[1] **Srujana Inturi ***

[2] **M. Swamy Das**

Review of Automatic Computer Program Evaluation and Assessment Tools and Methods

Abstract: **-** Understanding the computer language is more important in today's world. All Computer Science students must have practical and proficient programming skills, which can be obtained through intensive exercise practices. Due to the regular rise in the number in a class, the evaluation of programming exercises imposes a heavy toll on the teacher or instructor, mainly if it must be performed manually. Manual grading for programming assignments might be time-consuming and error-inclined. Already available tools generate remarks with failing test instances. This research includes a thorough literature on the evolution of the recent (2004–2022) development of automatic programming assignment grading systems. From both a pedagogical and a technical viewpoint, the primary aspects supported by the tools and their diverse techniques were examined. In conclusion, several new systems are being built while also acknowledging the underlying causes of this situation. Building open-source systems and collaborating on their expansion is recommended as one solution. This paper concludes with suggestions for future research paths and possible enhancements to automatic code evaluation.

Keywords: Computer Science, Automated programming assessment, Grading tools, Education, Learning programming.

I. INTRODUCTION

Education in Computer Science is among the most popular academic fields worldwide. If you're passionate about computer hardware and software, you probably already know that earning a Bachelor's or Master's degree in CSE/IT can help you land a lucrative job. Most industries in the current digital era rely on data and software solutions. Everything is impacted by computer science and IT, including Scientific research, the advancement of healthcare, transportation, banking, communications, you name it. Now, even everyday items like refrigerators, microwaves, and door locks are wired to our Wi-Fi networks and personal assistants.

With a degree in Computer Science, you can gain the knowledge and abilities necessary to solve problems and create the next wave of devices or software that will enhance the lives of millions of people. Even people who do not work in the information and communication technologies field or do not aspire to work in that field should have some programming knowledge. Teaching a programming subject is more complicated than any theory subject because it involves a distinct procedure to assess and evaluate student-written code. In any educational system, assessment and evaluation is an essential component as it helps assess and evaluate the student's understanding, get feedback, and finally grade a student. The current manual system is time-consuming and tedious to evaluate the student's written code. However, teachers are free to assign grades however they see proper. On the other hand, there are exams when the questions and the answers could be slightly unclear and open-ended while still being accurate.

To reduce the teacher's time and maintain uniformity in the assessment and evaluation, several automatic tools have been built for the last few decades [1]. However, the pandemic has given rise to automatic grading tools at universities that offer traditional or distance learning [2,3]. Technological advances and communication systems have made education more accessible through Information and Communication Systems (ICT). The utilization of automatic grading tools has accelerated during the COVID-19 pandemic.

The benefits of Automatic Computer Program Evaluation are;

- It saves time and effort for the graders, cuts down on the number of issues presented to a student, and eliminates the need to restrict the number of applicants being evaluated based on the number of available graders.

[1] Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Gandipet-75, Hyderabad, India. isrujana_cse@cbit.ac.in

[2] Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Gandipet-75, Hyderabad, India. msdas_cse@cbit.ac.in

* Corresponding Author Email: isrujana_cse@cbit.ac.in

- It can deliver feedback in real-time. Students would not need to wait for faculty members to obtain information about the quality of their program to better themselves and their chances of being accepted.
- It can deliver feedback in real-time. Students would not need to wait for faculty members to obtain information about the quality of their program to better themselves and their chances of being accepted.

The remaining sections of this work are organized into the six major categories discovered in the reviewed review and survey publications. Sections 3-4 discuss the six issues and provide details on the findings from more specialized research publications that address them. The review offered in this work includes answers to the three research topics raised, which are then covered in Section 5. It also offers guidance for further research and advancement in this area. The final section summarizes the article and future directions in the related area.

II. RELATED WORK

From the literature review, a lot of works and automatic tools that can be used in education are seen. This paper covers various topics related to methods and tools for automatically evaluating codes published between 2004 and 2022. The analysis of these studies allowed us to identify six critical issues with the growth of automated code assessment systems

Table 1. General Classification of Methods

Assessment Type	Manual Assessment Automatic Assessment Semi-automatic Assessment
Method	Static Method Dynamic Method Hybrid Method Artificial Intelligence
Representation	Text-Based Token Based Abstract Syntax Tree-Based Program Dependency Graph

2.1. Coding / Program aspects

To minimize instructors' workload while assisting students to learn, automated assessment methods were first developed in 1960 [4]. With automatic tools, it is possible to spot undesirable code errors by automatically evaluating them. The review and research studies demonstrate that interest is focused on three functional components like different types of errors: syntax, runtime, and logical errors [5,6&7]. Non-functional components are another focus of various assessments of code accuracy [8], style-related issues [5,7,9], metrics-based code quality [5,7,10 &11], efficiency [5,11], and GUI [9]. Plagiarism is another issue that can be resolved through similarity analysis [7]. Most reviews concentrate on programming rather than other computer science-related abilities. However, several of them make mention of methods for evaluating students' testing abilities [5,9,11].

When it comes to the techniques and methods utilized for automated code assessment [5,10-14], they are classified as static and dynamic. On the techniques employed in software engineering, particularly in the disciplines of software [6,20] and software quality [6], more sophisticated techniques are used. Good test data are crucial when working with test-based evaluations, especially if created automatically [16]. The depth of the analysis is an essential piece of information. For instance, they may be completed on a tree, graph, or bytecode as an intermediate form [17]. Finally, another set of approaches relies on their being a single, correct solution that students may use as a reference for their work [6].

2.2. Feedback

An essential part of autonomous assessment technologies in education is the capacity to retrieve a measure that evaluates assigned programs to respond to something like the teaching objectives [5]. To explain this metric to both teachers and students, feedback received from the evaluations is necessary. Quality feedback, particularly constructive input, is essential for technology use in educational settings. More precisely, effective feedback should outline for students how to address any outstanding issues and move forward with their work [18]. The level of feedback given to students should be improved due to developed mechanisms. Automating feedback generation can improve learning because feedback is a constant motivator [13].

2.3. Supported Tools

The tools created to facilitate automatic code assessment have been the subject of several assessments [13,14,19], as shown in Figure 4. They primarily draw attention to the vast array of unique qualities that could exist on the one hand and the difficulties in developing them on the other. Assistance in evaluating, grading, and managing programming assignments is a commonly cited capability. Fundamental analyses with short results depend on graders were the emphasis of early systems; tool-oriented systems designed on test engines and other tools like style checkers came next; and ultimately, web-oriented tools rounded out the range [5,7]. These three prior generations of tools are reviewed. Such platforms must be developed while considering various tangible factors, including architecture selections, programming languages, applied technologies, etc [19]. It is also crucial to choose the type of tool, which is typically a library, standalone program, or plugin for learning management systems [15,17,21].

2.4. Integration with Learning System

Due to the significant increase in students enrolling in programming courses in traditional and online learning environments, the need for automatic evaluation tools to assist teachers is mentioned in every review [10,13,18]. Regarding their application in education, these systems are learning engines that improve students' motivation, development, self-evaluation, and computer science-oriented skills. These platforms are intended to evaluate the achievement of teachers' learning objectives [20]. They may focus on teachers who must assign grades or students who need to enhance their abilities [13,21]. Formative or summative testing can be performed using automated code assessment tools in a fully or semi-automated environment or even manually [6,13,20,21]. Formative assessment offers the chance to provide learners with challenging activities to train and transition to continuous evaluation for a course [18,20]. The resubmission policy is under consideration in both cases, with proposals for limitless submissions for formative reviews. They are compatible with various educational methodologies, including competency-based, MOOCs, and distance learning [9,18].

2.5. Efficiency

Finally, more study and research need to be done, especially on the level of the assessments that have been produced. However, all the reviews agreed that the assessment and accompanying feedback needed to help the learning process [5].

III. Methods Or Approaches For Assessment

As noted earlier, a systematic literature analysis was carried out and found thirty different assessment tools for grading programs. The tools are listed in Table II. Further, three tool classification methods are suggested, as indicated in Table 1.

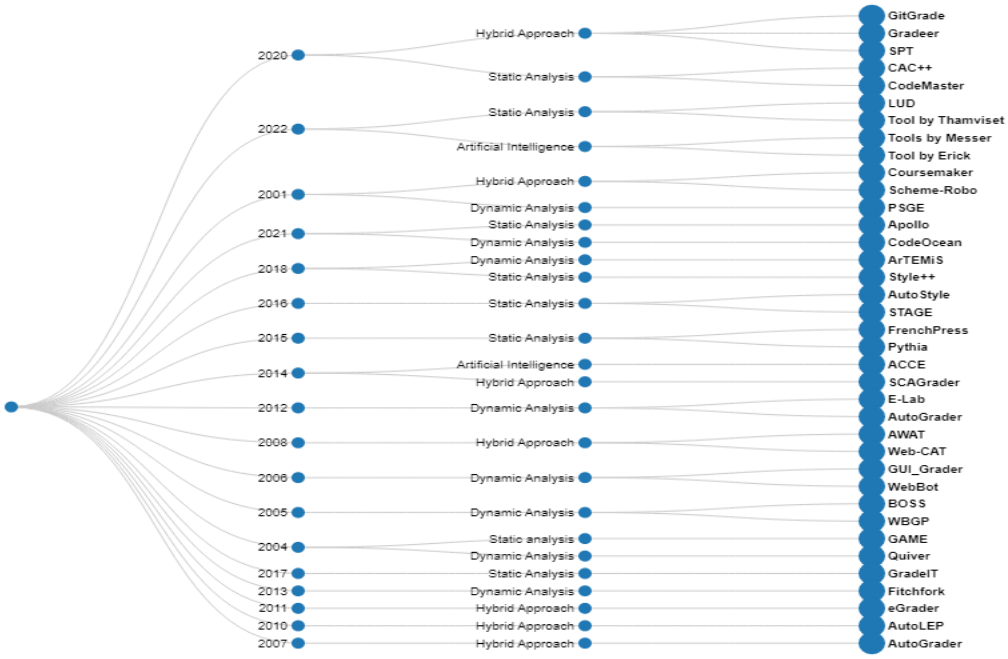

Figure 1. The year-by-year breakdown of the number of tools that employ a given approach.

3.1. Assessment Type

There are three assessment techniques based on the assessment process conducted;

1. Manual Assessment: The instructor manually assesses the programming assignment with the tool's aid [5]. When the student code is submitted, it is compiled and executed by the tool. This is done in the local student's PC under the instructor's guidance. A log with all the details like compilation, source program, execution, and documentation is provided to a section of the professor's server after the program has completed its execution.

Table 2: Advantages and disadvantages of different assessment methods

Method	Advantages	Disadvantages
Manual Assessment	Programs written by students are not required to stick to a precise input-output format. There is complete assistance for the instructor's careful inspection.	There needs to be a systematic approach to the assessment. The instructor's time and effort are not significantly cut down.
Automatic Assessment	There is a systematic approach where students can be assessed based on syntax, semantics and program quality. The instructor's time and effort can be strongly reduced.	Students must follow the input and output format. There is no assistance for the instructor's inspection.
Semi-Automatic Assessment	The assessment is partially systematic. Instructor involvement is partially supported.	The instructor's time and effort can be partially reduced.

2. Automatic Assessment: The tool evaluates a programming assignment automatically. However, before the assessment procedure, the instructor must specify the criteria for evaluating the computer code [5,21]. Therefore, the evaluation criteria should be established before the evaluation procedure. For example, when utilizing eGrader [23], the instructor must give grading rubrics explaining how the assessment is done. EGrader evaluates each student's program using the grading schema's evaluation criteria.

3. Semi-Automatic Assessment: This tool administers the evaluations automatically, but the instructor must manually review the source code [5]. One key task of the tools listed by [24] is to automatically verify whether students' programmes provide the correct output given a set of input data. When one of the other programmes provides a different result than expected, the tool prompts the teacher to physically compare the discrepancies between the programme outcome and the intended output.

3.2. Methods

Automated Program evaluation systems are categorized according to the program evaluation approach. The distinction is made between static and dynamic analysis.

a. Static Analysis: Analyzing a program statically involves reviewing its source code without executing it. This technique examines the program's structure and content to acquire the required data, as shown in Figure 2

- Programming style analysis: This approach measures a program's quality by considering its readability. The requirements for a highly legible program are to understand variable names, the usage of constants, appropriate line spacing, etc., [25].
- Semantic error detection: A semantic mistake is found when a statement is syntactically valid yet results in an error when the program is executed.

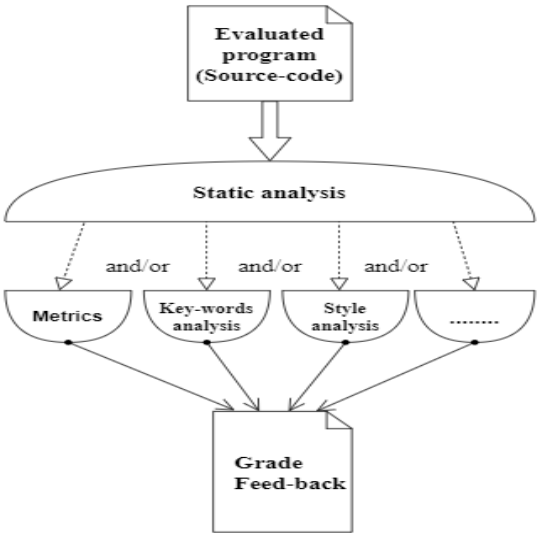

Figure 2. Static Analysis Method

Beginner programmers commonly make semantic errors like division by zero and never-ending loops. Some semantic mistakes might cause significant issues, such as system failure. Because of this, even some systems that are intended to evaluate programs dynamically use this strategy to prevent mistakes of this kind [26]

- Software metric analysis: The complexity and dependability of source code can be determined by assessing specific program features. Metrics comprise the dimensions of size, code quality and time complexity that may be assessed over the source program, as well as the frequency of comments, the average size of statements given by several operands and operators, the statement number between the beginning and end etc., [27,28,29].
- Keyword Analysis: The challenge with that approach is locating the evaluator-defined important keywords that must be located in the evaluated assignment's source code [30,31].
- Structural and non-structural similarity analysis: It involves comparing the assessed program to a collection of program solutions offered by the teacher to find similarities as part of the static analysis technique. The grade for the student's response will be the highest score from the comparison [27,31-34].

b. Dynamic Analysis: Dynamic methods collect techniques that require the execution of code. In most cases, they are done by analyzing the output generated by code execution and comparing the findings to those of a reference solution, as shown in Figure 3.

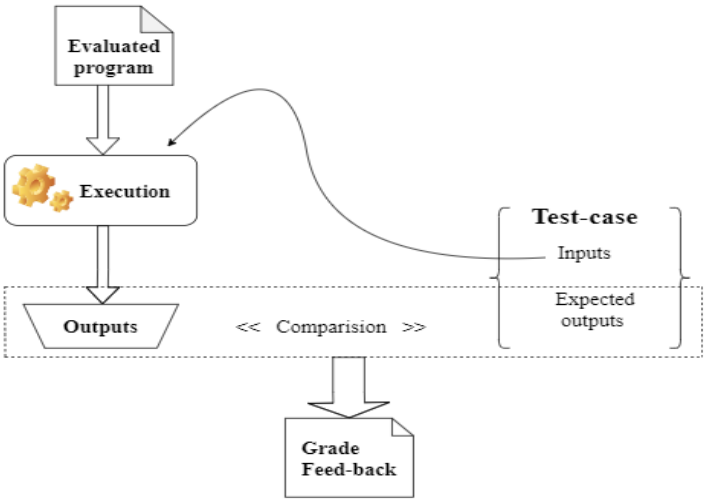

Figure 3. Dynamic Analysis Method

Different approaches to evaluating a program are distinguished within dynamic analysis:

- The black-boxing method views the software as one entity. Consequently, the code output is examined, and the only possible verdict is correct or incorrect [27,35,36].

- In grey-boxing, each programming function is evaluated independently. Therefore, the grade of the executed code is the aggregation of the partial grades of each module based on weighted percentages. It is helpful because a source program may be marked even with an invalid function [37-39].

c. Hybrid Approach: Combining static and dynamic analysis, hybrid approaches enable a code evaluation for a particular aspect. Typically, similar techniques are employed when assessing programs whose output is code. Hull et al. [40] showed the execution of provided algorithms produces SVG charts. They are then examined using a static technique to extract the data necessary to assess the contribution. In a study by Sztipanovits et al. [41], the authors suggest evaluating web apps by ensuring they have the desired component. Static analyses are used to investigate the HTML code of created web pages. Combining dynamic and static analysis, hybrid techniques enable grading code submissions with syntax faults or provide no output [34]. This is particularly relevant for courses designed for beginning programmers.

d. Artificial Intelligence: Moreover, artificial intelligence processes and analyses a computer program's many aspects. In a few studies [42,43]. Naive Bayes machine learning algorithms are used to evaluate source programs based on a grammar of features collecting principal aspects that human instructors consider when evaluating a piece of code. In a study by Simanjuntak [26], deep learning is also used to check if ER diagrams are close to the anticipated answer. Using a convolutional neural network architecture, Gradjanin et al. [44] evaluated the resilience of a web or html page by finding similar screenshots of the website before and after adding new information. The multi-layer neural network's output is combined with other measures to give a perfect score. Clustering techniques automate style grading [45] by finding frequent errors. The author hypothesizes there may be a finite number of solutions to a given question.

3.3. Representation

Several studies have been performed on automatic grading systems. Although there are numerous automatic grading procedures, their operations are identical. The distinction between them is their data representation.

- Text-based: This is the most accessible and efficient of all the other techniques. In this technique, programs are viewed as a string. Comparing two strings typically involves calculating the appropriate edit distance [46]. Before comparing code fragments, the raw source program is used directly or with minimum processing or standardization implemented.
- Token-based: This method uses a lexical analyzer to transform source code into tokens. Typically, token-based methods are more resistant to program modifications such as formatting and spacing. Substring or suffix tree matching algorithms are used to compare sequences of tokens.
- Abstract Syntax Tree Based: This method uses parsers to build a syntax representation of the source program in the structure of an Abstract Syntax Tree (AST) or parse tree. Then, sub-tree matching methods are utilized to determine the degree of similarity between programs.
- Program Dependency Graph: In this method, the source program is converted into an intermediate representation Program Dependency Graph (PDG), representing the control and data similarities between programs. Using isomorphic sub-graph matching methods, comparable functional programs are compared. This method is ideal for locating the reordered, newly added or removed statements, complicated statements, and noncontagious programs.

IV. TOOLS REVIEW

The primary motivation for creating or using these tools is to help students, especially those just starting in the field of computer science, sharpen their coding skills. The abilities will be enhanced by completing several programming activities. Students may work on the challenges as soon as they get helpful comments. It would help them see their errors and enhance their abilities. In addition, students get a tangible advantage in the form of a grade that is not influenced by the personal preferences of the faculty [47].

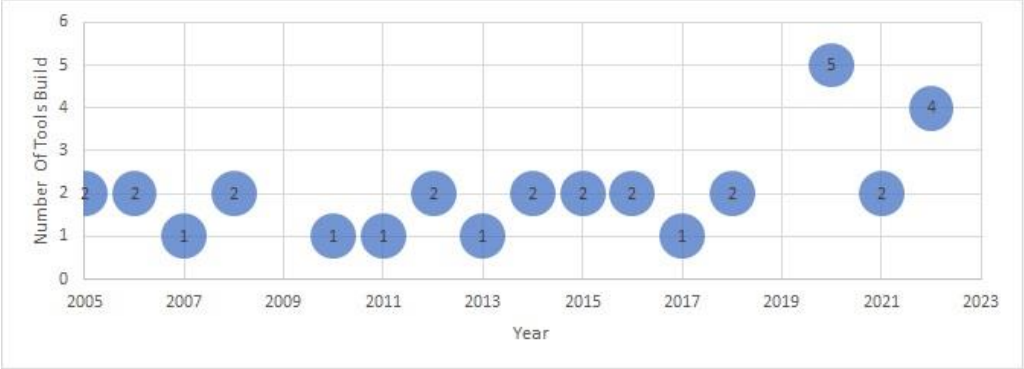

Figure 4. Number of tools by publication year

Manual grading is impractical due to the number of students and their programming assignments in a regular engineering course. Another objective is maximizing their available time to keep the academic staff manageable. The time saved can be utilized for other productive tasks, such as course planning and development, or to provide more individualized attention to students.

4.1. Analysed Key Features

Four key characteristics of automated code evaluation systems are noted in [33]: Programming languages supported and used, deployment architecture, submission receipt and storage, Metrics for evaluating performance, support for automatic or semi-automatic assessment, and feedback provision.

- Support for Programming languages - It is essential when a quick implementation is sought. It could decide if a tool is practical or not.
- Architecture of deployment- It indicates the condition of the hardware with which the tool interacts. Determining if the current environment supports the introduction of technology is beneficial. It will identify the necessary resources and help calculate the implementation cost in the worst-case situation.
- Work mode- Determines whether the tool can operate alone, for precise implementations, or as a plugin when used with another system, such as an LMS.
- Grading Metric- It demonstrates how the tool can determine a grade. It evaluates which metrics are considered during the grading procedure, including how the grade is calculated.

Table 3 summarises the tools examined in this paper's research. It includes the tool's name, references, year, methodologies and techniques, available programming languages, and assessment type for each.

Table 3.: Several characteristics of automated code assessment tools

Tool/Reference	Year	Methods and approach	Assessment Type	Programming Languages
LUD	2022	Static Analysis	Automatic	Multi-language
Tool by Thamviset [48]	2022	Static Method	Automatic	C++
Tools by Messer [49]				
Tool by Gaona [50]	2022	Artificial	Automatic	Multi-language
Apollo [51]	2022	Intelligence	Automatic	Multi-language
CodeOcean [52]	2021	Artificial	Automatic	Processing
CAC++ [53]	2021	Intelligence	Automatic	Agnostic
CodeMaster [54]	2020	Static Method	Automatic	C, C++
GitGrade [55]	2020	Dynamic Method	Automatic	App Inventor and
Gradeer [56]	2020 2021	Static Method	Automatic	Snap Projects
SPT [57]	2020	Static Method	Semi-Automatic	Agnostic
Artemis [58]	2018	Hybrid Approach	Automatic	Java
Style++ [33]	2018	Hybrid Approach	Manual Assessment	JAVA
		Hybrid Approach	Semi-Automatic	

GradeIT [59]	2017	Dynamic Method		All programming Languages
AutoStyle [60]	2015	Static Method	Automatic	C++
STAGE [61]	2008		Semi-Automatic	C
FrenchPress [32]	2015	Static Method	Automatic	C++
Pythia [67]	2015 2014	Static Method	Semi-Automatic	JAVA
ACCE [45]	2014	Static Method	Automatic	JAVA
SCAGrader [29]	2013	Static Method	Automatic	Agnostic
Pitchfork [18]	2012	Static Method	Automatic	Python
E-Lab [68]	2007	Artificial Intelligence	Automatic	C, JAVA and PHP
AutoGrader [35]	2011	Hybrid Approach	Automatic	Agnostic
eGrader [23]	2011	Dynamic Method	Automatic	C, C++ and JAVA
AutoLEP [34]	2008	Dynamic Method	Automatic	JAVA
AWAT [41]	2006	Dynamic Method	Automatic	JAVA
Web-CAT [2]	2006	Hybrid Approach	Semi-Automatic	Agnostic
GUIGrader [38]	2006	Hybrid Approach	Automatic	Agnostic
WebBot [62]	2005	Hybrid Approach	Automatic	Agnostic
BOSS [63]	2005	Hybrid Approach	Semi-Automatic	JAVA
WBGP [64]	2004	Dynamic Method	Semi-Automatic	Multi-language
GAME [65]	2004	Dynamic Method	Automatic	Mainly JAVA
Quiver	2001	Dynamic Method	Automatic	Agnostic
Coursemaker		Dynamic Method		JAVA, C and C++
PSGE	2001	Static Method	Automatic	C++, JAVA and MIPS
Scheme-Robo		Dynamic Method	Semi-Automatic	JAVA and C++
		Hybrid Approach	Semi-Automatic	Agnostic
		Dynamic Method		Scheme
		Hybrid Approach		

Most created tools are web-based platforms that analyze given code, including syntax checking, plagiarism detection, test case execution, and code quality evaluation. CodeMaster evaluates block-based programming for mobile applications by calculating their computational understanding of dimension-based complexity [66]. The j-assess is a client-side web application that evaluates JavaScript scripts via various industry-standard techniques [30]. Specific to Java programs, ProgEdu focuses on violations of code style [33]. TCL/TK, a web-based application, is used by WBGP, allowing structured comments to be added to uploaded source code [64]. An application and a web-based client are found in the BOSS system. It allows for both summative and formative evaluations since students can take exams before the last submission, and instructors can create their exams for final grading [63]. The E-Lab software program builds test cases automatically from a teacher-provided reference solution and then runs them against student-submitted code.

Several existing technologies consist of a client application. The autoStyle auto-grading system is a graphical user interface (GUI) tool that instructors and students may use to evaluate program style. It gives three types of hints approach, syntax, and program skeletons [60]. The tool Scheme-robot is much more straightforward, as students submit their program by sending a message to a specified email address. The recipient receives an output with a clone of their sub-mission and a list of points with comments [31]. The GAME application is a graphical user interface (GUI) program that marks Java, C, and C++ projects according to an instructor-defined marking schema and technique [65]. Apollo is a tool for detecting whether or not students have mastered learning objectives by

extracting features from submitted codes for assignments that have been separately constructed for each learning objective. Metrics computed with PMD [51] provide the evidence. AutoGrader executes input codes symbolically and compares them to standard output. This method reduces the need to produce or design test cases while keeping the capacity to give students counterexamples when their submissions are erroneous. Web CAT is recognized for measuring how well students have evaluated a code and is utilized in various courses developed to motivate students to submit assessments [2]. The Gradeer tool is a command-line interface (CLI) application that performs a function in reverse, in the sense that it is designed to assist in human evaluation [56].

V. FUTURE DIRECTIONS

The trends and advancements in the automatic evaluation of programming assignments have yet to be regularly collected and discussed for the last five years. Therefore, the following research concerns are discussed in this section:

- Programming Languages: Most automatic grading systems are designed exclusively for Agnostic or Java. This is consistent with Java's standing as an extensively used programming language for beginners. C, C++, Java, and Python are popular languages the system supports. Support for multiple languages (Agnostic) was utilized extensively on most competitive platforms. Some of the tools are language-independent. Once the system has been created to create and execute solutions in that language, any language run on the same environment can be automatically evaluated. This is especially true if the evaluation is based on comparing outputs. There is currently a high demand for Language-agnostic programming or scripting (also known as language-independent, language-neutral, or cross-language), which is a software development paradigm in which a specific language is chosen based on its suitability for a specific task (considering all factors such as ecosystem, performance, developer skill-sets, and so on), rather than solely on the development team's skill-set. It is observed that the latest systems support all programming languages.
- Learning Management Systems (LMS): Developing existing learning management systems (LMS) like Moodle to better accommodate the unique requirements of Computer Science education is attracting significant interest. One justification for LMS integration with automatic grading systems is to use all course management elements. A learning management system (LMS) that contains many courses (other than programming) has more users, which makes it an attractive target for hackers. When conducted in such an atmosphere, the execution of malicious programs is always a significant risk. Therefore, safeguarding automatic grading systems is essential. LMSS must be interconnected. Although there are a more significant number of positives than negatives when it comes to difficulties, this strategy, and the fact that there is a growing demand for it, brings automatically graded exercises into learning management systems.
- The following AA LMS extensions have been identified: CTPracTo add VHDL and Matlab questions to LMS Moodle, Automatic Grader [49] to grade Java assignments for students alterations in Sakai and Auto Grader to allow JAVA assign- Cascade6, Webwork and JAG [2,13] automated evaluation of JAVA questions in WeBWorK7, SISA- EMU) will assign Assembler programming tasks. Through Moodle, then VERKKOKE [59] to deliver socket coding and routing within any LMS
- GradingMetrics: The absence of a standard grading scheme is still a severe issue. First, every institution and even Every instructor has criteria for grading assignments. A computer program cannot determine a comment's originality or good judgment. Despite these facts and considering the significance of creating a collection of metrics, a definition of metrics is shown in Figure 2. Every metric could have a corresponding evaluation tool. A language interpreter for functionality, a language compiler for execution, and a test case-based program (JUnit in Java, for example) are used for compilation. For specific requirements, a specific program for style, design, complexity, and an external program (Checkstyle for style in Java, for example).
- Interoperability: Institutions typically design a platform to suit their particular needs related to the programming languages taught to students or course-specific requirements. A probable additional cause is the absence of standard models for grading assignments and the vast range of available assignments [62]. Nonetheless, several existing platforms rely significantly on industrial tools to do the studies [41], which may be a part of the answer toward greater collaboration and cooperation. In addition, three generations of automatic program grading systems are discussed in the review published in [5]. A potential hypothetical fourth one may be tools delivered in cloud services, application programming interfaces (APIs), or a Learning Tools Interoperability (LTI) interface. This would simplify including them in learning management systems [64].

Table 4. Grading Metrics

	Unsatisfactory	Satisfactory	Good	Excellent	Weight	Score	
Specific Requirements		x			30%	70%	
Coding Standards			x		30%	85%	
Documentation				x	10%	100%	
Execution			x		20%	85%	
Efficiency	x				10%	0%	
	0%	70%	85%	100%		Grade:	74%

VI. Conclusion

This paper examined the most recent advancements in automatic evaluation tools for programming assignments. This was accomplished by methodically gathering relevant publications published between 2001 and 2022 to determine what has transpired in the field since the last literature review on the subject was undertaken. Based on the data gathered, the recommendations can be given, indicating that new automatic assessment systems could be deployed more broadly. First, authors describing novel systems offer additional examples and a more comprehensive explanation of the system's operation. Additionally, more attention should be paid to the security of the evaluation systems. Support using a suitable sandbox on an application based on existing security solutions. In the end, security must be supplied to facilitate system installation without sacrificing security. However, constructing a sandbox may be challenging. The first configuration of the security system should be independent of the instructors' skills. The lack of open-source systems may be one of the factors contributing to the continued Innovation of new tools that are likely to stay internal. It can be understood that people only want to upload complete work, making it more difficult for others to share creative ideas. As a result, existing technologies would be more readily adopted by others if they were open-sourced on a well-known online version control repository such as Google Code, SourceForge or GitHub. The developers of automatic evaluation systems need to make their software accessible to the public and release it under an open-source licence so that others may find it simpler to contribute to the project. This will help promote the use of existing systems and prevent the needless reinvention of the wheel.

References

[1] Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. Journal on Educational Resources in Computing (JERIC), 5(3), 4-es.

[2] Edwards, S. H., & Perez-Quinones, M. A. (2008, June). Web-CAT: automatically grading programming assignments. In Proceedings of the 13th annual Conference on Innovation and Technology in Computer Science Education (pp. 328-328).

[3] Matthíasdóttir, Á., & Arnalds, H. (2016, June). E-assessment: students' point of view. In Proceedings of the 17th International Conference on Computer Systems and Technologies 2016 (pp. 369-374).

[4] Hollingsworth, J. (1960). Automatic graders for programming classes. Communications of the ACM, 3(10), 528-529.

[5] Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. Computer science education, 15(2), 83-102.

[6] Aldriye, H., Alkhalaf, A., & Alkhalaf, M. (2019). Automated grading systems for programming assignments: A literature review. International Journal of Advanced Computer Science and Applications, 10(3).

[7] Rahman, K. A., & Nordin, M. J. (2007). A review of the static analysis approach in the automated programming assessment systems.

[8] Caiza, J. C., & Del Alamo, J. M. (2013). Programming assignments automatic grading: a review of tools and implementations. INTED2013 Proceedings, 5691-5700.

[9] Staubitz, T., Klement, H., Renz, J., Teusner, R., & Meinel, C. (2015, December). Towards practical programming exercises and automated assessment in Massive Open Online Courses. In 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) (pp. 23-30). IEEE.

[10] Lajis, A., Baharudin, S. A., Ab Kadir, D., Ralim, N. M., Nasir, H. M., & Aziz, N. A. (2018). A review of techniques in automatic programming assessment for practical skill test. Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 10(2-5), 109-113.

[11] Liang, Y., Liu, Q., Xu, J., & Wang, D. (2009, December). The recent development of automated programming assessment. In 2009 International Conference on Computational Intelligence and Software Engineering (pp. 1-5). IEEE.

[12] Arifi, S. M., Abdellah, I. N., Zahi, A., & Benabbou, R. (2015, November). Automatic program assessment using static and dynamic analysis. In 2015 Third World Conference on Complex Systems (WCCS) (pp. 1-6). IEEE.
[13] Gupta, S., & Gupta, A. (2017). E-Assessment Tools for Programming Languages: A Review. ICITKM, 65-70.
[14] Ismail, M. H., & Lakulu, M. M. (2021). A Critical Review on Recent Proposed Automated Programming Assessment Tool. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(3), 884-894.
[15] Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010, October). Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli calling an international conference on computing education research (pp. 86-93).
[16] Romli, R., Sulaiman, S., & Zamli, K. Z. (2010, June). Automatic programming assessment and test data generation: a review on its approaches. In 2010 International Symposium on Information Technology (Vol. 3, pp. 1186-1192). IEEE.
[17] Striewe, M., & Goedicke, M. (2014, June). A review of static analysis approaches for programming exercises. In International Computer Assisted Assessment Conference (pp. 100-113). Cham: Springer International Publishing.
[18] Pieterse, V. (2013). Automated Assessment of Programming Assignments. CSERC, 13, 4-5.
[19] Caiza, J. C., & Del Alamo, J. M. (2013). Programming assignments automatic grading: a review of tools and implementations. INTED2013 Proceedings, 5691-5700.
[20] Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010, October). Review of recent systems for automatic assessment of programming assignments. In Proceedings of the 10th Koli calling an international conference on computing education research (pp. 86-93).
[21] Souza, D. M., Felizardo, K. R., & Barbosa, E. F. (2016, April). A systematic literature review of assessment tools for programming assignments. In 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET) (pp. 147-156). IEEE.
[22] Striewe, M., & Goedicke, M. (2014, June). A review of static analysis approaches for programming exercises. In International Computer Assisted Assessment Conference (pp. 100-113). Cham: Springer International Publishing.
[23] AlShamsi, F., & Elnagar, A. (2011, April). An automated assessment and reporting tool for introductory Java programs. In 2011 International Conference on Innovations in Information Technology (pp. 324-329). IEEE.
[24] Jackson, D. (2000). A semi-automated approach to online assessment. ACM SIGCSE Bulletin, 32(3), 164-167.
[25] Hegarty-Kelly, E., & Mooney, D. A. (2021, January). Analysis of an automatic grading system within first-year computer science programming modules. In Proceedings of 5th Conference on Computing Education Practice (pp. 17-20).
[26] Simanjuntak, H. (2015, October). Proposed framework for automatic grading system of ER diagram. In 2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE) (pp. 141-146). IEEE.
[27] Karavirta, V., & Ihantola, P. (2010). Automatic assessment of javascript exercises. Proceedings of the 1st Educators' Day on Web Engineering Curricula, Vienna, Austria, 5-9.
[28] Pape, S., Flake, J., Beckmann, A., & Jürjens, J. (2016, May). STAGE: A software tool for automatic grading of testing exercises: Case study paper. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 491-500).
[29] Yulianto, S. V., & Liem, I. (2014, November). Automatic grader for programming assignments using source code analyzer. In 2014 International Conference on Data and Software Engineering (ICODSE) (pp. 1-4). IEEE.
[30] Lingling, M., Xiaojie, Q., Zhihong, Z., Gang, Z., & Ying, X. (2008, December). An assessment tool for assembly language programming. In 2008 International Conference on Computer Science and Software Engineering (Vol. 5, pp. 882-884). IEEE.
[31] Saikkonen, R., Malmi, L., & Korhonen, A. (2001, June). Fully automatic assessment of programming exercises. In Proceedings of the 6th annual Conference on Innovation and Technology in Computer Science Education (pp. 133-136).
[32] Blau, H., & Moss, J. E. B. (2015, June). French Press gives students automated feedback on Java program flaws. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education (pp. 15-20).
[33] Chen, H. M., Chen, W. H., & Lee, C. C. (2018). An Automated Assessment System for Analysis of Coding Convention Violations in Java Programming Assignments. J. Inf. Sci. Eng., 34(5), 1203-1221.
[34] Wang, T., Su, X., Ma, P., Wang, Y., & Wang, K. (2011). Ability-training-oriented automated assessment in an introductory programming course. Computers & Education, 56(1), 220-226.
[35] Helmick, M. T. (2007, June). Interface-based programming assignments and automatic grading of Java programs. In Proceedings of the 12th annual SIGCSE conference on Innovation and Technology in Computer Science education (pp. 63-67).
[36] Insa, D., & Silva, J. (2018). Automatic assessment of Java code. Computer Languages, Systems & Structures, 53, 59-72.
[37] Cheang, B., Kurnia, A., Lim, A., & Oon, W. C. (2003). On automated grading of programming assignments in an academic institution. Computers & Education, 41(2), 121-131.
[38] Feng, M. Y., & McAllister, A. (2006, October). A tool for automated GUI program grading. In Proceedings. Frontiers in Education. 36th Annual Conference (pp. 7-12). IEEE.
[39] Fu, X., Peltsverger, B., Qian, K., Tao, L., & Liu, J. (2008). APOGEE: automated project grading and instant feedback system for web-based computing. ACM SIGCSE Bulletin, 40(1), 77-81.

[40] Hull, M., Guerin, C., Chen, J., Routray, S., & Chau, D. H. (2021). Towards Automatic Grading of D3. Js Visualizations. arXiv preprint arXiv:2110.11227.

[41] Sztipanovits, M., Qian, K., & Fu, X. (2008, March). The automated web application testing (AWAT) system. In Proceedings of the 46th Annual Southeast Regional Conference on XX (pp. 88-93).

[42] Srikant, S., & Aggarwal, V. (2013, December). Automatic grading of computer programs: A machine learning approach. In 2013 12th International Conference on Machine Learning and Applications (Vol. 1, pp. 85-92). IEEE.

[43] Srikant, S., & Aggarwal, V. (2014, August). A system to grade computer programming skills using machine learning. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1887-1896).

[44] Gradjanin, E., Prazina, I., & Okanovic, V. (2021, September). Automatic Web Page Robustness Grading. In 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO) (pp. 177-180). IEEE.

[45] Rogers, S., Tang, S., & Canny, J. (2014, March). ACCE: automatic coding composition evaluator. In *Proceedings of the first ACM conference on Learning@ scale conference* (pp. 191-192).

[46] Johnson, J. H. (1994, September). Substring Matching for Clone Detection and Change Tracking. In ICSM (Vol. 94, pp. 120-126).

[47] Higgins, C. A., Gray, G., Symeonidis, P., & Tsintsifas, A. (2005). Automated assessment and experiences of teaching programming. Journal on Educational Resources in Computing (JERIC), 5(3), 5-es.

[48] Hahn, M. G., Navarro, S. M. B., & de-La-Fuente-Valentín, L. (2022, July). LUD: An Automatic Scoring and Feedback System for Programming Assignments. In 2022 International Conference on Advanced Learning Technologies (ICALT) (pp. 384-386). IEEE.

[49] Messer, M. (2022, July). Automated Grading and Feedback of Programming Assignments. In Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2 (pp. 638-639).

[50] Gaona, E. F., Camacho, C. E. P., Castro, W. M., Castro, J. C. M., Rodríguez, A. D. S., & Avila-Garcia, M. S. (2021, October). Automatic grading of programming assignments in Moodle. In 2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT) (pp. 161-167). IEEE.

[51] Rump, A., Fehnker, A., & Mader, A. (2021). Automated Assessment of Learning Objectives in Programming Assignments. In A. I. Cristea, & C. Troussas (Eds.), Intelligent Tutoring Systems: 17th International Conference, ITS 2021, Virtual Event, June 7–11, 2021, Proceedings (pp. 299-309). (Lecture Notes in Computer Science; Vol. 12677). Springer. https://doi.org/10.1007/978-3-030-80421-3_33.

[52] Serth, S., Staubitz, T., Teusner, R., & Meinel, C. (2021, March). CodeOcean and CodeHarbor: Auto-Grader and Code Repository. In SPLICE 2021 Workshop CS Education Infrastructure for All III: From Ideas to Practice. Virtual Event (p. 5).

[53] Delgado-Pérez, P., & Medina-Bulo, I. (2020). Customizable and scalable automated assessment of C/C++ programming assignments. Computer applications in engineering education, 28(6), 1449-1466.

[54] Solecki, I., Porto, J., Alves, N. D. C., Gresse von Wangenheim, C., Hauck, J., & Borgatto, A. F. (2020, February). Automated assessment of the visual design of Android apps developed with the app inventor. In Proceedings of the 51st ACM Technical Symposium on computer science education (pp. 51-57).

[55] Zhang, J. K., Lin, C. H., Hovik, M., & Bricker, L. J. (2020, March). GitGrade: A Scalable Platform Improving Grading Experiences. In SIGCSE (p. 1284).

[56] Clegg, B., Villa-Uriol, M. C., McMinn, P., & Fraser, G. (2021, May). Gradeer: an open-source modular hybrid grader. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 60-65). IEEE.

[57] Ardimento, P., Bernardi, M. L., & Cimitile, M. (2020, June). Towards automatic assessment of object-oriented programs. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings (pp. 276-277).

[58] Krusche, S., & Seitz, A. (2018, February). Artemis: An automatic assessment management system for interactive learning. In Proceedings of the 49th ACM Technical Symposium on Computer Science education (pp. 284-289).

[59] Parihar, S., Dadachanji, Z., Singh, P. K., Das, R., Karkare, A., & Bhattacharya, A. (2017, June). Automatic grading and feedback using program repair for introductory programming courses. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (pp. 92-97).

[60] Moghadam, J. B., Choudhury, R. R., Yin, H., & Fox, A. (2015, March). AutoStyle: Toward coding style feedback at scale. In Proceedings of the Second (2015) ACM Conference on Learning@ Scale (pp. 261-266).

[61] Kaila, E., Rajala, T., Laakso, M. J., & Salakoski, T. (2008, November). Automatic assessment of program visualization exercises. In Proceedings of the 8th International Conference on Computing Education Research (pp. 101-104).

[62] Colton, D., Fife, L., & Thompson, A. (2006). A web-based automatic program grader. Director, 07.

[63] Joy, M., Griffiths, N., & Boyatt, R. (2005). The boss's online submission and assessment system. Journal on Educational Resources in Computing (JERIC), 5(3), 2-es.

[64] Juedes, D. W. (2005, October). Web-based grading: Further experiences and student attitudes. In Proceedings Frontiers in Education 35th Annual Conference (pp. F4E-18). IEEE.

[65] Blumenstein, M., Green, S., Nguyen, A., & Muthukkumarasamy, V. (2004, April). GAME: A generic automated marking environment for programming assessment. In International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. (Vol. 1, pp. 212-216). IEEE.

[66] Von Wangenheim, C. G., Hauck, J. C., Demetrio, M. F., Pelle, R., da Cruz Alves, N., Barbosa, H., & Azevedo, L. F. (2018). CodeMaster--Automatic Assessment and Grading of App Inventor and Snap! Programs. Informatics in Education, 17(1), 117-150.

[67] Combéfis, S., & Paques, A. (2015, July). Pythia reloaded: An intelligent unit testing-based code grader for education. In Proceedings of the 1st International Workshop on Code Hunt Workshop on Educational Software Engineering (pp. 5-8).

[68] Delev, T., & Gjorgjevikj, D. (2012). E-Lab: Web-based system for automatic assessment of programming problems. Web proceedings ICT-Innovations.