TAFFIES: Tailored Automated Feedback Framework for Developing Integrated and Extensible Feedback Systems

Matthew Pike · Boon Giin Lee · Dave Towey

Received: 30 October 2021 / Accepted: 13 January 2022 / Published online: 10 February 2022
© The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd 2022

Abstract
Delivering high-quality, timely and formative feedback for students’ code-based coursework submissions is a problem faced by Computer Science (CS) educators. Automated Feedback Systems (AFSs) can provide immediate feedback on students’ work, without requiring students to be physically present in the classroom—an increasingly important consideration for education in the context of COVID-19 lockdowns. There are concerns, however, surrounding the quality of the feedback provided by existing AFSs, with many systems simply presenting a score, a binary classification (pass/fail), or a basic error identification (“The program could not run”). Such feedback, with little guidance for how to rectify the problem, raises doubts as to whether or not these systems can stimulate deep engagement with the related knowledge or learning activities. In this paper, we propose TAFFIES, a framework to scaffold the development of AFSs that promote high-quality, tailored feedback for student’s solutions. We tested our framework by applying it to develop an AFS to mark and provide feedback to 160 CS students in an introductory databases class. In contrast to most introductory-level coursework feedback and marking, which typically generate significant student reaction and change requests, our AFS deployment resulted in zero grade challenges. There were also no identified marking errors, or suggested inconsistencies or unfairness. Student feedback on the AFS was universally positive, with comments indicating an AFS-related increase in student motivation. The experience of designing, deploying, and evolving the AFS using TAFFIES is examined through reflective practice, student evaluation, and focus group (involving peer teachers) analysis.

Keywords Automated feedback systems · Feedback · Assessment · Teaching and learning

Introduction
The impact of the COVID-19 pandemic caught a number of higher education (HE) institutions (HEIs) unprepared, requiring them to rapidly rethink and revise their approach to teaching and learning (T&L) practices, particularly regarding assessment and the provision of feedback to students [1–3]. This is especially true for Computer Science (CS) educators, who rely on in-person interactions during practical laboratory sessions to provide guidance to students, who may be just starting to learn to program.

Programming is a complex activity, requiring learners to follow strict syntax rules, decipher complex (and often unintuitive) error reports, and develop correct logical statements and structures to solve a given problem or task [4]. Learning to program is a practice-led activity [5]. Students cannot learn to program by reading a textbook or attending lectures, alone. Practice is a critical factor in the acquisition of programming skills [6]. Laboratory sessions are not only an opportunity for CS students to write code, but also allow them to obtain immediate, contextual feedback on their solutions from teaching staff and peers. Often, this feedback is multi-modal, delivered using spoken explanations, referencing to code on-screen through finger pointing, and occasionally the direct editing of the student’s code to demonstrate the necessary remedial
action. The provision of feedback is known to be a critical factor in reinforcing a student’s learning [7]. It can also be the source of great dissatisfaction among students, who are reportedly often less satisfied with feedback than with any other feature of their courses [8]. COVID-19 has led to increasingly distributed and hybrid HE learning environments which challenge the traditional forms of feedback provision. Alternative approaches are therefore required.

Many CS educators have explored the use of Automated Feedback Systems (AFSs) for the provision of feedback to novice programmers [9–12] AFSs have many attractive properties—including timeliness and consistency—and can be an always-available resource, allowing students to receive formative feedback outside timetabled sessions. AFSs are not a silver bullet, however, as there are many shortcomings associated with their use. Students often remark that AFS feedback is not sufficiently detailed or specific, perhaps providing only a “pass” or “fail” indicator of performance, with little or no guidance about how to proceed to improve or fix their code [13]. Educators have also highlighted issues, including the adaptability of these systems for use in particular assessments, which often requires additional programming to overcome technical constraints imposed by the AFS itself [13].

Many AFS frameworks and software tools are available for educators to use [14–18]. In general, there is a strong focus on identifying the correctness of a student’s solution. Whilst the correctness of a given solution is important, the tailoring of appropriate feedback to the student must also be considered in the evaluation of an AFS’s success. In this paper, we explore an alternative approach to the development and deployment of an AFS. Specifically, we propose a framework for developing AFSs: Tailored Automated Feedback Framework for Developing Integrated and Extensible Feedback Systems (TAFFIES). TAFFIES structures the development of AFSs in a manner that promotes generating tailored, detailed feedback on student solutions. Using a TAFFIES-based approach to developing an AFS, educators only begin generating and building the feedback that students receive after the solutions have been submitted. This contrasts with conventional approaches, where the marking logic and feedback are typically prepared in parallel with the task specification. Additionally, TAFFIES places the emphasis on generating tailored feedback rather than simply evaluating the correctness of a solution (as is common with existing AFSs): This is achieved through the direct interrogation of student solutions as a part of the development process.

TAFFIES

In this section, we introduce TAFFIES as a conceptual framework to guide and support the development of AFSs. TAFFIES does not prescribe a particular set of technologies for how AFSs should be implemented. Rather, TAFFIES provides a structure to develop AFSs that generate high-quality feedback for students. TAFFIES is presented as an abstract framework, but supported by a concrete case study implementation. TAFFIES comprises three distinct components:

1. Feedback Signatures (FSs)
2. Assessment Criteria Specification (ACS)
3. Feedback Coverage Checker (FCC).

Each component contributes towards the functional structure of an AFS. Component development is performed using an iterative development cycle [19], as shown in Fig. 1.

Feedback Signatures

Feedback Signatures (FSs) are the core concept upon which TAFFIES-based AFSs are constructed. FSs are the lowest denominator of TAFFIES, and consist of a signature-feedback pairing. Each FS has a signature component, which is a pattern against which a student’s submission is evaluated. Should a student’s submission match an FS, the feedback component associated with that FS will be communicated to the student, in the feedback report. Should a student’s solution not match a given FS’s signature, then no output is generated. FSs, therefore, are independent, standalone operations which can operate on any given input, allowing them to be re-used between assessments when and where this would make sense. This property also allows FSs to be shared among educators, hinting at the potential for a shared, open assessment resource (OER) for developing AFSs [20].

In addition to directly generating FSs based on submissions, an AFS has two additional sources for FSs: (1) reuse of FSs from other AFSs (or OERs); and (2) educator-inspired FSs, reflecting anticipated deficiencies. It should be noted that the potential feedback generated from these two other groups of FSs will only become actual feedback if the signatures are matched in the student submissions. If the student submissions do not match these FS signatures, then that FS’s feedback is not activated.

Assessment Criteria Specification

The Assessment Criteria Specification (ACS) provides the mechanism to specify the assessment criteria of the assignment for which the AFS is being developed. An ACS is defined in a simple, declarative manner using code. The ACS relates one or more FSs to a given assessment criterion. The ACS may be specified in a fine-grained manner, where many FSs are assigned to a single ACS to represent the constituent properties from which a criterion is formed. The ACS also
provides a mechanism for allocating marks according to FS activations. An ACS is assignment-specific and would not typically be re-used between assignments, but their constituent FSs may be re-used.

**Feedback Coverage Checker**

The Feedback Coverage Checker (FCC) allows AFS developers to obtain a *feedback coverage metric*. The development of a TAFFIES-based AFS follows an iterative development process [19]. As solutions are received, FSs should be developed to provide specific feedback on the errors/successes present in the solutions. The FCC plays an important role in ensuring that all students receive suitable and sufficient feedback on their submission. The FCC generates a Feedback Coverage Report (FCR) that details, for each criterion, the number of students that have received one or more feedback item. Other metrics, such as average number of feedback items per student, are also reported. The FCC enables pinpointing student submissions that have not received any feedback for a given criterion. This, in turn, allows the developer to tailor an FS for that particular solution. Having done so, the FCC is re-run and a new FCR generated. Since a single FS may apply to several student submissions, this can be an effective way of rapidly and efficiently provisioning detailed feedback for many solutions.

**Example**

When interacting with the SQL database using Python (or any other programming language), it is good practice to use a specifically formatted query, called a “prepared statement”, ¹ that reduces the possibility of SQL Injection attacks [21]. This use of prepared statements, therefore, is considered good practice. It is common, however, for students to use a less secure string-based approach. A TAFFIES-based AFS can provide feedback for this type of deficiency: an FS containing a code signature identifying string-based solutions would be specified. The FS feedback component would inform students that this approach is insecure, and should be replaced with a prepared statement. The entire FS would then be inserted into the ACS and linked to the specific assessment criterion. The FCC would then be run, and the number of matched submissions analyzed.

**Inspiration**

Much of the TAFFIES structure and approach draw inspiration from existing software engineering methodologies. The specification of FSs before the actual running of the AFS borrows from the principles of Test-Driven Development (TDD) [22]: TDD requires a developer to first write a test for a particular unit of functionality, before writing the code to fulfill that functionality [23]. The test can then be used to verify that the implementation meets the functional requirements specified in the test. However, in TAFFIES, TDD is used to identify the correctness of student solutions and provide feedback accordingly. Students’ solutions are therefore viewed as the ‘units’ under test. Test coverage measures the amount of testing performed by a set of tests [24]: In the context of TAFFIES, this relates to the coverage of feedback provided to the set of student solutions.

An Agile, iterative development methodology is also at the heart of TAFFIES [25]. Agile methods are commonly

---

¹ SQLite Prepared Statement—http://www.sqlite.org/c3ref/stmt.html.
used in software engineering contexts and allow companies to rapidly prototype and improve with every iteration [26]. Teams can make quick corrections based on stakeholder feedback, for example, which are quickly integrated into the next iteration cycle. The ability to adapt quickly to new solution formations in student submissions is what makes Agile a suitable approach to developing AFSs. Since it is difficult to predict the types of errors that students will make when developing their solutions, having a development mechanism designed to deal with emerging knowledge is a suitable choice.

**Case Study**

In this section, we present our application of the TAFFIES framework to providing feedback to students for a programming-based assessment.

**Context**

The University of Nottingham Ningbo China (UNNC), the first Sino-foreign HE institution, was established in 2004 [27]. It was established as part of China’s ongoing investment in improving both the quantity and quality of its higher education provision [28], and is both an innovation and a centre of innovation [29].

The University of Nottingham delivers the same degree content across all of its campuses, of which UNNC is one, and University of Nottingham Malaysia Campus (UNMC) is the other international campus. The Nottingham degree programs are fully accredited by relevant professional organizations, including UNNC CS. During the first year of the Nottingham CS program, a strong focus is placed on developing students’ programming skills. The “Databases and Interfaces” (DBI) class is compulsory for all CS undergraduates at UNNC. DBI provides students with a general introduction to the theory and practice of database systems, with students learning to create and interact with databases using a Structured Query Language (SQL) [30]. Additionally, students learn to design and implement graphical user interfaces (GUIs) in a web-based context, interacting with standard web technologies such as HTML [31] and CSS [32]. The DBI class has run, under various names and formations, for over a decade at UNNC and is a well-established part of the CS curriculum.

In the aftermath of the COVID-19 pandemic, the DBI teaching staff revised and adapted the class for a hybrid delivery, which included approximately 30 remote-learning students and 130 in-person students. Before COVID-19, DBI relied heavily on a centralized technical infrastructure, situated on the university campus. This infrastructure provided students with access to an Apache web-server and a MariaDB database server, allowing DBI students to complete weekly laboratory tasks and coursework. This infrastructure worked well with on-campus students, providing a reliable and convenient resource [33]. However, a number of off-campus students, especially those outside of Mainland China, faced great difficulties obtaining a reliable connection to the campus network. Some technical limitations were insurmountable, creating a significant challenge to the university and great frustration to students. The class instructors decided, for 2020–2021, to transition to a local technical infrastructure, meaning that the software required for the participating in DBI would be run directly on the students’ personal computing devices. In doing so, the instructors were keen to use freely available and lightweight (small memory and hard-disk requirements) software solutions [34], being aware that some students were working on constrained computing devices.

SQLite was used as the database software. SQLite is a small, fast, self-contained, highly reliable, full-featured, SQL database engine. SQLite is freely available for all major operating systems, and is a small download and installation (<1MB), an important consideration for students with poor network connectivity. Apache was replaced by Flask, a lightweight web framework written in Python [35]. Flask provides functionality to serve static (HTML, CSS) and dynamic webpages using data stored in SQLite (and other) databases [36].

The first week of laboratory sessions were dedicated to ensuring that students’ machines were correctly configured, and capable of using the software. This process was relatively smooth, with few students encountering technical difficulties.

DBI is assessed through both a written examination and coursework. The coursework is split into two activities, with Coursework-1 consisting of multiple quizzes assessing knowledge and understanding, and Coursework-2 (CW2) focusing on the application of this knowledge. CW2 will be the focus of the remaining sections. For CW2, students were presented with a fictional (but realistic) scenario, in which they were to complete a partially implemented inventory-tracking system. Students were given a partial code-base and an SQLite database, upon which their solutions would be developed. The database was structurally complete, with everything necessary to support the application’s data requirements. The data entries stored in the database, however, were incomplete, and issue students would be required to address

---

2 Apache Web-server: [https://httpd.apache.org/](https://httpd.apache.org/).
3 MariaDB Database: [https://mariadb.org/](https://mariadb.org/).
4 SQLite Database: [https://www.sqlite.org/index.html](https://www.sqlite.org/index.html).
5 Flask Web Framework: [https://flask.palletsprojects.com/en/2.0.x/](https://flask.palletsprojects.com/en/2.0.x/).
6 Python Programming Language: [https://www.python.org/](https://www.python.org/).
All students received an identical initial (partially implemented) code-base, which they would then extend to meet the requirements of the coursework. The coursework involved standard web technologies that students had become familiar with through the DBI course (HTML, CSS, SQL, and Python). Students were given approximately 3 weeks to complete CW2. In addition to a task description, students were also provided with a detailed marking scheme. Students were required to complete the following three tasks for CW2:

**Task 1. Database Initialization** Students were provided with inventory data stored in a Comma-Separated Values (CSV) file. Task 1 asked students to populate the database using the data stored in the CSV file. This required students to parse the contents of the CSV and to specify the appropriate SQL to insert the data into the database.

**Task 2. Read from the Database** The partially implemented code-base had the necessary information to display the details of the inventory, through a web-based user interface. Task 2 required students to develop the SQL necessary to extract and collate the details of inventory items, for display in the web application, and to integrate with the existing code-base.

**Task 3. Insert into the Database** Students were required to obtain input from users, through a web-based form, and insert it as a new item in the inventory collection. Task 3 also required students to perform the necessary validation and checking of the user-provided input.

### Applying TAFFIES to Developing an AFS

DBI CW2 solutions were submitted through Moodle, the learning management system used at UNNC. Solutions were then downloaded and prepared for marking. Following this approach, students would not need to be cognizant of the AFS, which may impact how they approach solving the assignment. This approach had the benefit of preserving the ecology of the task, with students using a typical development environment rather than one constrained or required by the AFS. The feedback itself was delivered in a feedback file on Moodle.

Security was a practical concern for the first phase of marking. All student solutions were run in a sandboxed Docker container, a lightweight virtualization technology that allows code to run in a consistent, isolated operating environment. In Docker, changes to a container’s files or settings by an application are discarded once the container is powered down. This ensured that the environment in which student solutions were run was consistent for all students, whilst following good security practices when running untrusted code. The output of each student solution was captured, and stored to disk, ready for the next stage, where the actual assessment was performed.

Having completed the initial stages, the coursework marker could now begin developing the AFS using a TAFFIES-based approach. At this stage, the output of the student’s solution was analyzed according to the marking criteria. Following the TAFFIES development process, it was that the marking criteria were implemented using code. This approach differs from the conventional, with the majority of existing AFSs being used as the framework around which assessment tasks are built. Typically, the marking logic used in an AFS will be developed simultaneously with the specification of the particular coursework. In this case, however, the AFS was developed after students had submitted their solutions.
solutions, thereby providing the corpus of solutions from which feedback was generated. In addition to providing specific, tailored feedback, this approach enables a high level of adaptability, something that is a limiting factor in many existing AFSs [13].

When marking, FSs for identifying the correctness of a solution were first specified. Correct solutions are straightforward to mark, as they meet predictable, pre-specified criteria: The marker can check whether the criteria were correctly realized in the student solution, and allocate a mark, accordingly. However, as is common in programming, multiple correct solutions may exist, some of which may possess more desirable properties—faster execution, lower memory utilization, or more explicit better coding practice, for example. Therefore, in addition to identifying the correctness of a given solution, where appropriate, the CW2 feedback was tailored according to the implementation approach taken by students. Once the necessary FSs were completed, they were then linked to the ACS, and the relevant mark assigned.

Having identified, marked, and developed feedback for correct solutions, FSs were then developed to address solutions which were not correct. The FCC was used to identify a solution that had not yet received feedback. This solution was then reviewed and an FS (or multiple FSs) was developed according to that solution’s particular weakness/shortcoming. Specific feedback was also generated to highlight the error and provide advice on how to attempt to rectify it. When a submission exhibited properties expected in a correct solution, partial marks were awarded. The FCC was then run again, with this process being repeated until the FCC reported that all solutions had received at least one item of feedback. Often, a single FS would “match” several submissions, providing an efficient way of generating feedback for all solutions.

**Reflection**

Good software engineering practice includes performing regular retrospectives, reflecting and learning from experiences [39]. The introduction of the AFS in the academic year 2020–2021 alleviated some of the challenges associated with the hybrid delivery of DBI. This enabled provision of uniform, detailed coursework feedback, to a large class, on time, with strong student endorsement. Developing the AFS was a time-consuming process, taking approximately the same amount of time required to manually mark 160 student submissions. The net benefit of consistent and correct marking, when combined with the incremental and iterative development methodology detailed here, a number of the shortcomings typically associated with AFSs were successfully avoided.

**Tailored Feedback** Feedback provided to students is specific, relevant, and detailed; and not simply determined on a pass or fail basis.

The DBI CW2 marking was successfully completed, with administration and feedback delivered on time. Students received a detailed report containing their individual feedback and score. In contrast to most introductory-level coursework feedback and marking, which typically generate significant student reaction and change requests, our AFS deployment resulted in no grade challenges. There were also no identified marking errors, or suggested inconsistencies or unfairness. Student feedback on the AFS was universally positive, with comments indicating an AFS-related increase in student motivation.

We suggest that our AFS addressed a category of errors that are commonly encountered when marking a large number of student assignments requiring detailed technical analysis. Specifically, our AFS promotes:

- **Consistency** The marking is deterministic: The feedback and grading of coursework is the same for all submissions that follow a particular implementation strategy. Consistency here refers to the provision of both marks and feedback.

- **Correctness** Simply automating the process of marking does not ensure the correctness of the output. However, when combined with the incremental and iterative development methodology detailed here, a number of the shortcomings typically associated with AFSs were successfully avoided.

**Conclusion**

The provision of feedback is an important, but challenging, aspect of developing programming skills in novice programmers. The work presented in this paper outlines a novel approach in the specification and development of an AFS, whose efficacy has been explored through a case study.

TAFFIES also demonstrates the application of two software engineering methodologies, Agile and TDD, in a T&L context.

We are planning to further explore the application of TAFFIES in other types of programming assignments. Future work will also seek to establish how well TAFFIES generalizes beyond programming-based assignments. In particular, we will explore the application of TAFFIES in Mathematics and CS fundamentals classes. In Mathematics, FSs could be generated to identify missteps in mathematical proofs. For fundamentals, FSs could also be used for data representation and conversion errors, e.g., converting between Decimal and Binary, a fundamental skill for computer scientists. This will serve as important validation of the proposed framework, which we expect will work across a broad range of task types and programming languages.
Future work will also be performed to evaluate the quality of TAFFIES-based feedback using expert human judges. There is also an opportunity to explore the application of TAFFIES outside code-based assignments. Given the flexible structures specified by TAFFIES, many types of assignments could be supported. Future work will seek to work with educators in other disciplines to explore the viability of supporting different types of assignments using TAFFIES.

Finally, we are planning to re-purpose the AFS as an OER, and make it available to the community, enabling novice programmers to develop their programming skills independent of classes or institution.

Funding This study was partially funded by the National Natural Science Foundation of China (Grant number 61872167). The authors also gratefully acknowledge the financial support provided by the Faculty of Science and Engineering, University of Nottingham Ningbo China.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Gill A, Irwin D, Towey D, Walker J, Zhang Y. Reacting to the coronavirus: a case study of science and engineering education switching to online learning in a Sino-foreign higher education institution. In: 2020 International conference on open and innovative education (ICOIE 2020), Hong Kong, China; 2020. p. 385–404.

2. Gill A, Irwin D, Towey D, Walker J, Zhang Y. Reflections post-lockdown: science and engineering education switching to online learning. Int J Mob Learn Organ (N.D.) (to appear)

3. Wang T, Towey D, Ng RY-K, Gill AS. Towards post-pandemic transformative teaching and learning: case studies of microlearning implementations in two post-secondary educational institutions. SN Comput Sci. 2021;2(4):271.

4. Jenkins T. On the difficulty of learning to program. In: Proceedings of the 3rd annual conference of the LTSN Centre for information and computer sciences, vol. 4. Loughborough, UK: LTSN-ICS; 2002. p. 53–8.

5. Tan P-H, Ting C-Y, Ling S-W. Learning difficulties in programming courses: undergraduates’ perspective and perception. In: 2009 International conference on computer technology and development. Kota Kinabalu, Malaysia: IEEE; 2009. p. 42–46.

6. Robins A, Rountree J, Rountree N. Learning and teaching programming: a review and discussion. Comput Sci Educ. 2003;13(2):137–72.

7. Shute VJ. Focus on formative feedback. Rev Educ Res. 2008;78(1):153–89.

8. Nicol D, Thomson A, Breslin C. Rethinking feedback practices in higher education: a peer review perspective. Assess Eval High Educ. 2014;39(1):102–22.

9. Towey D, Zhao K. Developing an automated coding tutorial OER. In: 2017 IEEE 6th international conference on teaching, assessment, and learning for engineering (TALE). Hong Kong, China: IEEE; 2017. p. 233–238.

10. Singh R, Gulwani S, Solar-Lezama A. Automated feedback generation for introductory programming assignments. In: Proceedings of the 34th ACM SIGPLAN conference on programming language design and implementation. New York, NY, USA: Association for Computing Machinery; 2013. p. 15–26.

11. Odékirk-Hash E, Zachary JL. Automated feedback on programs means students need less help from teachers. In: Proceedings of the thirty-second SIGCSE technical symposium on computer science education. New York, NY, USA: Association for Computing Machinery; 2001. p. 55–9.

12. Ala-Mutka KM. A survey of automated assessment approaches for programming assignments. Comput Sci Educ. 2005;15(2):83–102.

13. Keuning H, Jeuring J, Heeren B. Towards a systematic review of automated feedback generation for programming exercises. In: Proceedings of the 2016 ACM conference on innovation and technology in computer science education. New York, NY, USA: Association for Computing Machinery; 2016. p. 41–46. https://dl.acm.org/doi/10.1145/2899415.2899422

14. Gusukuma L, Barton AC, Kafura D. Pedal: an infrastructure for automated feedback systems. In: Proceedings of the 51st ACM technical symposium on computer science education. New York, NY, USA: Association for Computing Machinery; 2020. p. 1061–1067. https://dl.acm.org/doi/10.1145/3328778.3366913

15. Blau H, Moss JEB. Frenchpress gives students automated feedback on Java program flaws. In: Proceedings of the 2015 ACM conference on innovation and technology in computer science education. New York, NY, USA: Association for Computing Machinery; 2015. p. 15–20. https://dl.acm.org/doi/10.1145/2729094.2742629

16. Head A, Glassman E, Soares G, Suzuki R, Figueredo L, D’Antoni L, Hartmann B. Writing reusable code feedback at scale with mixed-initiative program synthesis. In: Proceedings of the fourth ACM conference on learning at scale. New York, NY, USA: Association for Computing Machinery; 2017. p. 89–98. https://dl.acm.org/doi/10.1145/3051457.3051467

17. Blumenstein M, Green S, Nguyen A, Muthukumarasamy V. An experimental analysis of game: a generic automated marking environment. ACM SIGCSE Bull. 2004;36(3):67–71.

18. Adams MD. Aristotle: a flexible open-source software toolkit for semi-automated marking of programming assignments. In: 2017 IEEE Pacific rim conference on communications, computers and signal processing (PACRIM), Victoria, BC, Canada: IEEE; 2017. p. 1–6. https://ieeexplore.ieee.org/document/8121886

19. Larmar C. Agile and iterative development: a manager’s guide. Boston: Addison-Wesley Professional; 2004.

20. Towey D, Reisman S, Chan H, Demartini C, Tovar E, Margaria T. OER: six perspectives on global misconceptions and challenges. In: 2019 IEEE international conference on engineering, technology and education (TALE). Yogyakarta, Indonesia: IEEE; 2019. p. 889–895.

21. Thomas S, Williams L, Xie T. On automated prepared statement generation to remove SQL injection vulnerabilities. Inf Softw Technol. 2009;51(3):589–98.

22. Janzen D, Saiedian H. Test-driven development concepts, taxonomy, and future direction. Computer. 2005;38(9):43–50.

23. Beck K. Test-driven development: by example. Boston, USA: Addison-Wesley Professional; 2003.

24. Zhu H, Hall PA, May JH. Software unit test coverage and adequacy. ACM Comput Surv (CSUR). 1997;29(4):366–427.

25. Boehm B, Turner R. Management challenges to implementing agile processes in traditional development organizations. IEEE Softw. 2005;22(5):30–9.

26. Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R et al. Manifesto for agile software development; 2001.
27. Feng Y. University of Nottingham Ningbo China and Xi’an Jiaotong-Liverpool University: globalization of higher education in China. High Educ. 2013;65(4):471–85.

28. Mok KH, Jiang J. Massification of higher education: challenges for admissions and graduate employment in China. In: Managing international connectivity. Diversity of Learning and Changing Labour Markets. Singapore: Springer; 2017. p. 219–43.

29. Towey D. Open and flexible learning as an alternative in mainland Chinese higher education. In: Emerging modes and approaches in open and flexible education, pp. 12–26. The Open University of Hong Kong Press, Hong Kong, China; 2014.

30. Eisenberg A, Melton J. SQL: 1999, formerly known as SQL3. ACM Sigmod Rec. 1999;28(1):131–8.

31. Hickson I, Hyatt D. HTML5. W3C (2008). http://www.w3.org/html/wg/html5/. Accessed Dec 2021.

32. W3C: CSS 2.1 Specification. http://www.w3.org/TR/CSS2/. Accessed Dec 2021.

33. Pike M, Lee B-G, Towey D. The robots will rule: improving coursework marking and feedback through an automated system. In: Proceedings of the 2021 international conference on open and innovative education (ICOIE 2021), Hong Kong, China; 2021. p. 268–77.

34. Lakhani KR, Von Hippel E. How open source software works: “free” user-to-user assistance. In: Produktentwicklung Mit Virtuellen Communities. Jersey City, USA: Springer; 2004. p. 303–39.

35. Grinberg M. Flask web development: developing web applications with Python. Newton: O’Reilly Media, Inc.; 2018.

36. Aslam FA, Mohammed HN, Mohd JM, Gulamgaus MA, Lok P. Efficient way of web development using python and flask. Int J Adv Res Comput Sci. 2015;6(2):54–7.

37. Shafranovich Y. Common format and MIME type for comma-separated values (CSV) files. RFC 4180, October (2005). https://tools.ietf.org/html/rfc4180. Accessed Dec 2021.

38. Špaček F, Sohlich R, Dulík T. Docker as platform for assignments evaluation. Procedia Eng. 2015;100:1665–71.

39. Dybá T, Dingșîyr T, Moe NB. Agile project management. In: Software project management in a changing world, Chap. 11. Jersey City, USA: Springer; 2014. p. 277–300.

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.