Automated Assessment: Experiences From the Trenches

Andrew Luxton-Reilly
University of Auckland
Auckland, New Zealand
andrew@cs.auckland.ac.nz

Ewan Tempero
University of Auckland
Auckland, New Zealand
e.tempero@auckland.ac.nz

Nalin Arachchilage
University of Auckland
Auckland, New Zealand
nalin.arachchilage@auckland.ac.nz

Angela Chang
University of Auckland
Auckland, New Zealand
angela@cs.auckland.ac.nz

Paul Denny
University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Allan Fowler
University of Auckland
Auckland, New Zealand
allan.fowler@auckland.ac.nz

Nasser Giacaman
University of Auckland
Auckland, New Zealand
n.giacaman@auckland.ac.nz

Igor' Kontorovich
University of Auckland
Auckland, New Zealand
i.kontorovich@auckland.ac.nz

Danielle Lottridge
University of Auckland
Auckland, New Zealand
d.lottridge@auckland.ac.nz

Sathiamoorthy Manoharan
University of Auckland
Auckland, New Zealand
s.manoharan@auckland.ac.nz

Shyamli Sindhwani
University of Auckland
Auckland, New Zealand
shyamli.sindhwani@auckland.ac.nz

Paramvir Singh
University of Auckland
Auckland, New Zealand
p.singh@auckland.ac.nz

Ulrich Speidel
University of Auckland
Auckland, New Zealand
u.speidel@auckland.ac.nz

Sudeep Stephen
University of Auckland
Auckland, New Zealand
sudeep.stephen@auckland.ac.nz

Valerio Terragni
University of Auckland
Auckland, New Zealand
v.terragni@auckland.ac.nz

Jacqueline Whalley
Auckland University of Technology
Auckland, New Zealand
jacqueline.whalley@aut.ac.nz

Burkhard C. Wünsche
University of Auckland
Auckland, New Zealand
burkhard@cs.auckland.ac.nz

Xinfeng Ye
University of Auckland
Auckland, New Zealand
xinfeng@cs.auckland.ac.nz

ABSTRACT

Automated assessment is commonly used across the spectrum of computing courses offered by Tertiary institutions. Such assessment is frequently intended to address the scalability of feedback that is essential for learning, and assessment for accreditation purposes. Although many reviews of automated assessment have been reported, the voices of teachers are not present. In this paper we present a variety of cases that illustrate some of the varied motivations and experiences of teaching using automated assessment.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

automated assessment, teaching, computing education, feedback

ACE '23, January 30-February 3, 2023, Melbourne, VIC, Australia

ACM Reference Format:
Andrew Luxton-Reilly, Ewan Tempero, Nalin Arachchilage, Angela Chang, Paul Denny, Allan Fowler, Nasser Giacaman, Igor' Kontorovich, Danielle Lottridge, Sathiamoorthy Manoharan, Shyamli Sindhwani, Paramvir Singh, Ulrich Speidel, Sudeep Stephen, Valerio Terragni, Jacqueline Whalley, Burkhard C. Wünsche, and Xinfeng Ye. 2023. Automated Assessment: Experiences From the Trenches. In *Australasian Computing Education Conference (ACE '23), January 30-February 3, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3576123.3576124

1 INTRODUCTION

One of the key factors in learning is obtaining feedback on the quality of student work [13]. Looking at the broader educational research on feedback, Wisniewski et al. [29] describe feedback as being about a task, a process, one's self-regulation, or one's self. In this paper, we focus on feedback about a task. To be effective, feedback about a task should provide learners with information that allows them to understand the gap between their current and desired state [23]. Mulliner and Tucker [22] surveyed students and teachers about feedback practices. They found high agreement between teachers and students that feedback should be:

. . . timely, constructive, encouraging, provide detailed direction for

future improvement (feed-forward), be linked to a marking scheme and be specific about the failings of the work.

Assessing students and providing feedback in large computing classes can be challenging [24]. Automated assessment tools (AATs) are deployed in educational contexts to improve processes and outcomes for both teachers and students, particularly in large classes [7, 28]. AATs are reported to benefit teachers by reducing administration involved in collecting and distributing student work for grading, ensure consistency of marking, reduce overall workload associated with grading, and by supporting teachers to identify and reduce academic misconduct in assessed work. Students may benefit from AATs through an increase in the timeliness [3] and quantity of feedback [11], and improvement of self-regulation skills [23].

There are many ways that AATs can be used within courses in the computational sciences to provide feedback, including both summative and formative feedback. In this practice report, we share the collective experiences of teachers using AATs through case studies and reflections across a wide range of contexts in the hope that it helps practitioners considering the use of AATs and the potential impacts on course delivery, student behaviour, and teaching and learning.

2 RELATED WORK

Automated assessment is a widely used and widely studied topic in Computing Education. A review of introductory programming research by Luxton-Reilly et al. [20] in 2018 found that the majority of papers related to feedback focused on tools used to automate that feedback, mainly for the purposes of reducing the cost of marking, and to provide rapid feedback to students. However, the provision of rapid feedback is observed to result in some students focusing on passing test cases through rapid iterative submission cycles rather than systematically improving their software product.

In 2003, an ITiCSE Working Group [4] surveyed tertiary teachers about their perceptions of computer aided assessment. At that time the authors concluded that using AATs is not a panacea for the problems of assessment and that the technology has several limitations. Notably the "black and white" (p116) marking systems typically used by these tools can be problematic where partial marks are desired, and significant time and effort is needed to set up such systems. Since this early survey, technology has advanced and the use of AATs are more prevalent. Due to the quantity of papers about AATs, we focus here on the several reviews reported in the literature.

In early work, Ala-Mutka [1] surveyed the use of AATs for programming assignments. The review is organised around the features supported by tools, and finds that dynamic assessment using a sandbox is essential, and it is common to test functionality using test cases. Some systems evaluated aspects of style (mainly code conventions), or used standard software metrics to evaluate source code. The benefits for teachers include administration support for marking workflow, semi-automatically grading (e.g., automatically grading correctness and manually grading style), and encouraging students to improve their work by allowing repeat submissions. Ala-Mutka [1] advocate that teachers consider carefully how to incorporate AATs into their courses, since there is the potential for negative impact.

In the same year, Douce et al. [8] published a review of automatic test-based assessment of programming. They provide a historical overview, categorising tools into three generations, the first of which demonstrate the viability of automated assessment of programming, but have limited application. The second-generation tools use command line and locally supported GUIs, while the third generation use the web to support submission of code using web browsers. The authors conclude with the observation that AATs can support students by helping them to build confidence, and can support teachers by making the assessment of programs more efficient.

Ihantola et al. [15] extends the previous work by conducting a systematic review of AATs for programming exercises published 2006–2010, focusing on the features supported by the systems. At the time, they found the majority of the systems supported Java, increasingly AATs were integrated into Learning Management Systems, and testing the functionality of code is the most common form of evaluation. They also discuss the value in combining manual marking with AATs, and report only a small number of systems that support higher level computing courses such as web-programming, concurrent programming, and SQL.

Although the features supported by tools that automatically assess programming is the main focus of the Ihantola et al. [15] review, they also reflect on a variety of ways that AATs could be used in the classroom to limit the potentially harmful practice of trial-and-error problem solving. These include: (a) limiting the number of submissions; (b) limiting the amount of feedback; (c) imposing a time penalty; (d) using parameterised questions that change after each submit; and (e) limiting the time available to complete work.

Souza et al. [27] conducted a systematic review of assessment tools for programming assignments, focusing on identifying the tools that have been reported and the main characteristics of those tools. They identified 30 tools reported in publications from 1985–2014. The tools are categorised according to a variety of features including: immediacy of feedback; checks for functional correctness; checks for efficiency; checks for adequate test suites; use of software metrics; level of documentation; compliance with style guidelines; and plagiarism. As with previous reviews, the focus is the technology rather than pedagogy.

Keuning et al. [16] reports the results of a systematic review of automated feedback generated for programming exercises that identified 101 tools described in 146 papers. They investigate the nature of the feedback generated, the techniques used to generate feedback, how the tools are used by teachers, and the evidence for quality and effectiveness of the feedback or tool. They found that many tools provided feedback of more than one type, with most tools providing feedback about mistakes. Almost half of the tools also provided some hints about how students could progress, while fewer tools provided explanations about the task or subject matter required to complete the task. A variety of technical solutions are employed by the tools, the most common of which is automated testing, followed by program transformations and basic static analysis. Keuning et al. [16] conclude by observing that the diversity of automated feedback is increasing, but teachers still do not always find it easy to adapt the tools to their own needs.

Deeva et al. [6] reviews automated feedback systems from 2008–2019. Their *Technologies for Automated Feedback* classification framework comprises four main components: Architecture, Feedback, Educational Context, and Evaluation. As we are primarily interested in the application of AATs for practitioners, we focus on the Feedback category, which comprises Adaptiveness (whether the system provides feedback that adapts to the task, the student, or is non-adaptive), Timing (feedback is immediate, on request, or occurs on task completion), Degree of Learner Control (students have High Control over feedback, Mild Control over feedback, or No Control), and Purpose (Corrective, Suggestive, Informative or Motivational). Despite this framing, the details of classroom use by practitioners is not reported in the review.

Paiva et al. [25] reports a "state-of-the-art" review of research on AATs in computing in which 778 publications between 2010 and 2021 were analysed. They found reports of AATs used in the subject domains of visual programming, programming assignments, system administration, formal languages and automata, software modelling, software testing, databases, web development, parallelism and concurrency, mobile development and computer graphics. Automated testing involves blackbox testing (outcomes), whitebox testing (source code analysis), code quality, software metrics, test development and plagiarism checking. The authors identify 128 AATs, but focus on analysing the features supported by a subset of 30 tools. Although the main focus of the review is the tools and technologies, the authors find evidence that AATs: (a) reduce teacher workload; (b) improve student learning; (c) increase the number of activities solved by students; (d) is generally well-received by students, and (e) engages students working alone.

In summary, a lot of work has explored the use of AATs in computing, including at least 778 publications over the last decade. More than a dozen reviews of AATs have been conducted during the last 20 years, most of which focus on the technology used to perform the assessment, and the features that are supported by the tools. Although some reviews have considered aspects of AATs other than implementation details, such as measures of effectiveness, there is no broad analysis of the teachers' motivations for adopting AATs, how it is used to support teaching practice, teacher perceptions of AATs, or the impact of AATs on student behaviour and attitude.

3 USE CASES FOR AUTOMATED ASSESSMENT

In this section we briefly report case studies that illustrate the diversity of AATs we have used.

3.1 Tools and Infrastructure

We use several tools, both publicly available and locally developed, to support automated assessment, including:

CodeRunner This is a web-based plugin to the Moodle Learning Management System (LMS). It is designed to assess the correctness of programs using a set of test cases, and is highly configurable due to the use of templates that can be customised during the test case evaluation [19]. Being part of Moodle, assessment can consist of CodeRunner-based questions, as well as other question types supported in Moodle quizzes.

LMS Quizzes Commonly used LMSs , such as Canvas[1] and Blackboard[2], support a variety of standard question types such as multiple-choice and short-answer questions that can be marked automatically.

Dividni Dividni [21] supports the creation and delivery of individualised assessments, which cannot scale to large classes without an AAT. Dividni also allows creation of question banks that can be exported to LMS systems (such as Canvas and Moodle) or digital assessment platforms (such as Inspera[3]) leveraging the automatic assessment facilities these systems and platforms provide.

Perusall[4] This is an online social learning platform where students can engage with their peers and experience the course material collaboratively.

STACK System for Teaching and Assessment using Computer algebra Kernel [26], provides assessment of linear algebra understanding, and formative feedback that supports student learning.

OpenGL We use an extension to CodeRunner that allows student code to be included in an OpenGL program skeleton and evaluated in a sandbox running MESA. Students see generated images and test results. Student solutions are compared to a sample solution using pixel colours of the output image, OpenGL invariants (such as the model-view matrix), and program output (function calls, parameters, and results).

HCI We use *MarCHIr*, a first-of-its-kind locally produced tool for automatically marking Human Computer Interaction submissions. This tool assesses web sites created by students for use of an assigned colour, and adherence to accessibility principles including labeling, alternative text for images, and colour contrast. It also assesses the form for adherence to visual design principles, and tests basic interactive functionality such as a pop-up window.

Standard Test Frameworks We use standard unit test frameworks (e.g. JUnit) to support both unit testing and system testing. The system testing is supported by providing a layer to pass system-level actions to the JUnit-managed tests.

Bespoke Test Frameworks We have developed our own test frameworks, in particular to support system testing that does not use standard test frameworks. These typically run from the command line and control input, monitor output, and compare the output to a specification of what is expected.

Build tools We use build automation tools such as `make` and `maven` to manage other test frameworks, or to provide extra test support such as for integration testing.

GitHub We use the continuous workflow actions provided by GitHub[5] to support automated assessment. This can used in conjunction with build tools, but the GitHub API provides access to the history of the student's work, allowing support for assessing such things as code review activity.

3.2 Automated Assessment of Programming

In this section we describe our experiences using systems that support automated assessment of source code.

[1] https://www.instructure.com/canvas
[2] https://www.blackboard.com
[3] https://www.inspera.com
[4] https://www.perusall.com
[5] https://github.com

Case 1 — CS1 Introduction to Programming.

Context: In a large introductory Python programming course, we use CodeRunner to automatically assess short programs in weekly laboratories, timed quizzes (30 minutes), and computer-based exams that have been used in both invigilated and uninvigilated modes.

Instructional design: All exercises and examination questions use the same process. When students submit their code, the system immediately reports which tests pass and fail, and displays the expected and actual output. Students may submit their program as many times as they wish, but a penalty scheme is applied after 2 incorrect submissions, reducing the mark by 5% for each subsequent incorrect submission to a maximum of 50% penalty. One or two example test cases are visible to the students, but most are not shown until the student submits a solution. A sample of typical test cases remain hidden to ensure solutions cannot be engineered simply to pass the tests.

Driver: Automated assessment is used in this context to provide an unlimited amount of feedback with clear unambiguous grading criteria that is assessed rapidly and consistently with low cost.

Reflections: We observe student feedback about automated assessment to be generally positive, but students do not typically see the output from the AAT as feedback, and rate feedback lower than most other categories in student course evaluations. Students need to be taught explicitly why we use AATs and how it works. Additionally, students should be given the opportunity to practice submitting to the AAT to reduce anxiety about using an automated tool for higher stakes assessments. Students tend to rely on the automated tests rather than develop their own tests, and become frustrated with failing hidden tests. The opportunity to resubmit code until they get the correct answer encourages students to keep working on their solutions until they pass all the tests, but students appear to conflate opportunity with expectation and some complain that the course requires them to spend excessive time to get full marks in all the assessed work.

Case 2 — CS1 Programming for Engineers.

Context: In a very large ($n > 1000$) programming course for engineering students, we developed a custom tool to provide scheduled formative feedback.

Instructional design: We publish a small number of interim deadlines prior to the final deadline for summative grading. Students can submit work in progress at any time prior to such a deadline. No feedback is provided at the time of submission, but once one of the interim deadlines is reached, our tool grades all submissions against a test suite and sends each student email that gives the number of passing tests across each task of the assignment. This format gives the student an indication of which tasks require attention, without revealing exactly which tests have failed.

Driver: The motivation for delaying feedback in this way is to help students develop good time-management skills. It is well known that many students put off working on large programming assignments until deadlines are imminent, and this procrastination can be a serious problem [5, 9, 12]. By scheduling the feedback opportunities, we incentivize students to design their own tests and to start their work early so that they can benefit from the feedback.

Reflections: From our experience, it is clear that the scheduled feedback opportunities markedly impact submission behaviour compared to when they are not used. Although we do not have objective data that students start their work earlier, which was our original goal, we certainly observe a sharp increase in submissions with around half of all students typically submitting work more than a week before the final deadline. We have also seen that at-risk students, who have poor grades leading up to the assignment, perform significantly better if they take advantage of the formative feedback provided [7].

Case 3 — Data Structures in Computer Science.

Context: Coderunner is used in large first-year data structures and algorithms courses (CS2) taught in Python. This course has weekly laboratory activities, which comprise a set of approximately ten programming problems.

Instructional design: In this course we assess efficiency by setting exercises that required students to use an efficient solution, because the naive solution would take too long and automatically generate a "time-out" message (e.g., require a sorting algorithm that sorted an input file with 100,000 elements).

Driver: The course uses an AAT primarily to provide students with immediate feedback while reducing the marking load.

Reflections: This is an effective approach, but the assessment constraints need to be carefully explained to students because they are not typically familiar with running time being a factor in the evaluation, and the code still produces the correct answer. This leads some students to question the accuracy of the automated assessment results.

Case 4 — Data Structures and Algorithms using OOP.

Context: We use a bespoke process involving unit tests in two large software engineering courses (300+ students) teaching object-oriented programming, data structures and algorithms (CS2) taught in C++ (in one course with `make`) and Java (in another course with `maven` and JUnit).

Instructional design: Unit tests are shipped along with assignment base code, allowing students to run the tests "offline" as part of their development setup. The approach students follow is akin to test-driven development, except most of the test cases are provided to them. A build system (`make` or `maven`) is used to execute test cases. The same test cases are used, along with additional test cases, for summative assessment purposes after students submit their code for grading. The complete set of test cases are then shared with students, so they can check where they lost marks. Although GitHub is used to manage assignments, testing remains offline as part of the local build. For final assessment, scripts are used to run all submissions and email results to students.

Driver: We aimed to reduce grading time and reduce ambiguity of the specifications described in the instructional handouts, since students can use the tests to determine exactly what is expected.

Reflections: Taking a fully offline approach has worked extremely well for these large courses. The assignments are changed each year, but the same unit test setup is used. This makes generating new assignments easier every year as they are "molded" within this same template. Including the test cases along with the assignment code means that students are encouraged to set up a professional development environment, including the compiler,

build tool, versioning with Git, and retain flexibility to use their preferred IDE. Running test cases is trivial and convenient within the IDE. Students like the objective assessment, especially as the first assignment gives them all the test cases used in the summative assessment. As the course progresses, we provide a smaller proportion of test cases to encourage students to test on their own.

While setting up the development environment is a valuable lesson for students, there will be some that struggle to install some of the necessary tools. To overcome this, drop-in help clinics are provided for students get assistance, and university's computers with all the necessary software are available.

Case 5 — Object Oriented Programming.

Context: A large (300+ students) second-year object-oriented programming course for software engineering students, covering fundamental OOP design concepts using Java. A custom JUnit scaffolding library was implemented to allow system-level test cases for marking programming assignments implemented as Command Line Applications (CLIs).

Instructional design: Students get half the tests, with the remainder revealed after the deadline. All tests are used for marking. The marking is performed by a script that uses `maven` to build the submission and run the tests. The script determines the number of passing and failing tests, which the script converts to a mark. The script then emails their marks and report to each student.

Driver: The primary motivation for using system-level test cases, instead of unit test cases, is to give students freedom and experience in applying OOP design. A test case is simply a sequence of commands, and test oracles that verify the absence/presence of specific strings in the console. Under the hood, our bespoke system simulates the user of the application that types the commands on the keyboard one by one and captures the output of the console to check the absence/presence of the specified strings.

Reflections: Using test cases to mark assignments has worked very well, although some students initially struggled with the test suite. As students have access to half the tests, passing the provided test cases before submitting the assignment is a confidence booster for students. Our system-level test cases were well received by students. Students need to make sure their CLI application produces the correct output given the specified input, oblivious to implementation details. The system-level test cases are very easy to understand and interpret because each test case shows the input typed in the console and what output we expect to observe.

Case 6 — Large Scale Software Development.

Context: This is a third-year course with about 100 students looking at issues involved in large-scale software development, with a focus on web development. We use GitHub continuous integration (CI) workflow actions to execute a `maven pom` specification to run both unit and integration tests. The integration tests involve setting up a web server, installing the student implementation of a REST interface, and then running the REST tests. This structure is used for small exercises in weekly labs and a large team project.

Instructional design: Students are given all artefacts needed—JUnit test classes (for both unit and integration tests) and `pom` specification, with no missing or hidden tests. This is deployed through GitHub Classrooms. This allows them to run the tests in their local environment. The final assessment is performed at the due date for the assessment item based on the current state of the students' repositories.

Driver: While the primary motivation is to provide an objective means for students to check whether their implementation is valid, this approach also reduces the marking workload, and ensures consistency.

Reflections: In the initial offering of this course, some project teams used the CI workflow as their build process rather than doing everything locally. We then imposed a limit of the number of times they could trigger the CI workflow and that appears to have resolved the problem.

Case 7 — Computer Graphics.

Context: We use a bespoke plugin to the CodeRunner tool in a large (300+) third year Computer Graphics course which includes: OpenGL primitives, illumination and shading, 3D transformations and modelling, texture mapping, ray tracing, and parametric curves and surfaces. The module uses C/C++ for programming tasks.

Instructional design: The course assessment involves weekly programming quizzes (a graded and an optional ungraded one) and a mid-term test and final exam, all of which use AA. Students have unlimited attempts and can see their marks immediately.

Driver: We use an AAT to increase the quantity, timeliness, consistent, and availability of feedback. Using an AAT allows us to have weekly quizzes with immediate feedback. Human markers often do not notice problems with image output when grading student work, while an AAT provides more consistent results and fewer complaints. Using an AAT provides feedback when students experiment by themselves. Finally, using a web-based AAT avoids the need for local installation of OpenGL, which some students struggle with.

Reflections: Feedback from students indicated that those with limited experience and equipment were able to use the web-based tool easily, and students were generally positive about the instant feedback provided.

The tool improved teaching since it made lectures more interactive, and students could test their understanding using the weekly quizzes. Analysis of usage patterns suggest that students spend more time on practicing content using short automarked exercises than they spend on larger manually-assessed assignments.

Several issues were observed: (a) the AAT lacks formative feedback; (b) sometimes students were unable to see the difference between expected and actual output in the images which caused frustration; (c) some students use a trial-and-error approach to produce the expected output; (d) it is difficult to determine partial marks; (e) it was sometimes challenging to detect plagiarism; (f) some of the top students felt that developing a substantial solution from scratch would provide a stronger understanding of the subject.

Case 8 — Web Development.

Context: This is a large (450+ students) third-year web development course in which students get a full-stack development experience. The server backend is developed using C# and consists of practical work constructing APIs. The frontend uses JavaScript (and CSS, SVG, and HTML).

Instructional design: The course has a total of six scaffolded assignments, each building on the previous. Two of the assignments require students to develop open and authenticated APIs. The rest

of the assignments require students to develop a frontend using these APIs, and penetration-test the server backend. There are two time-limited quizzes that assessing the students' knowledge of constructing the server backend and the client.

Driver: The primary driver for automation is the quality and consistency of assessment. At the same time, the automation yielded prompt feedback.

Reflections: The assignments and quizzes have previously been manually graded. There were number of grading consistency issues, resulting in lengthy re-marking exercises, and delayed feedback. This prompted us to develop a bespoke grading tool for testing and marking the server backend development. The tool is completely automated, and resulted in reducing the re-marking requests from over 50% to under 5%. It is more difficult to automate grading of the client frontend, due to the numerous user-interface elements the frontend has. Therefore we could only partially automate the assessment of the assignments related to the frontend.

3.3 Automated Assessment of Non-Programming Artefacts

In this section we share our experiences with automated assessment systems that evaluate student work that does not include source code.

Case 9 — Computer Systems.

Context: We use Moodle Quizzes and CodeRunner in our first-year and second-year computer systems courses. The first-year course (400+ students) provides an overview of the layers that make up a modern computer system. It uses Moodle Quizzes to run weekly tutorials and several assignments. The second-year systems course (200+ students) teaches caches, virtual memory, assembly language programming (using LC-3), and how a high-level language (C) is implemented at the machine level. Moodle is used for weekly quizzes, and CodeRunner for C programming labs and the LC-3 assignment.

Instructional design: In the first-year course, some quizzes are entirely auto-marked and others include essay and short-answer questions that are manually graded. The second-year course allows the students to take weekly quizzes on Moodle with multiple attempts, where we use a question bank to randomise the questions presented in each attempt. After two trial attempts, the third attempt is graded. The LC-3 assignment and C programming labs use CodeRunner to assess the functional correctness of solutions.

Driver: We use AATs primarily to provide timely feedback as students get immediate results. This gives students an opportunity to improve their understanding of current content so they are prepared for what follows. The use of AATs also reduces the marking load, and provides individualisation to support academic integrity.

Reflections: The weekly automated quizzes help students to remain engaged with the course material, and reinforces their learning. Our use of the multiple attempts with randomised questions in the quiz were appreciated by the students, who report that it helps them to practice key concepts.

Case 10 — Human Computer Interaction.

Context: The tool *MarCHIr* is used in a large (300+ students) third-year human computer interaction (HCI) course that includes an assignment on the design and development of a high fidelity web prototype. The learning outcomes include an understanding of usability, accessibility, and visual design as well as the ability to creatively apply and implement these within an interactive web prototype.

Instructional design: When students initially receive the handout for the assignment, *MarCHIr* assigns a personalised colour to every student which is provided through a batch email. Then, students develop a website with a homepage and a form using their assigned colours as the 'brand' colour. The homepage has few constraints, inviting creativity, whereas the form requires placement of a prescribed number of web elements.

MarCHIr assesses the entire prototype for use of the assigned colour and adherence to accessibility principles including labelling, alternative text for images, and colour contrast. It assesses the form for adherence to visual design principles as well as tests basic interactive functionality such as a pop-up window. In order for *MarCHIr* to work submissions must conform to a small set of naming conventions. Students can pre-test their prototype to verify that it meets all the naming requirements. After submission, students receive feedback comprising half automatic grading from *MarCHIr* and half the marks and written comments from manual markers.

Driver: We created *MarCHIr* to provide efficient marking for large cohorts and to offer objective marking for HCI content that is typically considered subjective by our technical computer science and engineering students. Our motivation to include personalisation was to prompt individual creativity and to prevent cheating.

Reflections: *MarCHIr* successfully assesses the implementation of an interactive web prototype and adherence to accessibility standards and visual design principles, making it a useful tool alongside manual marking. While we invested more time upfront to develop *MarCHIr*, we saved significant marking effort after the fact. The course evaluations showed fewer comments complaining about subjective marking, compared to earlier years.

Case 11 — Cybersecurity & Computer Networks.

Context: Personalised and AATs are used in the Cybersecurity and Computer Network courses, which include a second-year course and two third-year courses. The third-year courses use one individualised assignment and two individualised tests each, while the second-year course uses 11 assignments and 2 quizzes, all individualised. Individualisation is facilitated through the Dividni [21] framework.

Driver: In these courses individualised assessments are primarily used to reduce academic misconduct. In addition, automated grading provides consistency and fairness that is difficult with manual grading.

Instructional design: Individualised quizzes and assignments are generated using Dividni [21]. Quizzes are imported to Canvas or Inspera, both of which facilitate auto-grading. Assignments are deployed to a local server where single-sign on identifies the students and provides idempotence to the assignments. Students submit their solutions to a bespoke submission tool and the submissions are auto-graded using Dividni.

Reflections: Developing individualized assignments for large classes requires careful consideration so that the grading of such questions can easily be automated. AATs enable multiple grading

at no additional cost, so we could allow students to re-attempt tests and unsuccessful assignment questions without significant workload.

Case 12 — Object-oriented Analysis and Design.

Context: Blackboard tests are used in a second-year object-oriented analysis and design course that focuses on requirements elicitation and expression (semi-formal grammar and fully dressed use cases) and using UML models (use case, sequence, activity, and class diagrams) to define and clarify requirements.

Instructional design: A typical assignment involves an authentic case study of a ticketing system with an external client who had implemented similar systems. The case study had some hidden requirements and students were able to ask the client for clarification throughout the assignment. The assessment was presented as a series of incomplete models where students had to fill in the missing elements. Each group of potential responses included distractors that were based on common errors from past students. To correctly complete the models the students needed to be able to evaluate the potential solutions and map them to the case study requirements.

Driver: The main motivation for using automated assessment was to ensure efficiency when marking complex models in a large class. Consistency between markers was previously an issue, regularly resulting in the need to remark large sections of work, even when highly structured grading flow charts were introduced.

Reflections: Students found assessment manageable but the automated assessment entailed removing tasks which required them to use appropriate model elements and deal with laying out the diagrams. These aspects were instead tested in controlled quiz assessments. Consistency in marking was achieved and it was possible to easily regrade a question for the entire cohort if required. The grade average for the assessment was inflated but moderated by the controlled quizzes. It is recommended that anyone taking this approach include one diagram that is created and marked by hand. However, this approach has higher potential for undetectable plagiarism. Additionally, designing and testing the assignment takes time since every potential response needs to be tested in each gap. But the time to develop the assignment was substantially less than marking the more traditional assessment that required students to write requirements and create diagrams from scratch. Despite students needing to apply concepts and evaluate potential solutions to complete the assignment some colleagues remain unconvinced that this a reasonable format for assessment of modelling skills.

Case 13 — Object-oriented software development.

Context: In a large (300+ students) second-year software development course taught in Java, we created a reading assignment using Perusall.

Instructional design: A book chapter and a problem description for a software project is presented to the students in Perusall. Students are expected to work in small teams, discuss the problem description within their groups, and add 5–10 relevant and meaningful annotations to specific parts of the problem description, citing knowledge gained from the book chapter. The automarking feature of Perusall is complemented with a manual review of students comments by the instructor. The tool assessed students' submissions on the basis of such things as annotation content, number of assignment accesses, student engagement, time spent on the reading and annotating, number of comments made, and length of comments.

Driver: The intent of using Perusall was: (a) to enhance students' understanding of object-oriented concepts through readings; (b) introduce a group discussion-based assessment component to develop communication skills; (c) reduce the manual marking effort.

Reflections: Automation was essential to provide efficient assessment of student engagement with reading material. Manually grading student comments using an alternative, such as a forum, would be too costly, so the use of AATs for this task was essential. Feedback from students was positive with the majority of students reporting that the reading supported their learning of object-oriented concepts.

Case 14 — Collaborative learning of mathematics.

Context and Driver: To major in CS, probably all students must take a range of mathematics courses (e.g., Linear Algebra). By the end of these courses, which are often lecture-based, fast-paced, and content-intense, students are expected to become fluent in complex and abstract topics. To help students reach these targets many university teachers capitalize on the affordances of learning-support modules (sometimes also called "quizzes"). Such modules can provide explanations and hints as well as mark student solutions. Mathematics education research has developed a comprehensive body of knowledge on ways of unconventional thinking that students often develop (e.g., [18]), so if an input answer is consistent with a known alternative approach, a module can trigger a feedback message to help students revise their thinking. However, designing such feedback is an effort-consuming endeavour. Accordingly, an out-of-course pilot was initiated to explore how ATs could induce a desired change in students' thinking.

Instructional design: We invited dyads and triads of first-year students to participate in a pilot. In a laboratory setting, each group sat in front of a single computer screen and worked together on learning-support modules developed in STACK. Each module consisted of a sequence of questions, where each question appeared on a separate screen. Once the students clicked the "check" button, the module automatically assessed their answer and presented feedback based on previous research. We designed the modules so that students could progress to the next question without submitting their answer, return to the previous question, and attempt each question several times. Our focus was on students' collaborative engagement with feedback and its use to inform their thinking.

Reflections: Overall, the design of the modules "paid off" as feedback messages often assisted students in revising their thinking and correcting their work. The use of similar questions between which the students cycled back and forth played a key role in students becoming confident that their revised or current approach was correct. In some cases, corrective feedback spurred a change in students' reasoning even if the explanation was not consistent with group thinking. In those cases where a group was split in its thinking, feedback occasionally spurred productive discussions around why one rather than the other approach was correct, but in other cases the group dynamic was an obstacle to learning. Thus, collaborative learning from AATs emerges as a complex phenomenon that is shaped by a range of cognitive, meta-cognitive, social, and affective factors.

Case 15 — Mathematics in Computer Science.

Context: H5P (HTML5 Package) allows the creation of interactive content within Canvas and is used in the first-year discrete mathematics course. Interactive content with multi-choice quizzes is created (sourced mainly from past tests and exam papers) for students to practice in their own time.

Instructional design: In this course, students are asked to voluntarily complete a module as revision for a chapter on the relevant material. The interactive quizzes provide both answers and hints to resources when they click on an (incorrect) option. The quiz did not contribute to the final grade.

Driver: Many students find discrete mathematics challenging and have trouble translating concepts learned in class to application problems on assignments. Introducing these quizzes aims to enhance independent student learning by providing immediate feedback and increasing their confidence in problem-solving.

Reflections: Students have reported that the quizzes help them solidify their understanding as they get immediate feedback to self-correct their thinking process. However, some students chose not to complete the activity as it was optional.

4 DISCUSSION

As the case studies illustrate, there are many variables relating to the decision to use automated assessment. We summarise that variety below.

4.1 Nature of Assessment

As would be expected for courses in computer science and software engineering, providing automated functional testing was the most common form of assessment. However there was variation even in this group. In some cases, the tests were applied to a single method or a single class, that is, unit testing. In other cases the tests were applied to a set of files or classes and the testing was done at system level, exercising the system from an entry point (e.g., the `main` method in Java).

AAT are not limited to functional correctness. One course automatically assessed the fidelity of a web site prototype using standards in HCI and accessibility principles (Case 10). One course tested principles in graphics, such as image creation and transformation. For some courses, part of what was assessed was whether the proposed solution met algorithm efficiency requirements.

There were variations on non-code based assessment. Some were the familiar MCQs or short-answers, but one course experimented with supporting learning linear algebra, while another assessed engagement in group discussion.

4.2 Drivers

These cases are reported in institutions that have large class sizes and limited teaching support. Unsurprisingly the most common reason to use AAT is to reduce the cost of assessment, but it was not universal. For one course (Case 14) on linear algebra the motivation was to support student learning by providing structured interactive and formative feedback. A consequence of reducing the cost is that it allows assessments to be regraded cheaply (e.g., to allow provisional marks with re-submission, see Case 11).

The next most common reason was to provide immediate feedback to student work. As we have already noted, providing feedback is an important part of learning, and so doing so as quickly as possible would seem a reasonable goal. However this property also could engender unhelpful student behaviour where they rely solely on the AAT to complete the assessment (Case 6). To avoid this behaviour, but to help develop time management skills, AAT feedback was only provided at limited interim deadlines. The "reward" of the feedback encouraged an early start while avoiding degenerate behaviour (Case 2).

The criteria used for AAT provides an unambiguous specification for what was required for the assessment. This helps mitigate the difficulty of fully describing the functional requirements in natural text (Case 4). A consequence of this, and in some cases the main driver, was to resolve issues associated with accuracy of marking.In some cases, AATs identify subtle errors that are difficult for a human to detect (Case 7). Also, with human markers, there is the concern that they are not always consistent in their interpretation of the marking rubric (e.g., Cases 8, 12).

AATs can have pedagogic benefits. Having the AAT feedback immediately allows students to experiment with their submissions to help understand the consequences of different choices, such as what happens when different rotation parameters are provided for an image (Case 7). Having AAT feedback provides safety to the students to allow them to explore variations relating to the main criteria (Case 15).

An important pedagogic benefit that AAT allows is individualisation, where the assessment task a student has to perform is tailored to that individual. This is a non-punitive way to support academic integrity. However, grading individualised assessments manually is expensive and does not scale. AAT enables individualised assessments to scale to large classes (Cases 9, 11).

Some AATs improve accessibility to learning content. For example, several systems are delivered via a web-browser, which allows students to do their assessment on any web-enabled device. When the environment required is complex (e.g., Case 7), the AAT may remove significant barriers to learning.

4.3 Assessment Provision

Several different AATs were used (Section 3.1). The student interaction with the AAT can be fully online (i.e. students interact with a web application in a web browser and receive the AAT feedback in the same way), partially online (work is submitted via some web-based or online system but is assessed in by a separate system as a separate step), or fully offline (work is submitted directly to the AAT).

The availability of AAT feedback depends to some extent on the student interaction. If online, the feedback will typically be immediate and directly to the student, usually as a web page. This is typical of tools such as CodeRunner, Perusal, and other LMS-based AATs.

For partially online, when the feedback is available depends on when the assessment step is performed. Typically this step is performed by the teacher or assessment team and some days may have elapsed from when the work was submitted. The report may be only used by the assessment team to determine the final mark

and not provided to the student. Or it could be delivered to the student via a separate means (e.g., email Cases 2, 4, 10).

For offline, for the student to have access to the feedback report requires that the student has access to the AAT. In some cases the assessment was performed only by teachers with availability similar to the partially online case. However for some courses the students received all the necessary artefacts to allow them to use the AAT themselves (e.g., Cases 4, 6).

4.4 Assessment Type

The types of assessment managed by the AATs included "assignments" (deliverable worked on by an individual student in own time over more than a week) "lab exercise" (deliverable worked on by an individual student starting at a defined time—the "lab"—but possibly completed outside of the lab time) "test/exam" (limited time to work on it typically not more than 3 hours at a fixed time of the day), "tutorial exercise" (primarily formative exercise done in a teaching session) "weekly quiz" (formative but not necessarily done during a teaching session), and "project" (several weeks by a group).

The nature of the different types of assessment affect the choice of AAT and how it is used. For example, in-class exercises necessarily require immediate feedback. Offline AATs typically require some setup time that is not worth spending in-class, meaning an online system is the best choice.

Online assessment is typically impossible for certain assessment types. For example, system-level testing, especially with complex input and output, is difficult to provision through a web browser. Accordingly, this type of assessment typically uses a separate tool, that is, either partially or full offline (e.g., Cases 2, 5, 4, 6, 10).

The advantage of standard tools such as JUnit, `make`, or `maven` is that their deployment is well supported. Teachers need only provide the specification of the process (e.g., JUnit test classes, the makefile, or a maven POM file) and there are many resources available to students to help them get the systems running. With locally-created bespoke systems, their deployment can be non-trivial, and so are typically only used for situations where the students do not directly interact with the AATs.

4.5 Assessment Criteria Visibility

The visibility of the criteria used by the AATs can vary, from all criteria known to the student to nothing being visible—the students may not even be aware that an AAT is being employed.

Supplying all criteria was useful when the criteria was used to provide an unambiguous specification of the requirements (e.g., Case 7). However, when performing functional testing, it was common to have hidden tests (e.g., Case 1). An alternative is to not supply all of the tests that were to be used in the assessment. The students would get some of the tests but these were deliberately incomplete in their coverage, with a full test suite employed for the to complete the assessment (e.g., Cases 1, 2, 5, 4).

4.6 Assessment Outcome

The feedback produced by the AAT could depends on several factors, including who it is to be delivered to, what the visibility of the criteria is, what the goal of the assessment is, and the type of assessment. For example, when provided to students, it may be only a summary (e.g., Case 2); it may describe the criteria met (e.g., what tests, including inputs and expected outputs) were met; it may be the results for all criteria applied (details of tests that both passed and failed). It could even provide further feedback and advice on performance (e.g., Case 14). The different choices provide different levels of formative feedback.

The reports for teachers generally contain the results for all criteria, and may include extra information or provide aggregate data on class performance.

4.7 Submission Options

A decision that has to be made is how and when students submit their work. If the goal of its use is to provide formative feedback, then there will typically be no restriction of when, or how often, students may make a submission. For example, in an in-class setting with no summative component allowing unlimited submissions would be appropriate. However when the assessment mark is partially or wholly determined by the AAT and unlimited submissions are allowed, we found students can come to rely on it (e.g., Case 1). Options include limiting the number of submissions, have a penalty scheme for multiple attempts (Case 1), or limit when feedback is provided (Case 2).

When assessment is intended for both formative and summative feedback, this creates a tension as to what the right restrictions are, and potential unhappiness in students (Case 1).

5 RECOMMENDATIONS

In this work we have presented a variety of ways that automated assessment is used in contemporary computing courses, and share the experiences of teachers involved in these courses. We conclude with a set of recommendations based on those experiences.

> **Recommendation 1:** Familiarise students with the AAT environment prior to high stakes assessments.

Students sometimes delay starting assignments until close to deadlines, and discover too late that they do not know how to effectively use the AAT. A low-weighted activity scheduled early in the course, that mimics the AAT use of later assignments, will help to build student confidence and familiarity with assessment processes and resolve mechanical issues at a time that is not critically important.

> **Recommendation 2:** Provide a sample of test cases with the assignment to scaffold and encourage testing outside the automated assessment environment.

To encourage students to focus on testing their own code prior to submission to an automated assessment system, consider providing a sample of test cases, along with any necessary build tool configuration, within the assignment resources. This supports students to start testing their code with minimal effort.

> **Recommendation 3:** Ensure some test cases are hidden, but visible test cases should cover boundary cases and complex inputs/outputs.

Hidden test cases are needed to ensure students cannot engineer their solution to pass tests without creating a solution to the general problem specified. However, it can be frustrating for students when a hidden test covers an unusual case that they had not considered, especially when the grading scheme is all-or-nothing. The test cases should therefore be selected to make it unlikely that a genuine solution will pass the visible cases and fail the hidden ones.

> **Recommendation 4:** Consider using a mechanism to prevent unlimited submissions for the same problem to prevent undesirable trial-and-error behaviours.

The availability of unrestricted immediate feedback can lead to over-reliance on the tests provided by auto-graders [2]. Frequent mechanisms to reduce over-use of AAT feedback include limited submission opportunities, penalties for re-submission, time delays, or parameterised problem sets.

> **Recommendation 5:** Consider using features of automated assessment system to provide feedback that explicitly encourages students.

Positive feedback is known to improve student motivation and self-perception, and can improve learning performance[10], yet many of the automated systems emphasise failure to meet criteria. There is some research suggesting that although negative feedback can improve student self-assessment accuracy, it can also reduce student engagement and motivation [17, 30]. Hyland and Hyland [14] suggest that a combination of both positive and negative feedback is most effective, and that positive praise may result in students being more accepting of negative feedback. Given the focus of AATs on correctness, it is our view that systems should aim to include both positive feedback on correct elements as well as gaps between desired and actual performance.

REFERENCES

[1] Kirsti M Ala-Mutka. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (2005), 83–102.

[2] Elisa Baniassad, Lucas Zamprogno, Braxton Hall, and Reid Holmes. 2021. STOP THE (AUTOGRADER) INSANITY: Regression Penalties to Deter Autograder Overreliance. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. ACM, New York, NY, USA, 1062–1068. https://doi.org/10.1145/3408877.3432430

[3] Leopold Bayerlein. 2014. Students' feedback preferences: how do students react to timely and automatically generated assessment feedback? *Assessment & Evaluation in Higher Education* 39, 8 (2014), 916–931.

[4] Janet Carter, Kirsti Ala-Mutka, Ursula Fuller, Martin Dick, John English, William Fone, and Judy Sheard. 2003. How Shall We Assess This? *SIGCSE Bull.* 35, 4 (June 2003), 107–123. https://doi.org/10.1145/960492.960539

[5] Sophie H. Cormack, Laurence A. Eagle, and Mark S. Davies. 2020. A large-scale test of the relationship between procrastination and performance using learning analytics. *Assessment & Evaluation in Higher Education* 45, 7 (2020), 1046–1059.

[6] Galina Deeva, Daria Bogdanova, Estefanía Serral, Monique Snoeck, and Jochen De Weerdt. 2021. A review of automated feedback systems for learners: Classification framework, challenges and opportunities. *Computers & Education* 162 (2021), 104094. https://doi.org/10.1016/j.compedu.2020.104094

[7] Paul Denny, Jacqueline Whalley, and Juho Leinonen. 2021. Promoting Early Engagement with Programming Assignments Using Scheduled Automated Feedback. In *Australasian Computing Education Conference* (Virtual, SA, Australia) *(ACE '21)*. ACM, New York, NY, USA, 88–95. https://doi.org/10.1145/3441636.3442309

[8] Christopher Douce, David Livingstone, and James Orwell. 2005. Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)* 5, 3 (2005), 1–14.

[9] Stephen H. Edwards, Joshua Martin, and Clfford A. Shaffer. 2015. Examining Classroom Interventions to Reduce Procrastination. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. ACM, New York, NY, USA, 254–259.

[10] Emily Faulconer, John Griffith, and Amy Gruss. 2022. The impact of positive feedback on student outcomes and perceptions. *Assessment & Evaluation in Higher Education* 47, 2 (2022), 259–268. https://doi.org/10.1080/02602938.2021.1910140

[11] Graham Gibbs and Claire Simpson. 2005. Conditions under which assessment supports students' learning. *Learning and teaching in higher education* 1 (2005), 3–31.

[12] Yoshiko Goda, Masanori Yamada, Hiroshi Kato, Takeshi Matsuda, Yutaka Saito, and Hiroyuki Miyagawa. 2015. Procrastination and other learning behavioral types in e-learning and their relationship with learning outcomes. *Learning and Individual Differences* 37 (2015), 72–80. https://doi.org/10.1016/j.lindif.2014.11.001

[13] John Hattie and Helen Timperley. 2007. The Power of Feedback. *Review of Educational Research* 77, 1 (2007), 81–112. https://doi.org/10.3102/003465430298487

[14] Fiona Hyland and Ken Hyland. 2001. Sugaring the pill: Praise and criticism in written feedback. *Journal of Second Language Writing* 10, 3 (2001), 185–212. https://doi.org/10.1016/S1060-3743(01)00038-8

[15] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10)*. ACM, New York, NY, USA, 86–93. https://doi.org/10.1145/1930464.1930480

[16] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.

[17] Eun Jung Kim and Kyeong Ryong Lee. 2019. Effects of an examiner's positive and negative feedback on self-assessment of skill performance, emotional response, and self-efficacy in Korea: a quasi-experimental study. *BMC medical education* 19, 1 (2019), 1–7.

[18] George Kinnear, Ian Jones, Chris Sangwin, Maryam Alarfaj, Ben Davies, Sam Fearn, Colin Foster, André Heck, Karen Henderson, Tim Hunt, Paola Iannone, Igor' Kontorovich, Niclas Larson, Tim Lowe, John Christopher Meyer, Ann O'Shea, Peter Rowlett, Indunil Sikurajapathi, and Thomas Wong. 2022. A collaboratively-derived research agenda for e-assessment in undergraduate mathematics. *Int. J. Res. Undergrad. Math. Ed.* (8 2022), 31 pages. https://doi.org/10.1007/s40753-022-00189-6

[19] Richard Lobb and Jenny Harlow. 2016. Coderunner: A Tool for Assessing Computer Programming Skills. *ACM Inroads* 7, 1 (feb 2016), 47–51. https://doi.org/10.1145/2810041

[20] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[21] Sathiamoorthy Manoharan and Ulrich Speidel. 2020. Individualized Assessments using Dividni-Enhancing Learning via Assessments Unique to Every Student. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, New York, NY, USA, 1419–1419. https://doi.org/10.1145/3328778.3372545

[22] Emma Mulliner and Matthew Tucker. 2017. Feedback on feedback practice: perceptions of students and academics. *Assessment & Evaluation in Higher Education* 42, 2 (2017), 266–288. https://doi.org/10.1080/02602938.2015.1103365

[23] David J. Nicol and Debra Macfarlane-Dick. 2006. Formative assessment and self-regulated learning: a model and seven principles of good feedback practice. *Studies in Higher Education* 31, 2 (2006), 199–218. https://doi.org/10.1080/03075070600572090

[24] Claudia Ott, Anthony Robins, and Kerry Shephard. 2016. Translating Principles of Effective Feedback for Students into the CS1 Context. *ACM Trans. Comput. Educ.* 16, 1, Article 1 (jan 2016), 27 pages. https://doi.org/10.1145/2737596

[25] José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated Assessment in Computer Science Education: A State-of-the-Art Review. *ACM Trans. Comput. Educ.* 22, 3, Article 34 (jun 2022), 40 pages. https://doi.org/10.1145/3513140

[26] Chris Sangwin. 2013. *Computer aided assessment of mathematics*. Oxford University Press.

[27] Draylson M. Souza, Katia R. Felizardo, and Ellen F. Barbosa. 2016. A Systematic Literature Review of Assessment Tools for Programming Assignments. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. IEEE, 147–156. https://doi.org/10.1109/CSEET.2016.48

[28] Jacqueline L. Whalley and Anne Philpott. 2011. A Unit Testing Approach to Building Novice Programmers' Skills and Confidence. In *Proceedings of the 13th Australasian Computing Education Conference* (Perth, Australia) *(ACE '11)*. Australian Computer Society, Inc., Australia, 113–118.

[29] Benedikt Wisniewski, Klaus Zierer, and John Hattie. 2020. The Power of Feedback Revisited: A Meta-Analysis of Educational Feedback Research. *Frontiers in Psychology* 10 (2020). https://doi.org/10.3389/fpsyg.2019.03087

[30] Shulin Yu, Feng Geng, Chunhong Liu, and Yao Zheng. 2021. What works may hurt: The negative side of feedback in second language writing. *Journal of Second Language Writing* 54 (2021), 100850. https://doi.org/10.1016/j.jslw.2021.100850