Christopher Pyles\textsuperscript{1}, Francois van Schalkwyk\textsuperscript{2}, Gerard J. Gorman\textsuperscript{2}, Marijan Beg\textsuperscript{2}, Lee Stott\textsuperscript{1}, Nir Levy\textsuperscript{1}, and Ran Gilad-Bachrach\textsuperscript{1,3,4,5}

\textsuperscript{1}Microsoft  
\textsuperscript{2}Department of Earth Science and Engineering, Imperial College London, United Kingdom  
\textsuperscript{3}Biomedical Engineering Department, Tel-Aviv University  
\textsuperscript{4}Edmond J. Safra Center for Bioinformatics, Tel-Aviv University  
\textsuperscript{5}Sagol School of Neuroscience, Tel-Aviv University

\section*{ABSTRACT}

We continuously interact with computerized systems to achieve goals and perform tasks in our personal and professional lives. Therefore, the ability to program such systems is a skill needed by everyone. Consequently, computational thinking skills are essential for everyone, which creates a challenge for the educational system to teach these skills at scale and allow students to practice these skills. To address this challenge, we present a novel approach to providing formative feedback to students on programming assignments. Our approach uses dynamic evaluation to trace intermediate results generated by student’s code and compares them to the reference implementation provided by their teachers. We have implemented this method as a Python library and demonstrate its use to give students relevant feedback on their work while allowing teachers to challenge their students’ computational thinking skills.

\section{Introduction}

In her seminal paper, Wing \cite{29} argued that computational thinking is a skill that is not limited only to computer programming but that it has a much broader impact. Furthermore, Wing argued that computational thinking should be taught to any college freshman and should not be limited to a computer science major. Some policymakers argue that computational thinking should be taught in the compulsory education system. In Finland, for example, “algorithmic thinking” is a mandatory topic for first-grade students \cite{5}. Accordingly, computational thinking should be taught at a massive scale \cite{15}, with higher education playing a critical role in ensuring graduates have the necessary industry-transferable digital skills for the workplace \cite{1}. This makes tools for automated guiding of students as well as for providing automated formative (auto-assessment) and summative (auto-grading) feedback of paramount importance \cite{27, 30}. Indeed, the need for such tools has been identified over 60 years ago \cite{14}. The Adoption of Auto-Grading (AG) and Auto-Assessment (AA) tools has seen significant growth in recent years, and it is now widely recognized as an enabler of global-scale education pedagogy and academic support system \cite{12, 9, 17}. The recent growth of remote learning challenges educators to provide more support for unattended and asynchronous learning.

Many AA & AG tools are built around a set of unit tests \cite{11, 8, 23, 5, 16}. This method for dynamic evaluation provides a pass or fail score for the entire student’s code or for subsets of the code for which the API was specified in advance. This means that, to enable fine-grained feedback, the instructor has to provide a detailed description of the API – a process that has been referred to as unit test “scaffolding”, or microlearning \cite{24}. However, in many cases, this process deprives the students of a better discovery-oriented learning experience that a tutor and not an automatic unit test can provide.

As an example, a student who is asked to write a program to compute the median of elements in a list may get a scaffolded code skeleton, similar to the one presented in Listing\textsuperscript{1} and asked to complete missing parts.

\begin{lstlisting}[language=Python]
S = [8.0, 5.0, -1.0, 3.0, 5.0, 2.25]
\end{lstlisting}
Due to the limitation of the automated evaluation tools, the instructor provides the student with the architecture of the solution. Thus, the student is practising basic programming skills and is not challenged with fundamental computational thinking tasks such as decomposing the problem and considering multiple possible solutions. This may be a good exercise for novice students doing their first coding steps, but it is ineffective when practising student’s ability to come up with the architecture by themselves. On the other hand, a much less restrictive exercise for computing the median may ask the student to complete the code as presented in Listing 2, allowing multiple implementation patterns and approaches. However, this format makes it harder to generate constructive feedback automatically.

In this work, we present a new method to assess student’s work which we have implemented in the PyBryt library which is available at https://github.com/microsoft/pybryt. In PyBryt, the instructor is providing one or several reference implementations of the given assignment. The teacher annotates these reference implementations by marking both expected and not expected intermediate values to be computed by the student. The instructor can also specify feedback to be given to the student depending on whether these annotated values are present or absent. The PyBryt library executes the student’s code and traces intermediate values to the ones marked by the instructor. Based on the matching between the observed values and expected values, feedback is given to the student. This approach allows controlling the freedom given to students: the instructor can provide fine-grained steps and trace the student’s ability to follow instructions, or alternatively, the student can work on an open problem by breaking it into sub-tasks and practising computational thinking.

In Section 2 we present the concept of reference implementations in PyBryt and how they are used to assess students code. In Section 3 we present several examples for the use of PyBryt for providing feedback to students and for plagiarism detection. We conclude with a discussion in Section 4.

2 Reference Implementations

To provide feedback to students or to grade their solutions, the instructor specifies their expectations. In PyBryt, expectations are defined using a reference implementation (RI) – a set of conditions expected from the students’ code. RI’s are constructed by annotating series of values expected to be present in the student’s solution, even if not explicitly specified in the guidelines for the assignment. For example, suppose the student is asked to implement the median function. Apart from the correct return value, the instructor may expect the student to sort the list of numbers in their implementation and calculate the list’s length to determine if it has an odd or even number of items. Therefore, the instructor annotates these values in RI. In PyBryt, this is done as shown in Listing 3. When the student’s code is executed in the PyBryt environment, all intermediate values that are created by the student’s solution are scanned to find the values marked by the instructor in RI. The instructor has the option to provide success-messages to be presented to the student if their code computes the expected values and failure-messages to present if the values are missing. Therefore, the instructor can provide formative feedback that can encourage and focus the students on ways to improve.
import pybryt

def median(S):
    sorted_S = sorted(S)
    pybryt.Value(sorted_S,
    success_message = "Great: The list is sorted correctly.",
    failure_message = "Advice: The list was not sorted.")

    size_of_S = len(S)
    pybryt.Value(size_of_S,
    success_message = "Great: The size of the list is computed.",
    failure_message = "Advice: The size of the list is not computed to determine if the number of elements is odd or even.")

    middle = (int)(size_of_set / 2)
    is_size_of_S_even = (size_of_set % 2) == 0

    if is_size_of_S_even:
        return (sorted_S[middle -1] + sorted_S[middle]) / 2
    else:
        return sorted_S[middle]

Listing 3: Median example annotated by the instructor

It is important to note that only the instructor adds annotations to their RI. Students are not required to modify their code. Instead, PyBryt executes students code in the debugger framework and traces the memory footprint to search for the values marked in the RI. All intermediate values are extracted regardless of variable names or function names. Accordingly, the teacher can design their assignments without "scaffolding" or using microlearning [24]. Furthermore, the computational complexity of the solution can be evaluated using the number of steps taken to run the program, which is a consistent measure that is independent of the hardware on which the program is executed or other programs running on the computer simultaneously. PyBryt allows instructors to specify in their RI the expected computational complexity of the solution and provide students feedback when they do or do not match these expectations.

To demonstrate the way PyBryt works consider the student code in Listing 4 as a solution to the median assignment. The solution does compute the median correctly; however, it will trigger two failure messages since expected intermediate values are not computed. On the other hand, the solution in Listing 5 will trigger two success messages even though the student did not use the same variable names and sorted the list in-place as opposed to the RI where new memory is allocated. Nevertheless, PyBryt recognizes that the required values were computed by the student and triggers success messages.

Listing 4: Median example - student implementation with numpy. Although this solution computes the median correctly, PyBryt can identify that the student uses a library call to compute the median.

Listing 5: Median example - student implementation with in memory sorting

PyBryt traces values computed by students solutions even when they are not assigned to variables. Note that the second value the instructor marked in the median assignment is the length of the list, however, in Listing 5 this value is computed but never assigned to a variable. Nevertheless, PyBryt detects that the value was computed and triggers the success message. To achieve that, PyBryt first analyzes the program by parsing its Abstract Syntax Tree (AST) and assigns each intermediate result to a variable. Hence, PyBryt first does static evaluation of the student’s solution.
import pybryt

def max_of_list(S):
    sorted_S = sorted(S)
    pb_sort = "pybryt.Value(sorted_S)
    pb_sort.failure_message = "Advice: sorting the list adds unnecessary complexity"
    size_of_set = len(S)
    return (sorted_S[len(S) - 1])

Listing 6: Finding the maximal value in a list

and creates a version of the code where every intermediate result is assigned to a variable. Next, it runs the code in debug mode to trace every value assigned to a variable while executing the code.

2.1 Logic and temporal rules over values of interest

In some cases, the instructor may be interested in defining logical rules over the annotated values. For example, consider an assignment where the student is expected to find the maximum value in a list. The instructor may wish to verify that the student does not sort the list since this increases the complexity of the solution. Therefore, the instructor may define the RI as shown in Listing 6.

In the RI presented in Listing 6, the logical not rule is introduced using the ~ symbol. This means that pb_sort is satisfied when the sorted version of S is not encountered, and the failure message is triggered only when the value is present. PyBryt supports several logical rules: the or operator |, the and operator &, and the not operator ~. Logical rules are useful in many settings. For example, if the student can use several data structures to store the data, an or-statement between the different values can be used. In some cases, it may be easier for the instructor to support such cases using multiple RI’s as discussed in Section 2.2.

Besides logic operations, PyBryt also supports two temporal operators: before and after. If we assume values v1 and v2, v_before = v1.before(v2) will trigger the success message if, in the student’s solution, the first time v1 is satisfied precedes the first time v2 is satisfied. Similarly, v_after = v1.after(v2) expects v1 to occur after v2. Since this is a common use-case, PyBryt also supports collections. A collection is a list of annotated values expected to be present. The instructor can specify whether the student’s solution should generate these values in the same order as in the RI or the order in which the values are generated is not a requirement. Moreover, the result of a logical or temporal operation is another Value object, allowing rules to be chained to create more complex conditions when needed.

2.2 Multiple reference implementation

When evaluating students code, PyBryt first runs student’s implementation, traces and stores every value that is being computed, and finally compares them to those marked in the RI. Since the values created by the student’s implementation are stored, they can be compared to several RI’s. A single RI is restrictive since there may be multiple algorithms for solving a given problem. For the median example, one possible solution is to sort the list and find the item in the middle of the sorted list (or the average of the two middle items if the number of items is even). This solution has time complexity of \( O(n \log(n)) \). However, there are more efficient solutions, such as the randomized quick-select algorithm [13] or the deterministic pick-pivot algorithm [4]. These solutions are very different from the sort-based solution and therefore generate a different memory footprint. Therefore, PyBryt allows different possible solutions to be covered by separate RI’s.

Since extracting the memory footprint from students’ solutions may be time-consuming, PyBryt generates an object that stores all values so that comparison to multiple RI’s is more efficient. In addition, the RI can be "compiled" into an object that stores all the required values and logical rules to further reduce the time.

Another setting in which the use of multiple RI’s is useful is when a mistake in an early stage may lead to a deviation in the computation. For example, consider the assignment that asks the student to count the number of times a substring matching the pattern ’ab?d’ appears in the text such that the matching is case insensitive. A student that did not convert the text to lower-case will not match many of the intermediate results expected and therefore may get many
error messages. However, if this happens to be a common mistake, the teacher may create a RI for this type of “solution”. It will capture the error and be able to test other aspects of the solution.

Having multiple RI’s allows additional types of assignments that can be given to students to practice their computational thinking skills. For example, an assignment can ask students to implement two types of sorting algorithms. Note that PyBryt can verify that the solutions are different by verifying that they match different RI’s since they generate different memory footprints.

### 2.3 Computational complexity

Since PyBryt runs students’ solutions in debug mode, it also counts the number of steps taken during the execution of the code. This can be useful when students are learning about computational complexity. Students can try different implementations and see how their choices affect the number of steps taken. This can be used in multiple ways. The teacher can specify the required complexity or ask students to implement both bubble-sort and merge-sort and draw a chart showing the number of steps it took for their solution to complete as a function of the input size.

Counting steps can be used even for learning fine-grain optimization techniques. For example, students can be asked to find the prime numbers in a given list of integers. In this assignment, caching can be used to accelerate the computation. Therefore, students can be asked to find an efficient solution, and PyBryt can be used to provide them with feedback about the efficiency. This can also be used for gamification by creating a class challenge to find a solution that uses the fewest instructions. Note that since students may be using different computers, measuring the time to execute may give an unfair advantage to students that have access to faster computers, but PyBryt can measure the complexity independent of hardware.

### 2.4 Programming style

PyBryt can be used to provide feedback helping students to improve their programming style. Consider, for example, an assignment where the student is asked to implement a function that takes a list of numbers and returns a list of these numbers squared. In Listing[7] two possible solutions are presented. Both solution1 and solution2 in Listing[7]are correct. However, solution1 using list-comprehension is more readable and thus preferred. To see which solution the student used, it is possible to check the intermediate values. When using a loop, as solution2 does, the result list is gradually built and therefore it starts as an empty set, followed by having the value [1], then [1, 4] and finally [1, 4, 9]. However, list comprehension creates the list in a single step. Therefore, the value [1, 4] , as an example, will never be observed, although the value [1, 4, 9] will, and the instructor can mark the value [1, 4] to identify the use of a for-loop. It should be noted that static methods for evaluating programs can also be used to identify different solutions. However, the benefit of using PyBryt for this task is the fact that the same tool and syntax can be used in many scenarios, which makes it easier for the instructor to use.

```python
1 def solution1(lst):
2     result = [x*x for x in lst]
3     return result
4
5 def solution2(lst):
6     result = []
7     for x in lst:
8         result.append(x*x)
9     return results
```

Listing 7: Two methods for computing the square of every item in the list

### 2.5 Tolerances and Invariances

A possible drawback of using PyBryt’s approach is tolerance to slight fluctuations in results. For example, when using real numbers, small numerical errors may occur due to the finite precision of the computation, e.g. the statement

```python
1 == ((0.3 * 3) + 0.1)
```

evaluates to False in Python 3.8.5. To overcome this issue, PyBryt supports tolerances when comparing numerical values. Both relative and absolute tolerances are supported in a similar fashion to NumPy’s isclose function [19].

Similar problems may be encountered when working with strings where the instructor may be invariant to using lower-case vs upper-case letters. PyBryt supports invariants by specifying them when declaring a value. Therefore, if the RI
def count_words(text: str) -> int:
    fragments = text.split()
    length = len(fragments)
    return length

Listing 9: Count words implementation

def count_words(text: str) -> int:
    ...  # remaining code

Listing 8: Count words assignment without scaffolding.

A student may implement the `count_words` function by splitting the string using built-in methods and then count the number of items in the resulting list as demonstrated in Listing 9.

To create a RI, the instructor may want to trace: (i) the list of words created by splitting the text (`fragments`), and (ii) the count of words as a Python int (`length`). This RI is shown in Listing 10. The key thing to observe is the ease with which the instructor can create a RI. To check the student’s solution, it can be executed with different values as inputs as demonstrated in Listing 11.

3.2 Insertion Sort

In this example, we consider an assignment in which students are asked to implement insertion sort. With PyBryt, the instructor does not need to create a scaffolded version of the solution, instead, the student can be asked to implement a function: `insertion_sort` that takes a list of integers as its input and returns the list sorted in ascending order using insertion sort.

Listing 12 presents a possible correct answer to this assignment while Listing 13 presents an incorrect solution in which the student’s outer loop does not iterate to the end (`len(x)-1` instead of `len(x)`). The instructor would like to distinguish the correct solution from the incorrect one and provide formative feedback. A unit-test can be used to tell the correct implementation from the erroneous one as demonstrated in Listing 14. From this feedback, both the student and educator can only see if the final solution is correct. However, even if the solution is correct, it cannot determine whether insertion sort was used or not. Moreover, if the student did not implement the code correctly, the feedback does not provide information on how to improve the code. Therefore, this method provides little value both in the formative feedback it provides and when used to provide summative feedback.
def count_words(text: str) -> int:
    # Split the text by spaces
    fragments = text.split()
    pybryt.Value(fragments, failure_message="It may help to first split the string
into a list of words", success_message="Great, the string was converted into a
list of words")

    # Count the number of items in the result
    length = len(fragments)
    pybryt.Value(length,
                 success_message="Great, identified the number of words in the
string",
                 failure_message="Cannot find a count of the number of words")
    return length

Listing 10: Count words reference implementation

assert count_words("Waltz, bad nymph, for quick jigs vex.") == 7
assert count_words("The five boxing wizards -- \njump quickly!") == 6
assert count_words("Sphinx of black quartz, judge my vow.") == 7

Listing 11: Unit tests for count words assignment

In contrast to non-informative feedback provided by unit-tests, using PyBryt, the solution in Listing 12 can be
converted into a RI such that the feedback for the incorrect assignment Listing 13 would be more informative as demonstrated in Listing 15.

From PyBryt’s feedback, the student can see that although the inner loop is correct, the outer loop is wrong. Because of this, the algorithm does not perform sort as expected. This allows the student to see if their final solution is correct, and PyBryt points the student to the “outer loop” to correct the mistake. Similarly, the instructor does not have to go into the source code and find an error. Instead, the educator can resort to PyBryt’s feedback. As such, PyBryt gives much more detailed feedback emphasizing all the points student’s function does right and where it can be improved. This way, we can ensure the function performs insertion sort and not some other sorting algorithm. To demonstrate that consider a student that instead of implementing insertion-sort calls Python’s sorted function to sort the list as shown in Listing 8. This solution will pass the unit test since the list is sorted correctly. However, PyBryt can recognize that the ingredients of insertion-sort are missing while the result is correct as demonstrated in the feedback provided by PyBryt in Listing 16.

Therefore, PyBryt has the potential of saving time for instructors when marking the submissions of entire classes. On the other hand, the student can see all the steps that they did right, which helps with the learning experience and it is more encouraging thus enhancing the overall student experience and increasing student satisfaction. When implemented as a self-assessment tool, students can receive immediate encouraging and constructive feedback which may create benefits from gamification perspectives.

def insertion_sort(x: list[int]) -> list[int]:
    for i in range(1, len(x)):
        current = x[i]
        j = i - 1
        while j >=0 and current < x[j] :
            x[j+1] = x[j]
            j -= 1
        x[j+1] = current
    return x

Listing 12: A correct implementation of an insertion sort assignment
def insertion_sort(x: list[int]) -> list[int]:
    for i in range(1, len(x) - 1):  # MISTAKE: len(x)-1 instead of len(x)
        current = x[i]
        j = i - 1
        while j >=0 and current < x[j] :
            x[j+1] = x[j]
            j -= 1
        x[j+1] = current
    return x

Listing 13: An incorrect implementation of an insertion sort assignment, note that the outer loop iterates to 'len(x)-1' instead of 'len(x)'.

x = [55, 111, -33, 65, 1001, -362, 451]
assert insertion_sort(x) == list(sorted(x))

AssertionError Traceback (most recent call last)
<ipython-input-6-ad50795e9ab5> in <module>
1 x = [55, 111, -33, 65, 1001, -362, 451]
----> 2 assert insertion_sort(x) == list(sorted(x))

AssertionError:

Listing 14: Unit test feedback for the insertion sort assignment. Note that the feedback is non-constructive from student’s perspective

3.3 Plagiarism Detection

Some accounts suspect that at least 20% of students are involved in programming plagiarism, even at the top universities. Others suggest that the numbers might be even higher: in a survey of 138 students, 72.5% admitted to have plagiarized at least once in a programming assignment. As a result, many methods have been developed for detecting plagiarism. To illustrate that, we note that at the time of writing this paper, Google Scholar reports that 21000 papers are matching the query 'plagiarism programming assignments', 7500 of them were published since 2017, and 2650 since 2020. Scholars also reported that reducing plagiarism does improve students competency. For instance, Pawelczak reported that since the introduction of automatic plagiarism detection, students scores improved by 8.6%.

In an attempt to conceal plagiarism, students use many techniques ranging from changing variable names and adding blank lines to breaking functions and adding non-functional code. PyBryt can act as a plagiarism detection tool by automatically converting students’ submissions into R1’s and comparing other students submissions to it. Since

https://www.nytimes.com/2017/05/29/us/computer-science-cheating.html
in August 2021

https://scholar.google.com/
with `pybryt.check(PyBryt_reference(1, 1)):`

```
insertion_sort([55, 111, -33, 65, 1001, -362, 451])
```

REFERENCE: exercise-1_1
SATISFIED: False
MESSAGES:
- ERROR: Hmmm... It looks like the algorithm does not perform insertion sort.
- ERROR: Please make sure the outer loop iterates from the second element to the last.
- ERROR: Please make sure the inner loop iterates towards left.
- SUCCESS: Amazing! The function returns the correct solution.

Listing 16: PyBryt feedback for the insertion-sort assignment when a student implement a sorting algorithm which is not insertion-sort

PyBryt compares only the values of variables, it is resilient to obfuscation techniques such as variable renaming, the addition of redundant code, changes to the order of computation that does not result in functional changes and more.

To use PyBryt as a plagiarism detection mechanism we note that if a solution was plagiarized then both the original version of the code and the copied version of the code will share most intermediate values. This is true even if variable names where changed or function names were changed. Moreover, even if a sophisticated student changed the order of calls to functions in places where it would not have consequences on the results then PyBryt can detect the plagiarism. One potential problem is that some intermediate values are very common and are not a reliable signal for plagiarism. To circumvent this problem, we propose using a method akin to TF-IDF to score the ‘suspiciousness’ of finding a value matching in the code submitted by separate students. If a value is common in the solutions of many students, then this value is not a significant marker for plagiarism.

4 Summary and Conclusions

Impagliazzo and Pear [15] argued that learning should include both knowledge, skill, and disposition. More precisely, Corbett and Anderson [6] found that in learning programming, students benefit from explicit and in-context guidance for coding assignments, compared to other means of feedback. This is especially important in light of the big gap in the faculty-to-teacher ratio in different colleges: Ertek et al. [10] presented in their study a comprehensive benchmark of the top 100 U.S. universities and found that the faculty-to-student ratio is a significant competitiveness factor.

In this work, we presented PyBryt, which uses a new approach to providing feedback to students on their coding assignments. PyBryt compares students solutions to RI’s by checking for the presence or absence of critical values, as marked by the instructors. Thus PyBryt is easy to use yet does not require the teacher to provide scaffolded exercises.

A side benefit of our approach is the ability to detect plagiarism. Plagiarism was identified as a concern in programming education since the 70’s [28, 20], and continues to worry educators today [13]. As PyBryt follows memory evolution during run time, it can detect similarity between programs - even if the source code is different, e.g. different variable names or redundant code.

This paper presents PyBryt and the way it is implemented. We are currently working on evaluating PyBryt in different pedagogical settings to measure its contribution to the learning process of students and the effort required from educators to use it. PyBryt can be extended in many ways. In its current form PyBryt supports only programs in python while it can be extended to support many other programming languages. Furthermore, it can support cross-lingual assessments since the intermediate results computed are invariant to the programming language used in many cases.

Moreover, PyBryt may be used beyond the classroom for tasks such as code-refactoring. Code-refactoring is a challenging task even for seasoned engineers. PyBryt can support this process by converting the original code into a reference implementation and testing the refactored code against it. This allows the detection of errors in the new implementation even when the code’s internal and/or external structure does not match the original one.

References

[1] Wahab Ali. 2020. Online and remote learning in higher education institutes: A necessity in light of COVID-19 pandemic. *Higher education studies* 10, 3 (2020), 16–25.
[2] Sahil Bhatia and Rishabh Singh. 2016. Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks. *arXiv:1603.06129 [cs]* (March 2016). [http://arxiv.org/abs/1603.06129](http://arxiv.org/abs/1603.06129)

[3] Douglas S Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas L Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, et al. 2019. nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. *The Journal of Open Source Education* 2, 11 (2019).

[4] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, Robert Endre Tarjan, et al. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461.

[5] Stefania Bocconi, Augusto Chiocchicielo, Giuliana Dettori, Anusca Ferrari, Katja Engelhardt, Panagiotis Kampylis, and Yves Punie. 2016. Developing computational thinking in compulsory education. *European Commission, IRC Science for Policy Report* 68 (2016).

[6] Albert T. Corbett and John R. Anderson. 2001. Locus of Feedback Control in Computer-Based Tutoring: Impact on Learning Rate, Achievement and Attitudes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) (**CHI ‘01**). Association for Computing Machinery, New York, NY, USA, 245–252. [https://doi.org/10.1145/365024.365111](https://doi.org/10.1145/365024.365111)

[7] Fábio Rezende de Souza, Francisco de Assis Zampirolli, and Guiou Kobayashi. 2019. Convolutional Neural Network Applied to Code Assignment Grading. In *CSEDU (1)*. 62–69.

[8] Stephen H Edwards and Manuel A Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*. 328–328.

[9] Bobbie Lynn Eicher and David Joyner. 2021. Components of Assessments and Grading at Scale. In *Proceedings of the Eighth ACM Conference on Conference at Scale*. 303–306.

[10] Gurdal Ertek, Bengi Tokdil, Ibrahim Güneydin, and Aytaç Gogus. 2014. Benchmarking Competitiveness of Top 100 U.S. Universities. In *Handbook of research on strategic performance management and measurement using data envelopment analysis*, Ibrahim H. Osman, Abdel Latef Anouze, and Ali Emrouneuzjad (Eds.). Business Science Reference, Hershey, PA, Chapter 10, 407–425.

[11] George E Forsythe and Niklaus Wirth. 1965. Automatic grading programs. *Commun. ACM* 8, 5 (1965), 275–278.

[12] Georgiana Haldeman, Andrew Tjang, Monica Babes-Vroman, Stephen Bartos, Jay Shah, Danielle Yucht, and Thu D Nguyen. 2018. Providing meaningful feedback for autograding of programming assignments. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 278–283.

[13] Charles A R Hoare. 1961. Algorithm 65: find. *Commun. ACM* 4, 7 (1961), 321–322.

[14] Jack Hollingsworth. 1960. Automatic graders for programming classes. *Commun. ACM* 3, 10 (1960), 528–529.

[15] John Impagliazzo and Arnold Neville Pears. 2018. The CC2020 project — computing curricula guidelines for the 2020s. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. 2021–2024. [https://doi.org/10.1109/EDUCON.2018.8363484](https://doi.org/10.1109/EDUCON.2018.8363484)

[16] Hamza Manzoor, Amit Naik, Clifford A Shaffer, Chris North, and Stephen H Edwards. 2020. Auto-Grading Jupyter Notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 1139–1144.

[17] Ramez Nabil, Nour Eldeen Mohamed, Ahmed Mahdy, Khaled Nader, Shereen Essam, and Essam Elwi. 2021. EvalSeer: An Intelligent Gamified System for Programming Assignments Assessment. In *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUC)**. IEEE, 235–242.

[18] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–37.

[19] Travis E Oliphant. 2006. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA.

[20] Karl J Ottenstein. 1976. An algorithmic approach to the detection and prevention of plagiarism. *ACM Sigcse Bulletin* 8, 4 (1976), 30–41.

[21] Dieter Pawelczak. 2018. Benefits and drawbacks of source code plagiarism detection in engineering education. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1048–1056.

[22] Juraj Petrík, Daniela Chudá, and Branislav Steinmüller. 2017. Source code plagiarism detection: The Unix way. In *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 000467–000472.
[23] Avneesh Sarwate, Creston Brunch, Jason Freeman, and Sebastian Siva. 2018. Grading at scale in earsketch. In Proceedings of the Fifth Annual ACM Conference on Learning at Scale. 1–4.

[24] Jan Skalka, Martin Drlik, Lubomir Benko, Jozef Kapusta, Juan Carlos Rodríguez del Pino, Eugenia Smyrnova-Trybulska, Anna Stolinska, Peter Svec, and Pavel Turcinek. 2021. Conceptual Framework for Programming Skills Development Based on Microlearning and Automated Source Code Evaluation in Virtual Learning Environment. Sustainability 13, 6 (2021). https://doi.org/10.3390/su13063293

[25] Dejan Sraka and Branko Kaucic. 2009. Source code plagiarism. In Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces. IEEE, 461–466.

[26] Michael Striewe and Michael Goedicke. 2014. A review of static analysis approaches for programming exercises. In International Computer Assisted Assessment Conference. Springer, 100–113.

[27] Maddalena Taras. 2005. Assessment–summative and formative–some theoretical reflections. British journal of educational studies 53, 4 (2005), 466–478.

[28] J Traxler. 1970. Plagiarism in programming: A review and discussion of the factors. WIT Transactions on Information and Communication Technologies 12 (1970).

[29] Jeannette M Wing. 2006. Computational thinking. Commun. ACM 49, 3 (2006), 33–35.

[30] Naomi E Winstone and David Boud. 2020. The need to disentangle assessment and feedback in higher education. Studies in higher education (2020), 1–12.

[31] Susilo Veri Yulianto and Inggriani Liem. 2014. Automatic grader for programming assignment using source code analyzer. In 2014 International Conference on Data and Software Engineering (ICODSE). IEEE, 1–4.