IJPTL

International Journal of Practices in Teaching and Learning (IJPTL)

Published: August 2022

STeM: A Web-Based System for Managing Test Artifacts in Software Testing Course

Nor Shahida Mohamad Yusop
Universiti Teknologi MARA Cawangan Selangor
40450 Shah Alam, Selangor, Malaysia
nor_shahida@uitm.edu.my

Adib Suhaimi Mohd Fikri
Universiti Teknologi MARA Cawangan Selangor
40450 Shah Alam, Selangor, Malaysia

Nursyuhaila Yahaya
Universiti Teknologi MARA Cawangan Selangor
40450 Shah Alam, Selangor, Malaysia
syuhailayahaya@gmail.com

***Abstract*—** With the increasing demand for sophisticated and cost-effective software testing, many software industries are looking for highly skilled software testers. To meet that demand, many universities have offered software testing courses in their software engineering programmes. At the Faculty of Computer and Mathematical Sciences Universiti Teknologi MARA, a software testing course was offered to Bachelor of Information System Engineering students in semester 5. As part of the course assessment, students are required to conduct system tests in which they need to design, execute, and document test cases. Currently, the use of Microsoft Word and Excel Spreadsheet to document test cases and test results are not effective for test case management. The most common problems experienced by the students when using Microsoft Word to manage test cases are: 1) difficulty to ensure document consistency when various team members update the test artifacts, 2) unable to track and monitor the test status when the test results are kept individually, and 3) improper test-requirements traceability, since separate traceability documents need to be maintained and updated throughout the development life cycle. This paper reports on the development of STeM web-based system to assist the test case management in a software testing course. The STeM serves as a collaborative platform for the students to build test cases, create requirements traceability, record test cases results, monitor test case status, and produce test reports. In addition, the lecturer can monitor the test progress through this system too.

Keywords—software engineering programme, software testing course, test case management

I. INTRODUCTION

Software testing is imperative in ensuring the quality of software products. According to report from Cambridge University, the global cost of finding and fixing software systems has risen to $312 billion annually, and this cost constitutes about half of the development costs of a typical software project. With the increasing complexity of software products, the software industry needs highly-skilled software testers. To meet this industrial demand, many universities have offered software testing courses in their software engineering or computer science programmes.

At the Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, the software testing course is offered to students of Bachelor in Information System Engineering. The students enroll into this course in semester 5 after passing System Requirement and Analysis, and System Design and Implementation courses in semester 3 and 4, respectively. There are five types of test activities covered in the software testing course syllabus: 1) identify test conditions/scenarios, 2) design test cases, 3) build test cases, 4) execute test cases, and 5) compare the test outcomes. These five activities are assessed in group assignments, where the students are required to conduct system tests on software projects. Through this testing assignment, students get hands-on experience in preparing system tests, particularly in designing test cases, executing test cases, and documenting test artifacts. Currently, all the test artifacts such as test cases, test results, and defect reports are written in Microsoft Word and Excel spreadsheet. Since these artifacts are stored separately by each student, matters related to document consistency, transparency, tracking of test progress, and traceability between test cases and requirements become a challenge. In software project development, it is common for people involved in the project to want to know about the testing progress and status – when will the testing be completed? How many test cases have been executed? How many test cases have failed? What is the testing progress? However, in the absence of a centralized repository to store all these test data or metrics, it is difficult to answer such questions.

To enhance the university teaching and learning in software testing, we have developed a web-based test management system as an alternative platform for students to create,

manage, track, and monitor test activities. According to Parveen et al. (2007), test management is critical to enabling test assets and artifacts such as test requirements, test cases, and test results are accessible and reused. In addition to providing access to test documentation, Burnstein (2006) suggested to include testing status features in the test management system to monitor work efforts, test schedule, and monitor test progress. According to IEEE Standard 829-2008, good test case management also should consider various aspects of the testing process that include preparation of test execution, monitoring the execution, reporting test progress, assessing test results, and analysis of anomalies. Although many open source test management tools such as JIRA, GitHub, and TestRail are available for the students to manage the test activities, most of these tools are limited and not suitable to be used in educational project settings. In our software testing course, for example, the test management elements only focus on the management of requirements, test cases, and test results that were produced during the system test. However, our review on existing open source test management tools found that most of the features offered are too advanced to be used by novice users and not suitable for learning purposes. Therefore, the need to develop a customized test management tool for learning purposes is very much needed to improve the test process.

The rest of this paper is organized as follows. Section 2 introduces the learning activities in the software testing course offered at UiTM. Section 3 gives an overview of related work. Section 4 presents the development process of STeM. Section 5 reports the user feedback of STeM. Finally, Section 6 concludes the paper and suggests future work.

II. LEARNING ACTIVITIES IN SOFTWARE TESTING COURSE

This section introduces learning activities conducted during a software testing course at the Faculty of Computer and Mathematical Sciences, UiTM. Students register for the software testing course in the fifth semester after passing System Requirement and Analysis and System Design and Implementation courses in the third and fourth semester, respectively. It is a 3-credit hour course with 28-hours of class lectures and 28-hours of lab sessions conducted in a 14-week semester. In addition to learning the theoretical software testing foundations, students are also exposed to practical testing activities by conducting system test on a project developed in the previous semester.

A. Learning Activities

To expose students with real-world testing experience, students are required to conduct the system test on a software that was developed by them during the Design and Implementation Course. Students worked in groups of six to eight based on their personal choices. They are required to carry out the five test activities below:

1. Identify test conditions – Test condition is an item or event that could be verified by a test. Before designing a test case, students are required to determine what can be tested and prioritize these test conditions. For this activity, students need to 1) identify a list of positive and negative test conditions, and 2) compile these test conditions in a Google Sheet so that everyone can update in one document.
2. Design test cases – Test case design determines how the test conditions will be tested. A test case is a set of tests performed in a sequence and related to a test objective. During the test case design, students will have to identify 1) specific input values, 2) expected outcomes, and 3) environment prerequisites. This information is recorded in the same Google Sheet where the test conditions are stored.
3. Build test cases – During this activity, the Software Test Description is prepared. Students are required to document 1) test case ID, 2) test case name, 3) description of the test cases, 4) test procedure, 5) pre-and post-conditions of the test, 6) test input, and 7) expected outcome. This information is written in Microsoft Word document.
4. Execute test cases – The system test is focused on manual checking of the software under test by following the test procedures in the test case. The student-tester needs to 1) enter inputs, 2) observe outcomes, and 3) make notes about any problems as they proceed.
5. Compare test case – The actual outcomes of each test case must be compared and investigated if the software being tested worked as expected. Each student-tester will record the following information in Excel Spreadsheet: 1) test results either pass or fail, 2) remarks on why certain test cases failed, and 3) submit defect reports or some input on further enhancement if the test case failed. At the end of the testing phase, all the test results are compiled and reported in the Software Test Report.

B. Test Artifacts

Test artifacts are test deliverables or documents that are created when the testing is being carried out. The test artifacts are required by the team members and the stakeholders to know about the progress in the project. In the real software testing environment, test artifacts consist of test strategy, test plan, test case, test data, requirement traceability matrix, test coverage, defect report, and test summary report. In our software testing course, we only focused on four artifacts, which are test case, requirement traceability matrix, defect report, and test summary report. These artifacts are documented in the following documents:

1. Software Test Description (STD): The STD document describes the test preparation, test cases,

and test procedures to be used during the system testing. Besides that, a requirement traceability matrix is also presented in the STD to illustrate the relationship between requirements, design, and test case. However, manual traceability matrices using Microsoft Word are vulnerable because when any element of traceability data is modified, the affected relationships must be updated manually.

2. Software Test Report: The STR document summarizes the testing activities and tests results after testing is completed. Currently, an individual student-tester will record the test execution results in their local Excel Spreadsheet. This process, however, is impractical where team members are unable to track the testing status, and lecturers could not monitor the testing progress.
3. Defect Report: Defect report is a document that describes a defect found by a tester when conducting a system test. The purpose of a defect report is to state the problem as clearly as possible so that developers can reproduce the problem, debug, and fix it. Currently, any problems found during system testing are recorded in Microsoft Word or Excel Spreadsheet.

III. Related Work

A. *Software Testing in University Education*

Many software industries have reported that software systems are becoming increasingly complex, however, there is a shortage in recruiting "quality" software testers. To address this workforce shortage, many universities have offered software testing courses in their software engineering programmes. However, some research has reported that undergraduate computing courses do not provide students with an integrated view of software test contents compared to other undergraduate courses in general. Instead, students view software testing as a fundamental course for learning how to use testing tools, rather than acquiring test design skills (Garousi, 2011). In fact, Scatalon et al. (2019) have reported that there are still complaints about the teaching of software testing in education because the focus is on theory and there is a lack of practice to show students how to apply the concepts.

Software testing education is not a "one-size-fits-all" approach (Barr et al., 2020; Elgrably & Ronaldo Bezerra Oliveira, 2020). In different universities, although different pedagogical approaches were introduced, the software testing curricula generally consist of theoretical and practical elements. For example, many universities have started using a real-world project in their software testing curricula. Through project-based learning, students are exposed to a wide variety of test activities and techniques and are able to experience real-world testing practices. At Anhui SanLian University, China, android-based testing was introduced for students to test a lightweight autonomous learning system (Yu, 2019). Meanwhile, at Guangzhou University, to improve software testing capabilities, software testing educators use public and private cloud platforms to run software test experiments (Wen et al., 2019). From the review of software testing education (Garousi et al., 2020), the most frequent included test activities in software testing courses were test case design and test execution. Limited exposure is given on test process, planning and management. Other than classroom teaching, and project-based learning, at certain universities, software testing training is given to students as part of the education curricula. However, not many universities provide this opportunity. In Hong Kong, for example, only a small percentage of students had completed formal software testing training as part of their university education (Chan et al., 2005)

Other than having a dedicated course on software testing, testing was also taught as part of a regular programming course. In Computer Science courses, for instance, Junit testing frameworks were used to write and execute unit tests in Java programming language (Bai & Stolee, 2020). This can help students to write better code and detect errors faster.

B. *Test Case Management Tool*

Test management process consists of activities to organize, control, track, and monitor test artifacts throughout the whole software testing cycle. While there are many open source test case management tools available to support these activities, most of the tools are limited to be used in educational environments. Due to time constraints, test management activities in the software testing course typically focused on organizing test design, tracking test progress, and creating traceability between requirements and test cases.

In previous research, not many studies have explored the practical use of test case management tools in software testing courses. The closest to our study can be found in a study carried out by Safana and Ibrahim (2010). In this study, the authors compared the strengths and weaknesses of several test management tools to select the most suitable tool to manage the tests of an educational-project called OBA. They have chosen the SpiraTeam tool over Quality Center, Rational Test Manager, SilkCentral Test Manager, and Wip-CAFÉ TMS tool. The reasons for selecting the SpiraTeam over others include its ability to support requirement management, test case management, release planning, iteration planning, incidence tracking, and artifact relationships.

In industrial research, many studies have focused on the development of effective test management tools for certain types of projects. For example, Eldh et al. (2010) developed an automated test management tool to support large and complex intensive systems with continuous builds. Compared to standard test management tools, this newly developed tool is capable to create test cases based on test specifications and system components, automate test scheduling, automate test execution process, and auto

generate test suite. Herramhoff et al. (2006) developed two tools to support accessibility testing in creating test suites and store test execution. Parsifal is a test case editor developed as a desktop application to allow editing of test description files, while Amfortas is a web application used to store evaluation results. Jinil et al. (2019), have customized the Cradle® tool to support systematic management and association of information from system requirements, test cases, test events, and test results. In this research, the author identified items to be managed, the attributes of each item, and the relationship between them.

IV. Development of STeM

Software testing is important in verifying the system requirements. According to (Kim et al., 2019), the information of test management activities should be systematically managed, especially the interrelation between system requirement, test cases, and test results. A good system requirement should be tested by at least one test case, and the test status of each test case should be recorded. In addition, during the test execution defects may be found, and these defects must be linked to the related test case. Currently, the use of word processors and spreadsheets to record and manage the requirements, test cases, test results, and defect reports are not effective to query certain test metrics and monitor test progress. In fact, the traceability between requirements and test cases are difficult to track.

STeM is a web-based test management tool to record requirements, test cases, and test results. In addition, the users can monitor the progress of test execution and the success and failure rate of the test. In a software testing learning environment, the STeM tool enables lecturers to review test cases development, monitor the testing progress, and verify defects validity more effectively. The following subsections describe main aspects of the development of the STeM tool in more detail.

A. Key Features of STeM

Generally, the STeM tool allows students to record requirements, test cases, test results and monitor test progress. In addition, the STeM tool enables the project manager (referring to lecturer) to review test cases development, monitor the testing progress, and verify defects validity more effectively. The following are the key features of STeM tool:

- *Easy-to-use design* – The system is developed in a simple design that accommodates the testing process carried out during the software testing course.
- *Support multiple projects*– The system supports multiple projects. Users can create new projects, update and view project information.
- *Personalized user permissions* – In the software development team, there are a range of roles involved, including software developers, testers, project managers and all of whom need access to different information depending on their roles. A project manager, for example, will only require read-only access for monitoring and reporting, while software developers and testers have full access to the system.
- *Document repository* – The system allows users to store test cases, requirements, and test results. All this information can be updated and deleted where necessary.
- *Assigning and tracking* – The system has the ability of task assignment. Project manager can assign specific test cases to certain testers, and he or she can monitor the progress.
- *Traceability* – The information structure in this system is designed based on a navigation tree for a better traceability representation from the requirements, to test case, to defect, with a record of the tests executed and the person responsible.
- *Reporting* - The tool can generate reports and allow users to download the reports.

B. Design and Development

The STeM is a web-based application and was designed based on the 'Model – View – Controller' design pattern, supported by Laravel web application framework. The Model is responsible for managing data received from the Controller, the View produces a presentation of the Model (data) in a particular format, and the Controller responds to the user input and converts it to commands for the Model or View. Using Laragon tool, the application frontend and database management system was developed using PHP and MySQL, respectively. The STeM tool is available at https://testcasemanagementtool.herokuapp.com

C. Graphical User Interface Design

Below are some screenshots of the developed STeM tool. The main interface of the system is shown in Fig. 1. This interface is the main dashboard that lists the projects that have been registered in the database. There are four menu options:

1) Add new project: register a new project and continue working with it when required by the user.
2) Delete: delete the project if the project is no longer needed.
3) Update: update the project name when the project name changes.
4) View: add or view requirements for the selected project. Choosing this option will bring the project interface as shown in Fig. 2. In this interface, users can create new requirements, update and delete requirements.

To support traceability between requirements and test cases, users can add relevant test cases for each requirement (See Add Test Case option in Fig. 2). In the test case form

as shown in Fig. 3, user is required to fill in the following test case information:

1. Test case identifier – a unique identifier for the test case which is manually assigned
2. Expected execution time – expected time to complete the test case
3. Test case title – the essence of the test
4. Pre-condition – preparative steps for setting the system and testing environment
5. Steps to execute test case – the required steps to execute the test case
6. Expected result – the expected result after executing the test case
7. Priority – the priority to execute the test case (low, medium, high)
8. Attachment file – attached file, screenshot

Once the test case has been executed, the user can update the test results in Test Result Details form as shown in Fig. 4). The test result attributes are:

1. Identifier – a unique identifier for the test result which is manually assigned
2. Result – the status of the test cases either pass or fail
3. Comment – make notes to the test execution related to the observation of the test execution or suggestions
4. Attachment file – attached file, screenshot, event log, test log

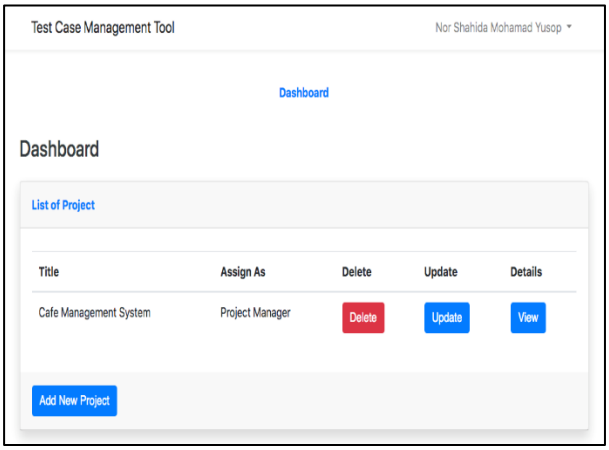

Figure 1: Main application interface (Manage Project Use Case)

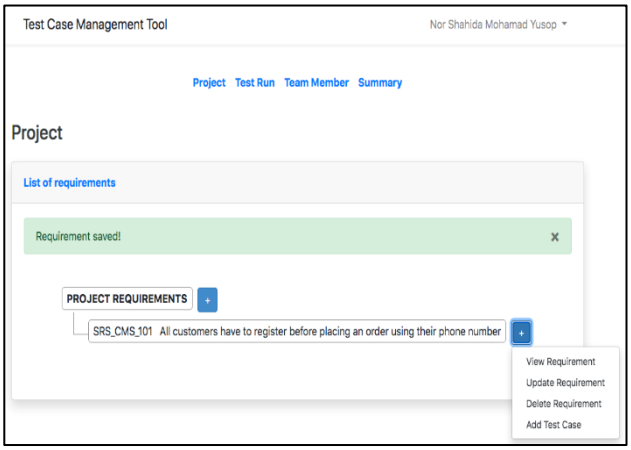

Figure 2: Project interface to manage requirements and test cases

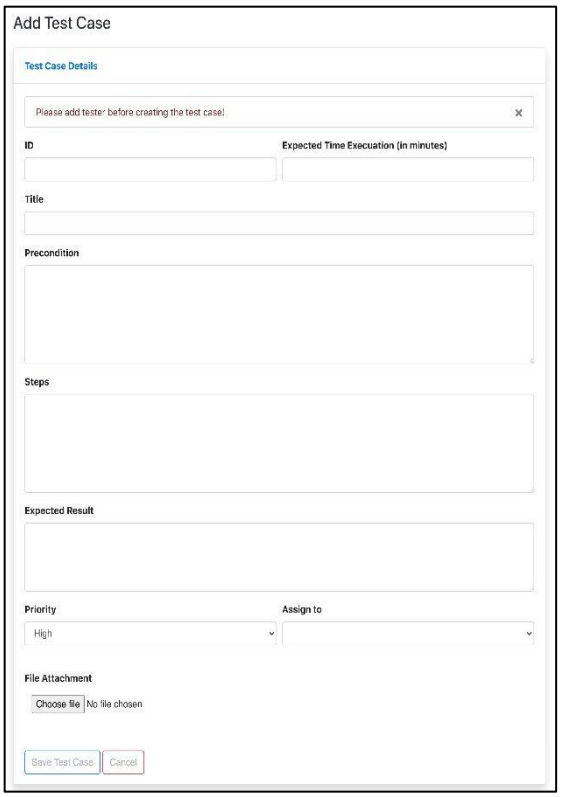

Figure 3: Test case form to add new test cases

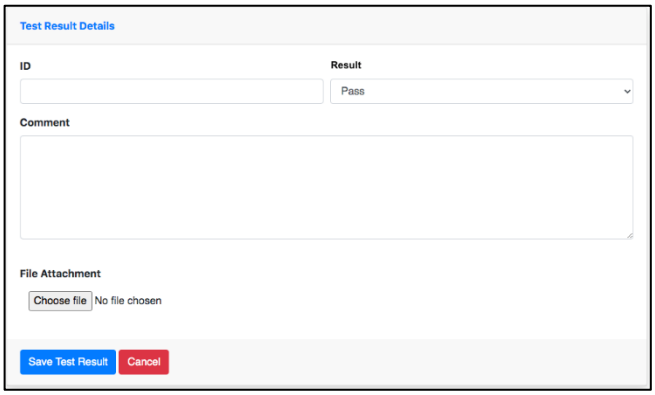

Figure 4: Test result details form

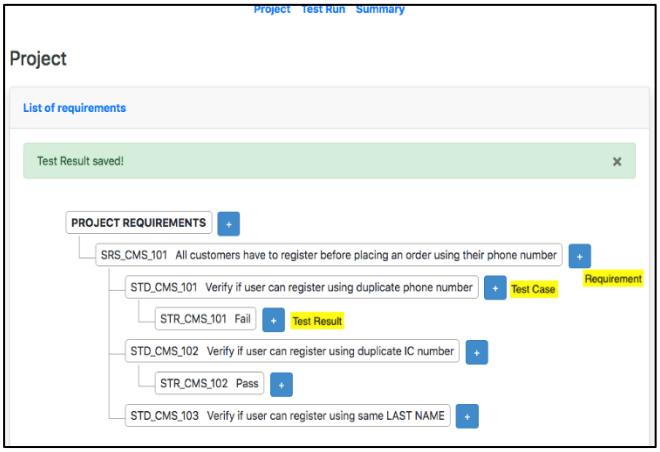

Figure 5: The hierarchical navigation structure to view traceability between requirements, test cases, and test results

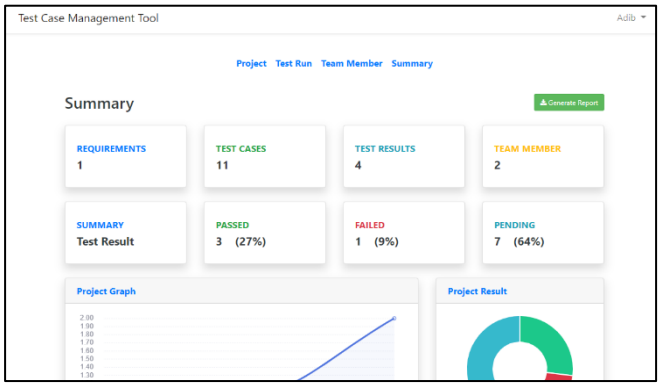

Figure 6: Reporting Interface

To improve the readability and traceability between the requirements, test cases, and test results, hierarchy navigation structure design was used, as shown in Fig. 5. The top node is the project name, followed by the first, second and third level child nodes which are the requirements, the test cases, and the test results respectively.

Figure 6 shows the summary of the testing information. In this reporting interface, users can get information about: 1) number of requirements for a specific project, 2) total number of test cases created, 3) number of test cases that have been executed, 4) number of passed or failed test cases, and 5) number of pending test cases. This information can also be visualized in graphs and pie charts for analysis purposes. All users can access this reporting interface but only the project manager is allowed to export the information.

V. User Feedback on STeM

The STeM tool was developed using prototyping methodology in which a prototype was built, tested, and reworked based on the feedback from users. Two prototypes have been developed and demonstrated to the UiTM students and lecturers of the Information System Engineering software testing course. After the demonstration session, students and lecturers filled the feedback form. Table 1 summarizes the feedback gathered. Based on the feedback from the first prototype, two functionalities were refined: 1) the ability to update user profile, and 2) additional option to export test information to .pdf file. In addition, the content organization of traceability has been changed from a listing style to a tree navigation structure to improve readability.

In the second prototype, no major changes were requested. Only the Overview Main page was removed from the system. The main features and designs of STeM are as reported in Section 4.

Table 1: Summary of user feedback during first and second prototype evaluation

Prototype	Feedback	Refinement
1	The user profile should provide an updated operation.	Add new functionality to manage the user profile. The user will be able to update their profiles. The information that can be updated: names, email address, and password.
1	The system should trace the requirement with the test case and its result. The current system needs the user to open so many pages, so it is hard to read and trace the requirements with a test case. The system should make it a tree structure for easier readability and traceability.	The user interface of requirement, test case, and the test result will be changed due to difficult readability. All the requirements, test cases, and results will be in a tree structure on one page that will have all the functionalities to manage information on the requirement, test case, and test results.
1	The Project Manager should be able to print out or export all information of the project testing into pdf.	New functionality on reporting is added. The report consists of all information on requirements, test case, and test result.
2	Function and feature work fine, but why do project managers need an overview tab? Because it just lists all the requirements and test cases the same as the project page. The overview page should be excluded from the system	Exclude overview page from the system

VI. Conclusion and Future Work

This paper presents some educational efforts in bridging industrial practices into the learning of software testing courses at higher educational institutions. Although there are many kinds of test management tools available in the

market, the features presented in STeM may be similar but have been adapted to learning environments that only focus on organizing the test artifacts. Currently, the STeM can only support test case creation, traceability between requirements, test cases, test progress monitoring, and reporting. Using STeM, students and lecturers has the following benefits:

- Centralized repository for test artifacts: Students can create requirements, test cases and upload test results in a single repository. This avoids the creation of multiple documents. At the same time, it is easy for the lecturers to review the accuracy and validity of the requirements and test cases.
- Manage traceability easily: The hierarchical navigation structure enables the requirements, test cases, and test results to be seen in one screen. In this way, students can ensure that all functional requirements are accounted for in test cases.
- Provide test metrics: As part of the course assessment, lecturers are able to monitor students' performance and their test progress from the recorded test metrics, such as total test pass/fail, total test runs, total test cases executed per day or per person.

In the future, the STeM system is planned to support other test management process that can provide the following functions:

- Test monitoring and control – provide a mechanism to monitor the status of projects either on schedule or otherwise.
- Defect report management – provide the ability to record defect reports for specific test cases. In addition, traceability between requirements, test cases, test results, and defect reports can be improved.
- File import/ export – provide the ability to import test cases from excel or other types of file, and automatically create the navigation tree structure. In addition, the data from the system can be exported to certain types of file like, .cvs or .doc.

ACKNOWLEDGMENTS

Support from the Fundamentals Research Grant Scheme (FRGS) under contracts FRGS/1/2018/ICT 01/UITM/02/1, Universiti Teknologi MARA (UiTM) is highly acknowledged.

REFERENCES

Bai, G. R., & Stolee, K. T. (2020). Improving Students' Testing Practices. In *Proceedings - 2020 ACM/IEEE 42nd International Conference on Software Engineering: Companion, ICSE-Companion 2020* (pp. 218–221). https://doi.org/10.1145/3377812.3381401

Barr, M., Nabir, S. W., & Somerville, D. (2020). Online Delivery of Intensive Software Engineering Education during the COVID-19 Pandemic. In *2020 IEEE 32nd Conference on Software Engineering Education and Training, CSEE and T 2020* (pp. 244–249). https://doi.org/10.1109/CSEET49119.2020.9206196

Burnstein, I. (2006). *Practical Software Testing: A Process-Oriented Approach*. New York, New York, USA: SpringerProfessional Computing.

Chan, F. T., Tse, T. H., Tang, W. H., & Chen, T. Y. (2005). Software testing education and training in Hong Kong. In *Proceedings - International Conference on Quality Software* (Vol. 2005, pp. 313–316). https://doi.org/10.1109/QSIC.2005.57

Eldh, S., Brandt, J., Street, M., Hansson, H., & Punnekkat, S. (2010). Towards fully automated test management for large complex systems. In *ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation* (pp. 412–420). IEEE. https://doi.org/10.1109/ICST.2010.58

Elgrably, I. S., & Ronaldo Bezerra Oliveira, S. (2020). Model for teaching and training software testing in an agile context. In *Proceedings - Frontiers in Education Conference, FIE* (Vol. 2020-Octob). https://doi.org/10.1109/FIE44824.2020.9274117

Garousi, V. (2011). Incorporating real-world industrial testing projects in software testing courses: Opportunities, challenges, and lessons learned. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training, CSEE and T 2011 - Proceedings* (pp. 396–400). https://doi.org/10.1109/CSEET.2011.5876112

Garousi, V., Rainer, A., Lauvås, P., & Arcuri, A. (2020). Software-testing education: A systematic literature mapping. *Journal of Systems and Software*, *165*. https://doi.org/10.1016/j.jss.2020.110570

Herramhof, S., Petrie, H., Strobbe, C., Vlachogiannis, E., Weimann, K., Weber, G., & Velasco, C. A. (2006). Test case management tools for accessibility testing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 4061 LNCS, pp. 215–222). https://doi.org/10.1007/11788713_32

Kim, J., Yeom, C., & Shin, J. (2019). Management of software tests using CASE tools. In *ISSE 2019 - 5th IEEE International Symposium on Systems Engineering, Proceedings* (pp. 2019–2022). https://doi.org/10.1109/ISSE46696.2019.8984584

Parveen, T., Tilley, S., & Gonzalez, G. (2007). A case study in test management. *Proceedings of the Annual Southeast Conference*, *2007*, 82–87. https://doi.org/10.1145/1233341.1233357

Safana, A. I., & Ibrahim, S. (2010). Implementing software test management using SpiraTeam tool. In *Proceedings - 5th International Conference on Software Engineering Advances, ICSEA 2010* (pp. 447–452). IEEE. https://doi.org/10.1109/ICSEA.2010.76

Scatalon, L. P., Garcia, R. E., Carver, J. C., & Barbosa, E. F. (2019). Software testing in introductory programming courses a systematic mapping study. In *SIGCSE 2019 - Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 421–427). https://doi.org/10.1145/3287324.3287384

Wen, W., Sun, J., Li, Y., Gu, P., & Xu, J. (2019). Design and Implementation of Software Test Laboratory Based on Cloud Platform. *Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019*, 138–144. https://doi.org/10.1109/QRS-C.2019.00039

Yu, J. (2019). Design of a Lightweight Autonomous Learning System for the Course of Software Testing Based on Android. *Journal of Physics: Conference Series*, *1288*(1). https://doi.org/10.1088/1742-6596/1288/1/012051