

An information system where security and privacy are not properly handled would be unacceptable in current circumstances, where information theft is prevalent. Hence, security and privacy are empirical issues that must be addressed in the design of any information system. The concept of multi-level user security based on different data sensitivities is described, showing the levels of encryption strength that offer various degrees of confidentiality for different users. Not all of the users require very strong encryption, such as the students. In a network used by thousands of students, implementing the strongest encryption may impact on efficiency, such as network speed, which would be affected by the addition of encryption processing overhead.

We also discuss aspects of the assessment process and how the programming evaluation is achieved. The marking process is not only based on an output specified by the lecturer. Instead, it is based on the source code level, where reconstructions of code are implemented. Preceding the conclusion, there is a discussion of the threats the system might face and the potential solutions for those challenges.

Literature Review

Background of Automated Marking Tools

Over the past decades, there have been research studies that offer solutions for automated marking of programming code (Matt, 1994; Foxley *et al*., 1997; Masrom *et al*., 2009; Ashoo *et al*., 2013). The vision to have interesting system features has been discussed in the past. It is significant that technology was the only constraint to some of that development. In the early stages, the central focus was on the mechanics of automated marking and less focus was given to general protection or specific security for the features.

The unix-based Ceilidh, was developed by the Learning Technology Research group at Nottingham University and was first used in 1988, mainly designed to evaluate C and C++ assignments (Foxley *et al*., 1997). It could handle more than one submission made by each student for each assignment. This would enable students to improve their codes in order to get higher scores. In addition, its aim was to handle a range of different types of tests, such as assessing programme code, multiple choice questionnaires and evaluating constructed responses and essays. It presents the results of an analysis to the students in a transparent manner, so that they will not only get the score, but also the justification and evidence.

Kassandra, was a system designed to ease the burden of teaching assistants, by evaluating all programming assessments, for example, fortran, maple, MATLAB and oberon assignments (Matt, 1994). Its main purpose was to check the accuracy of programmes produced by students. The Kassandra system was based on the observation that a number of assignments could be easily and fairly evaluated. Accuracy was tested by comparing the student submissions with the results provided by instructors. This system would respond to errors made by the students. Kassandra also provided a feature for the students to check their own scores. All the students could access their assignments for correction or review. Security-wise, distinct programmes were created for students and lecturers. The results generated were stored in a log file, which was only accessible by lecturers.

CourseMarker (Foxley *et al*., 2001), was a flexible, secure and user-friendly system, that was developed at the University Nottingham, as a successor to the Ceilidh system (Higgins *et al*., 2003). CourseMarker had an object-oriented design, to overcome the weaknesses in Ceilidh. CourseMarker also had a

number of additional capabilities, for example, the assessment of diagram-based work. Lecturers were given a wide variety of statistical data as part of the results (Joy *et al*., 2000). Besides that, CourseMarker had a plagiarism detector in its system, where it compared the submission it marked with others produced by the rest of the students (Higgins *et al*., 2002). In the later version (Higgins *et al*., 2005), enhancements were made, including some security features.

The BOSS Online Submission System, was developed by an employee in the computer science department at Warwick university (UW, 2009). The system was designed to facilitate the delivery of online programming exercises and at the same time allowed each assessment to be evaluated immediately (Joy *et al*., 2005). BOSS offered a user-friendly system, where students could use this system to test their programming exercises. Once satisfied, they could securely submit their assignments to a specific lecturer (Douce *et al*., 2005). The BOSS system provided a form of preliminary checking when the assignment was submitted. The system was based on the comparison of the textual output (Joy *et al*., 2000). The BOSS system also provided a dedicated graphical user interface for students and lecturers. The latest version of the system included a web-based application. This would allow the lecturers to review submissions using a traditional web-browser. Similar to CourseMarker, the BOSS system also had a plagiarism detector (Douce *et al*., 2005).

Programming Contest control (PC^2), was developed at California State University, Sacramento (Ashoo *et al*., 2013). It is a popular system used to host programming competitions around the world. Virtual internet-based and site-based competition modes are possible. PC^2 lets the contestants submit the programming source code to the judge via a network. The judge then retrieves the source code, recompiles and executes the programme. PC^2 also offers automated marking, based on specific programme input and output. The system uses a cryptographic mechanism to provide protection on the competition data. The software provides disk files encryption and network traffic encryption.

Ceilidh, Kassandra, CourseMarker, the BOSS Online Submission System were examples of AASs evolved from various requirement of their owners. Lots of new generation of the software were introduced in recent years such as Web-CAT (Edwards and Perez-Quinones, 2008), Easyaccept (Sauve and Neto, 2008), VERKKOKE (Alstes and Lindqvist, 2007) to name a few. Each of these offers some features i.e., VERKKOKE introduced a feature that make it compatible with the existing popular Learning Management System (LMS) Moodle and Optima via SCORM package.

Petri *et al*. (2010) presented a systematic literature review of the more recent AASs. They argued that there were many systems developed by researchers, where the prototypes were mainly for internal use. Petri *et al*. (2010) highlights the need for a standard of auto marking systems. By having a standard, we can avoid from repeating developing researcher-oriented systems.

Hence, we are motivated towards improving the existing framework of AAS, highlighting what are desired features should be in the consideration for developing an open system with security in mind for educational purposes at higher learning institutions in the future. In this study, we highlight several important design and security issues including on the need for multi-layered security implementation on the system.

Multi-agent Systems

Much traditional software design was a host-based architecture and later moved into client/server based. An multi-agent approach (Marhusin *et al*., 2008; Sulaiman *et al*., 2011) offers a more dynamic social-oriented based communications.

The features of Multi-Agent Systems (MAS), have within the frame of their objectives the ability to be proactive, reactive, social, truthful, benevolent, adaptive, autonomous and rational (Bellifemine *et al*., 2007; Weiss, 2013) are the reasons for adopting this approach in software systems (Soh *et al*., 2004; Masrom *et al*., 2009). MAS is a multi-platform environment, in which an agent can be added or removed with minimal impact to the system.

An agent-based automated programme code assessment tool was introduced (Masrom *et al*., 2009). They described the main components and the functional features of the system. The student, lecturer and assessment modules are the main components. The functionality of the system includes the roles played by some agent-based components, such as tasks related to the central agent, student and lecturer agents and the agent that performs the assessment.

The Proposed Architecture

Our proposed solution is called the Automated Programming Code Assessment tool (APCA) and is based on a multi-agent architecture. The solution can be developed as a full-fledged system on its own, or as a plugin for a large e-learning system, such as (Moodle, 2013). Integration with a mobile app to cater

for mobile users is also possible via the use of cross-device platform such as android base. Figure 1 shows the main architecture of APCA.

Based on Fig. 1, five main actors are identified. The administrator is only involved in system administration, such as initialising default data, maintaining records, creating top user accounts. The lecturer and the course coordinator are both designated lecturers, but the latter is created for a course taught by more than one lecturer. The tutor is the person who conducts the tutorial session which could be a student, a tutor, or the lecturer. The students are the people who enrol in the course.

The figure can be viewed from the functional agent perspective. The agents process and transfer data among them in a secure way, where the data and the system are viewed as being independent of each other via the application of cryptography.

We describe the major roles of functional agents and security issues they are concerned with, in the following sections.

The Components

APCA users are classified as (1) students, (2) tutors, (3) lecturers, (4) course coordinators and (5) administrators. The other components of the tool can be categorised into the marking and security-related operations, which include the question engine, the answer engine, the scoring engine and the security engine. Overall, there are nine parts in the system, which require careful design.

Major Roles of Functional Agents

We identified nine parts as the major functional components of the system. Each part will be transformed as an agent.

Student Agent

One student may have one tutor, belong to one tutorial group and also be under one larger lecture group. Each student will first need to authenticate, after which their programming code in answer to the problem assigned by their lecturer will be submitted.

Tutor Agent

A tutor will handle a tutorial session and this could also be the lecturer. Upon authentication, a tutor can post a question, supply an answer scheme and configure scoring details, which could be unique for that tutorial group.

Lecturer Agent

A lecturer may teach one or more programming subjects. For each subject, the lecturer may have several tutorial groups. Upon authentication, a lecturer can post a question, supply an answer scheme and configure scoring details.

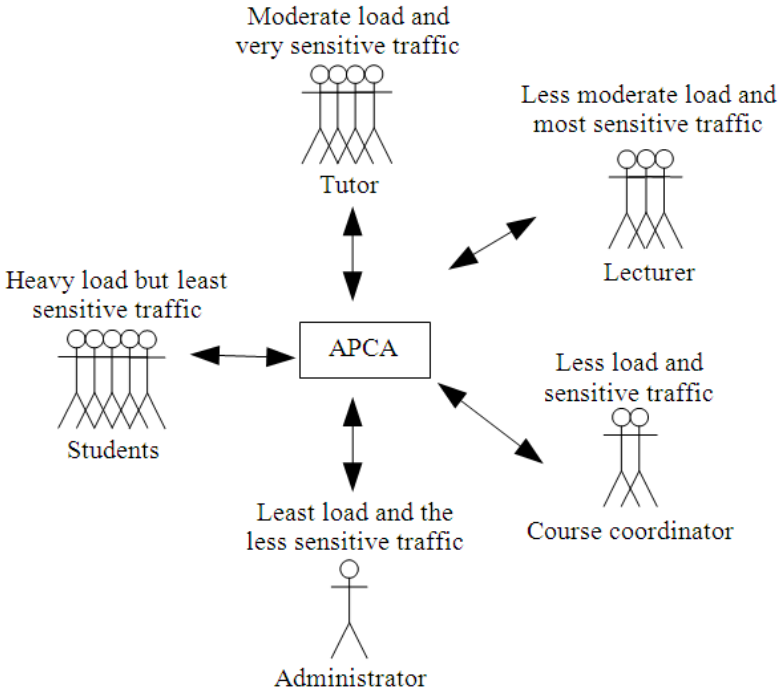

Fig. 1. The main architecture of APCA

Course Coordinator Agent

One course might be taught by many lecturers, either team teaching or not. After authentication, a course coordinator will have the power to synchronise the management of the assessment tool.

Administrator Agent

In the system, an administrator's main role is to set up the environment. Upon authentication, the administrator is responsible to pre-register students, tutors, lecturers, course coordinators and courses. Using data encryption, the administrator will not be able to view communication among other users, unless the administrator is the intended recipient.

The following agents are non-human and are not bound to any tutor, lecturer, course coordinator, or administrator.

Question Agents

The agents will collect questions and post them to the students. Questions could also be passed from coordinators, to lecturers, to students and to question agents accordingly.

Answer Agents

The agents will collect answer schemes and the score details and post them to the respective answer agents of lecturers and tutors. The answer agents could also pass the information from coordinators, to lecturers, to students and to answer agents accordingly.

Scoring Agents

These agents will reside on the trusted side of the team and they could be either at the host of any tutor, lecturer, or course coordinator and perform the scoring operations there.

Security Agents

These agents are responsible for authenticating and authorising users, such as students, tutors, lecturers and course coordinators. They also authenticate and authorise the operations performed by other agents. Security agents determine the strength of security required for each agent communication and also for storage.

Discussion

The Assessment Process

Foxley *et al*. (1997) described the types of evaluation possible in two ways: Static and dynamic.

Static Properties:

- Code layout
- Indentation
- Choice of identifiers
- Readability
- Comments
- Program structure
- Denotations
- Complexity
- Warnings
- Constructs

Dynamic Properties:

- Run against several test data sets
- Compare against other approaches to the same program
- Validate the output
- Measure the execution speed or performance

What was suggested by Foxley *et al*. (1997) were just a set of assessment facets and the sum of these could be used to gauge a performance. The output of the compiled version of the program will be compared with the schema provided. Marks are given depending on the results of the comparison by taking into consideration those facets

Inspired by their work, the marking process should evaluate both the static and dynamic properties of the student's programme, using certain metrics. A dynamic answer agent should be able to evaluate program code not merely based on a rigid expected programming output. The dynamic output is possible when the program expects some input from users or from other programs. Some dynamics could also happen when the program involve in using random engine or some artificial intelligence algorithms.

In the event where the solution to a programming problem could be dynamic, the answer agent should be able to construct several versions of potential solutions and would consider a solution by a student as a potential new solution as well. In the case where a submission of programme code is not correct or contains compile errors, the answer and scoring agents should work together to reconstruct the code to reach potential solutions and give a score based on some ratio, depending on the number of codes potentially generating correct solutions.

The Security

Kassandra (Matt, 1994) introduced security feature as a way to ensure a distinct access between lecturer

and students. To achieve that, a separate user interface is created with a method of authentication. Implementation by Foxley *et al*. (1997) did not describe its security features. In the later version called CourseMarker (Foxley *et al*., 2001) a secure feature was introduced. The aim was to ensure the execution of the code were done within a sandbox environment so that any harmful execution would be just within the contained environment.

Kassandra was among the earliest AAS (Matt, 1994) and its security features were limited to protection against unauthorised access and access rights on read and update records in the system.

Username and password is no longer a safe implementation (RSA, 2011), various additional approaches are needed to enhance security. Thus, various proposed systems in the literature employed some level of security features. Whether their implementation details are sufficient is subject to further discussions. We always play some trade-offs, a highly secure system might just have features that overkill the fundamental idea of having the system in the first place and raise usability issues.

Nevertheless, cryptography remains the most effective way of obstructing illegitimate access to data. At logon page, we have seen cryptography at work in captcha/recaptcha Google, 2014 and various one time-based token techniques (RSA, 2011).

PC^2 encryption implementation ensure data safe while in storage and in transit (Ashoo *et al*., 2013). However the strength of the encryption is not dynamic. What is desired here is the freedom to automatically choose the right strength dynamically, knowing that there are various constraints on devices.

The key size of the encryption algorithm is an important parameter. Giry (2014) gives a recommendation on the key sizes of the algorithm. The recommendation is based on a specific calculation theory using the number of years the key strength could guarantee protection. However, the formula is designed to resist mathematical attacks and attacks specially crafted with the aim to cut the time required while cracking the encryption key.

There are many choices of encryption algorithm that experts have proven to be reliable, which could be used in agent communications. We could obtain a set of varied key length from the algorithms as per Table 1. In order to ensure the performance penalty of the system is within the acceptable level, the performance evaluation need to be assessed and applied automatically by the system but with options to override the configuration.

Table 1. Encryption algorithms and possible key strength to be used

	Potential algorithms	Key strengths
1	AES	259-bit
2	AES	192-bit
3	AES	128-bit
4	3DES	168-bit
5	Blowfish	112-bit

Agents' communication is not restricted to any private or public network, which means students, tutors, lecturers and coordinators, can access the tool from anywhere within a local area network or the Internet using their credentials. In order to strengthen the communication channel and the data, multi-level security encryption is proposed, similar to previous evaluations (Sulaiman, 2010; Sulaiman *et al*., 2011).

A trade-off is necessary between the need to achieve a certain level of data confidentiality related to the auto marking mechanism and at the same time to manage the volume of traffic efficiently is necessary. Theoretically a stronger encryption algorithm and a longer key tend to have consequences in terms of speed and bandwidth. As the number of students varies, but most of the time there is a high volume, it is desirable to apply the lowest, yet effective encryption algorithm and key, so that the submission of programming code is secure. The volume of traffic generated by all students in an e-learning environment where the strongest available encryption algorithm and the longest possible key is used, compared to a situation without such an implementation, is something we would like to evaluate in subsequent research.

Communication between tutors/lecturers and lecturers/course coordinators, would require an elevated level of security, for example, asymmetric encryption algorithms and keys. Understandably, the information they deal with would require a suitable level of confidentiality. The number of this type of users in an e-learning environment is smaller. Thus, implementing the strongest encryption algorithms with some key length could be possible, without hampering the performance of the e-learning environment and the traffic of the network involved.

The same detailed security implementation is needed for the data storage. When the data needs to be stored on the same machine using the strongest security mechanism, it would not be a problem because the constraint is only on the CPU load to implement the encryption and decryption (Ferguson and Schneier, 2003). However, when the data needs to be transferred from one agent to another, involving a different host,

determination of a suitable encryption algorithm and key strength are required because each machine tend to have different processing power (Sulaiman, 2010; Sulaiman *et al*., 2011). In case of mobile devices, stronger encryption means extra load of processing requirement which definitely can drain the battery.

Threats/Challenges and Potential Mitigations/Solutions

With the implementation of data encryption to multi-level users, it is aimed that the system could provide secure data in transit and storage. The transfer of the data is by using the agent-based technique, which offers the ability to build much needed features such as data security, portability on data transmission and failover recovery, so as to ensure a failure on one agent will not affect the whole system. The following are among the threats to the system.

The User is not really the Account Holder

This could possibly happen when the user account has been stolen such as via brute force attack. It is where the illegitimate user already knows the targeted user-id and the password is guessed via brute force dictionary. However, we can protect the user account from this kind of attack by implementing an account lockout policy. An attempt to unlock the account would require a secondary communication medium, for example, an email. Another potential for account loss is if the user accesses a machine which has been installed with a key logger. We assume here that the user, before using the system, must activate some security mechanism, for example, an anti-virus programme that has been installed beforehand to detect malware. The implementation of ReCaptcha (Sor, 2013) would also help to frustrate the attacker.

Vulnerability Exploitation

If a machine has vulnerability and the attacker knows the kind of suitable exploits, there is a potential of remote shell execution happening. If this occurs, the machine is no longer considered safe, but the attacker also needs to break the encrypted data as well. The most critical part in the system is the central server, which keeps the authoritative data. Even though other computers can keep the same copy of a particular record, there must be at least one copy that is marked as the primary source.

One potential solution is to make sure data is replicated to other hosts as well and the moment the record is saved, its integrity is checked and recorded. There could be an Integrity Agent, which has the role to check the integrity of data files in all participating hosts. Inevitably, we must rely here on the assumption that it is unlikely that a particular file is corrupted due to vulnerability exploitations if all hosts have the same hash value, unless the hacker must access all participating hosts and update the data.

There are also security issues which is considered very fundamental when recently a serious security vulnerability in the open SSL library called the heart bleed was come to surface (Finkle, 2014; Heartbleed.com, 2014). With that level of vulnerability any computers implementing whatever encryption algorithm, as long as the encryption relies upon the library, they are virtually susceptible to attack that can harvest the encryption key from the memory.

Solving a vulnerability require us to patch it so that no one else could exploit it. The gap between a vulnerability discovered and patched remains the main problems that require fixing. Auto update sometime requires software or system reboots that might prohibit user from choosing this option. Alternatively, we require a mechanism such as a detection model which is described in (Marhusin, 2012) to detect any attempts to exploit vulnerable software or system as any code will require execution and/or transmission.

Admin Privilege Exploitation

Assuming the privacy of the data in the system is fully safe from an unintended user, the next threat is only when an administration privilege is misused to delete a user account. If the cascade update and cascade delete in the database is implemented, a deletion of primary key data will result in a total deletion of all records associated to that primary key. A disaster like this could be prevented by implementing a strict procedure for any request to delete a user account.

API Hooking

Assuming there is no malware in the machine, the system might be safe. However, if the machine is infected with some malware capable of API hooking, the use of encryption is grossly defeated. This is because the malware could capture the plaintext at the lower layer, as the encryption process is implemented at the higher layer. Note that there are various reasons for API hooking are used legitimately. Thus, it is challenging task to detect any malicious API hooking.

Rogue Agents

Dubious agent in an agent environment is serious issue software developers must deal with. With a

lacking agent security, attacker could intercept traffics and replay the series of traffics at later time. Even if the attacker could not reveal the traffic, by chance, the play could possibly trigger a data integrity disaster in a data-critical related system. Protection against this attack is possible by means of strong authentication and authorization checking.

The authentication of agents can be related to what the end-user use or something generated by the system itself in a unique form. The authorization is important to ensure no rouge agent executing command not entitled for it to execute. Ferraiolo and Kuhn (1992) suggested that are three generic rules are possible related to authorization, namely in term of role assignment, role authorization and transaction authorization. From agent point of views the following rules may applicable:

- An agent can only execute *x* only if the agent has been assigned to a specific role
- If the agent's main role is to execute *x*, it must be authorized to do that only and should not execute *y* which is belongs to other distinctive agent roles
- The agent can execute *x* in which *x* is authorized to be executed under a particular role that the agent is classified into

In term of operation and implementation, there are expected that more computing resources required running this tool due to more features involved. Agent-based system with embedded security function such as cryptography and authentication features required more processing power and memory.

Conclusion

In this study we discussed a number of systems, related to automated programming code assessment and their limitations. We also proposed our architecture, called the automated programming code assessment tool for use in an e-learning environment. The proposed tool is developed using agent-based architecture and incorporates multi-level user security, which uses asymmetric encryption algorithms to strengthen security during data transmission and storage. The implementation of these encryption algorithms would vary, depending on the role of the various users such as students, tutors, lecturers, course coordinators and administrators. We also explained several threats the system might face and the potential solutions to these challenges. Some of these elements require separate area of research i.e., malware and intrusion detection systems.

The proposed architecture is limited to programming code based assessment. We did not discuss the capability in non-programming-code-based evaluation mechanism.

Our next research in this topic is to extend the capability of the system to deal with comprehension-based questions too. We intend to integrate the existing cognitive domain theories into the assessment mechanism to ensure that the questions are up to a certain standard and we will measure the quality and accuracy of such system.

Funding Information

The authors have no support or funding to report.

Author's Contributions

All authors equally contributed in this work.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Alstes, A. and J. Lindqvist, 2007. Verkkoke: Learning routing and network programming online. Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Jun. 23-27, ACM, Dundee, Scotland UK, pp: 91-95. DOI: 10.1145/1268784.1268813

Amant, K.S. and B. Still, 2007. Handbook of Research on Open Source Software: Technological, Economic and Social Perspectives. 1st Edn., Idea Group Inc (IGI), Hershey PA, ISBN-10: 159140892X, pp: 728.

Ashoo, S.E., T. Boudreau and A.L. Douglas, 2013. Welcome to the PC^2 home page.

Bellifemine, F.L., G. Caire and D. Greenwood, 2007. Developing Multi-Agent Systems with JADE. 1st Edn., John Wiley and Sons, Chichester, England, ISBN-10: 0470058404, pp: 300.

Bloom, B.S., 1984. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. Educ. Res., 13: 4-16. DOI: 10.3102/0013189X013006004

Douce, C., D. Livingstone and J. Orwell, 2005. Automatic test-based assessment of programming: A review. ACM J. Educ. Resources Comput., 5. DOI: 10.1145/1163405.1163409

Downes, S., 2005. E-learning 2.0. eLearn, 10: 1-1. DOI: 10.1145/1104966.1104968

Edwards, S.H. and M.A. Perez-Quinones, 2008. Web-CAT: Automatically grading programming assignments. Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education, Jun.-Jul. 30-02, ACM, New York, pp: 328-328. DOI: 10.1145/1384271.1384371

Ferguson, N. and B. Schneier, 2003. Practical Cryptography. 1st Edn., Wiley, Indianapolis, ISBN-10: 0471223573, pp: 410.

Ferraiolo, D.F. and D.R. Kuhn, 1992. Role-Based Access Controls. Proceedings of the 15th National Computer Security Conference, Oct. 13-16, Baltimore, MD, pp: 554-563.

Finkle, J., 2014. Little Internet users can do to thwart 'Heartbleed' bug.

Foxley, E., C. Higgins, E. Burke, C. Gibbon and A.M. Zin, 1997. The Ceilidh system an overview and some experiences of use. Proceedings of the Asian Technology Conference in Mathematics.

Foxley, E., C. Higgins, P. Symeonidis and A. Tsintsifas, 2001. The coursemaster automated assessment system-a next generation Ceilidh. Proceedings of the Conference on Computer Assisted Assessment to support the ICS Disciplines, University of Warwick.

Giry, D., 2014. BlueKrypt.

Heartbleed.com, 2014. The Heartbleed Bug. Heartbleed.com.

Higgins, C., T. Hegazy, P. Symeonidis and A. Tsintsifas, 2003. The course marker CBA system: Improvements over ceilidh. Edu. Inform. Technol., 8: 287-304. DOI: 10.1023/A:1026364126982

Higgins, C., P. Symeonidis and A. Tsintsifas, 2002. The marking system for course master. Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education, Jun. 24-26, ACM New York, pp: 46-50. DOI: 10.1145/544414.544431

Higgins, C.A., G. Gray, P. Symeonidis and A. Tsintsifas, 2005. Automated assessment and experiences of teaching programming. ACM J. Educ. Resources Comput., 5: 3-3. DOI: 10.1145/1163405.1163410

Jackson, D. and M. Usher, 1997. Grading student programs using ASSYST. Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education, Feb-Mar. 27-01, San Jose, ACM New York, pp: 335-339. DOI: 10.1145/268085.268210

Joy, M., P.S. Chan and M. Luck, 2000. Networked submission and assessment. Proceedings of the 8th Annual Conference on the Teaching of Computing LTSN Center for Information and Computer Sciences, (ICS' 00), University of Southampton, pp: 335-339.

Joy, M., N. Griffiths and R. Boyatt, 2005. The BOSS online submission and assessment system. J. Educ. Resources Comput. DOI: 10.1145/1163405.1163407

Marhusin, M.F., 2012. Improving the effectiveness of behaviour-based malware detection. PhD, Thesis, University of New South Wales, Canberra, Australia.

Marhusin, M.F., D. Cornforth and H. Larkin, 2008. An overview of recent advances in intrusion detection. Proceedings of the 8th IEEE International Conference on Computer and Information Technology, Jul. 8-11, IEEE Xplore Press, Sydney, NSW, pp: 432-437. DOI: 10.1109/CIT.2008.4594714

Masrom, S., A.S.A. Rahman and A.S. Shafie, 2009. Computer assisted assessment for computer programming course with agent based architecture. Proceedings of the 8th WSEAS International Conference on Telecommunications and Informatics (CTI' 09), Istanbul, Turkey, pp: 21-25.

Matt, U.V., 1994. Kassandra: The automatic grading system. SIGCUE Outlook, 22: 26-40. DOI: 10.1145/182107.182101

Moodle, 2013. Moodle.org: Open-source community-based tools for learning-Mozilla Firefox.

Petri, I., A. Tuukka, V. Karavirta and O. Seppälä, 2010. Review of recent systems for automatic assessment of programming assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Oct. 28-31, Koli, Finland, ACM New York, pp: 86-93. DOI: 10.1145/1930464.1930480

RSA, 2011. Why passwords aren't strong enough. Making the case for strong authentication, EMC Corporation.

Sauve, J.P. and O.L.A. Neto, 2008. Teaching software development with ATDD and easy accept. Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education Mar. 12-15, New York, USA, ACM, pp: 542-546. DOI: 10.1145/1352135.1352317

Soh, L.K., H. Jiang and C. Ansorge, 2004. Agent-based cooperative learning: A proof-of-concept experiment. Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, Mar. 03-07, Norfolk, Virginia, ACM New York, pp: 368-372. DOI: 10.1145/971300.971427

Sor, 2013. RECAPTCHA library for Java 0.0.7.

Sulaiman, R., 2010. MAgSeM: A multi-agent security framework for secure cyber services. PhD Thesis, University of Canberra, Australia.

Sulaiman, R., D. Sharma, W. Ma and D. Tran, 2011. A new security model using multilayer approach for E-health services. J. Comput. Sci., 7: 1691-1703. DOI: 10.3844/jcssp.2011.1691.1703

UW, 2009. BOSS online submission system. University of Warwick.

Weiss, G., 2013. Multiagent Systems. In: Intelligent Robotics and Autonomous Agents Series, Weiss, G., (Ed.)., MIT Press, Cambridge, Massachusetts, ISBN-10: 0262018896, pp: 867.