# Abordagens, Práticas e Desafios da Avaliação Automática de Exercícios de Programação

**Márcia G. de Oliveira** [1], **Elias Oliveira** [1]

[1] Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Espírito Santo
Vitória – ES – Brasil

***Abstract.*** *Automatic assessment has emerged as an important method to help teachers to correct exercises and enable immediate feedback for students, especially in large classes. Some proposals have evaluated items as correct execution of programs, coding, programming style, software metrics, plagiarism and even psychological aspects. However, despite the programming assessment systems have evolved, such systems are still limited to evaluate if educational goals really have been achieved. Thus, among many challenges, the main challenge of programming automatic assessment is to assess what is actually a teacher wants to evaluate. Aiming to discuss solutions to this challenge, this paper presents a review of programming automatic assessment and points out ways to make it a functional assessment in the next ten years.*

**Palavras-chave:** *Automatic Assessment, Programming Exercises.*

***Resumo.*** *A avaliação automática de programação surgiu como um importante método para auxiliar professores na correção de exercícios e possibilitar feedbacks imediatos a estudantes, principalmente em turmas numerosas. As soluções propostas têm avaliado itens como a execução correta de um programa, a codificação, o estilo de programação, as métricas de software, o plágio e até aspectos psicológicos. Mas, embora os sistemas de avaliação automática de programação tenham evoluído, tais sistemas ainda são limitados em avaliar se de fato objetivos educacionais foram alcançados. Dessa forma, entre tantos desafios, o principal desafio da avaliação automática de programação é avaliar o que de fato um professor de programação quer avaliar. Com os objetivos de discutir e propor soluções para esse desafio, este trabalho apresenta um sumário do estado da arte da avaliação automática de programação e aponta caminhos para tornar essa avaliação funcional nos próximos dez anos.*

**Palavras-chave:** *Avaliação Automática, Exercícios de Programação.*

## 1. Introdução

O conceito de avaliação automática de programação foi introduzido na década de 60 por Hollingsworth e surgiu como um importante método para auxiliar professores na correção de exercícios e possibilitar *feedbacks* imediatos a estudantes, principalmente em turmas numerosas[Hollingsworth 1960, Ebrahimi 1994].

As principais abordagens de avaliação automática de programação são a análise dinâmica, a análise estática e a análise dinâmico-estática. A análise dinâmica é baseada na

execução de um programa e avalia a corretude através da testagem funcional e estrutural. A análise estática é baseada na escrita do código-fonte e avalia itens como erros de programação de ordem sintática, semântica e estrutural bem como o estilo de programação. Já a análise dinâmico-estática é uma combinação da análise dinâmica e estática.

Aplicando essas abordagens de avaliação, as iniciativas de avaliação automática de programação têm contemplado os seguintes itens:

- A execução correta de um programa [Saikkonen et al. 2001]
- A codificação [Xu and Chee 2003, Wu et al. 2007, Naude et al. 2010]
- O estilo de programação [Rees 1982, Jackson and Usher 1997]
- As métricas de *software* [Hung et al. 1993, Truong et al. 2004]
- Indícios de plágios [Moussiades and Vakali 2005]
- Aspectos psicológicos [Curtis et al. 1979]

Embora os sistemas de avaliação automática de programação tenham evoluído, conforme destacam [Romli et al. 2010], tais sistemas ainda são limitados em avaliar se de fato objetivos educacionais foram alcançados. O principal desafio da avaliação automática de programação é, portanto, avaliar exercícios de programação o mais próximo possível de como um professor de programação avaliaria.

Com os objetivos de discutir e propor soluções para esse desafio, este trabalho apresenta um sumário do estado da arte da avaliação automática de programação e aponta caminhos para tornar essa avaliação funcional nos próximos dez anos.

Nas seções a seguir, apresentamos as abordagens, as práticas e os desafios da avaliação automática de programação no Brasil e no mundo. Na Seção 2, descrevemos as abordagens de avaliação automática de programação. Nas Seção 3, destacamos as principais iniciativas no desenvolvimento de avaliadores automáticos. Na Seção 4, apresentamos os principais desafios da avaliação automática de programação para os próximos anos. Na Seção 5, apontamos caminhos de pesquisas a serem seguidos visando vencer esses desafios em busca de uma verdadeira avaliação da aprendizagem de programação.

## 2. Abordagens de Avaliação Automática de Programação

Um bom sistema de avaliação automática de programas desenvolvidos por alunos iniciantes em programação deve atender os seguintes requisitos [Romli et al. 2010]: realizar testagens suficientes, verificar se programas estão de acordo com a especificação, reconhecer erros sintáticos e semânticos e fornecer *feedback* imediato.

Visando atender a esses requisitos, atualmente utilizam-se três abordagens de avaliação automática de programação: a análise dinâmica, a análise estática e a análise dinâmico-estática. Essas abordagens são descritas a seguir.

### 2.1. A Análise Dinâmica

A análise dinâmica tem como principal objetivo descobrir erros de execução em um programa [Zin and Foxley 1994]. Para isso, ela informa se um programa funciona corretamente conforme uma especificação através de testes da caixa-preta e da caixa-branca.

O teste da caixa-preta avalia o comportamento externo do programa, isto é, se a saída está correta de acordo com as entradas fornecidas. Já o teste da caixa-branca avalia o comportamento interno avaliando se os componentes de um programa funcionam.

Apesar de muito utilizada, a análise dinâmica é um modelo que oferece muitas desvantagens. Entre elas destacamos o tratamento da variabilidade quando há liberdade de escrita de código e a limitação de *feedback* ao avaliar somente os resultados e não o processo da programação [Naude et al. 2010]. Além disso, como a análise dinâmica é sensível a pequenos erros [Naude et al. 2010], um simples erro pode provocar falhas em todos os testes dinâmicos e resultar em uma avaliação injusta da aprendizagem.

A principal deficiência da análise dinâmica, porém, é a forte dependência do oráculo, isto é, de um mecanismo pelo qual a corretude da saída pode ser checada [Zin and Foxley 1994]. Como para muitos tipos de problemas não existe um oráculo padrão, nesses casos, a análise dinâmica seria inviável.

### 2.2. A Análise Estática

A análise estática é a estratégia de avaliação automática baseada na análise de código-fonte. Nessa abordagem, aplicam-se várias técnicas para analisar estilo de programação, métricas de software, similaridade estrutural e não-estrutural bem como para detectar palavras-chave, erros sintáticos e semânticos e até plágios [Rahman et al. 2008].

As principais vantagens da análise estática são o menor custo, a menor dependência do oráculo e a possibilidade de oferecer uma avaliação mais próxima da avaliação humana. O custo da análise estática é menor do que o da análise dinâmica porque não requer tempo e esforço de execução em máquinas servidoras [Truong et al. 2004].

A avaliação de análise sintática é mais justa porque contempla o processo de construção de programas. No entanto, a análise estática não contempla os resultados de execução como a corretude, funcionalidade e eficiência, que são importantes itens de avaliação para muitos professores de programação [Ala-Mutka 2005, Rahman et al. 2008].

### 2.3. A Análise Dinâmico-Estática

A análise dinâmico-estática é a abordagem que combina as abordagens de avaliação estática e dinâmica e é considerada a abordagem mais adequada para avaliar objetivos educacionais no domínio da programação [Romli et al. 2010].

A principal vantagem da análise dinâmico-estática é reduzir as desvantagens de aplicar apenas a análise estática ou a análise dinâmica. Como desvantagem, a análise dinâmico-estática herda a forte dependência dos modelos de soluções da análise estática e dos casos de testes, da análise dinâmica. Essa desvantagem amplia-se quando os programas submetidos têm alta variabilidade de soluções, as saídas não são previsíveis e demandam-se muitos casos de testes.

## 3. Iniciativas de Avaliação Automática de Programação

As Tabelas 1, 2 e 3 ilustram como as propostas de avaliação automática de programação evoluíram desde a década de 60 até a presente década.

Entre os trabalhos de análise dinâmica da Tabela 1, destacamos o *ProgTest*, que é um sistema recente de apoio automatizado à avaliação de submissões de programas escritos em Java junto com seus casos de testes.

O processo do *ProgTest* consiste em compilar os programas do professor e do aluno, executar os critérios de testes funcional e estrutural em ambos e integrar técnicas de

| Análise Dinâmica | | |
|---|---|---|
| **Trabalhos** | **Métodos** | **Características de Avaliação** |
| [Hollingsworth 1960] | Caixa-preta | Teste funcional de corretude |
| [Blumenstein et al. 2004] - GAME | Caixa-branca | Teste estrutural em programas Java e C; Notas atribuídas próximas ao do avaliador humano |
| [C.M. Tang and Poon 2009] | Caixa-preta | Teste de corretude por comparação de *tokens* da saida |
| [De Souza et al. 2011] - *ProgTest* | Caixa-preta e Caixa-branca | Avalia programas por critérios funcionais e estruturais; Avalia casos de testes; O oráculo é formado por soluções-modelos e casos de testes. |

**Tabela 1. Evolução da Análise Dinâmica**

testagem a conceitos de programação. Mas o trabalho do professor no *ProgTest* aumenta porque, além dos modelos de soluções, o professor deve fornecer também os casos de testes e, com a variabilidade de soluções, o problema amplia ainda mais.

| Análise Estática | | |
|---|---|---|
| **Trabalhos** | **Método** | **Características de Avaliação** |
| [Curtis et al. 1979] | Métricas de Software | Avalia complexidade psicológica na manutenção de software, mede performance de programação |
| [Rees 1982] - STYLE | Métricas de Estilo | Avalia estilo de programação por contagens de atributos de código-fonte |
| [Hung et al. 1993] | Métricas de Avaliação Automática | Avalia habilidades de programar, complexidade, estilo de programação e eficiência |
| [Xu and Chee 2003] | Diagnóstico baseado em transformação de código | Reduz programas a árvores sintáticas abstratas e a grafos para comparar soluções e detectar erros |
| [Truong et al. 2004] | Análise de Similaridade estrutural e Métricas da Engenharia de Software | Avalia qualidade da solução e similaridade estrutural entre um programa Java e a solução modelo |
| [Rahman et al. 2008] | Caixa Branca | Análise não-estrutural de similaridade para comparação de pseudocódigos |
| [Wu et al. 2007] - AnalyseC | Métricas de software, otimização de compilador e técnicas de visualização | Análise de programas em Linguagem C em nível estrutural e semântico |
| [Gerdes et al. 2010] | Estratégias de padronização de código | Avalia corretude e boas práticas de programação |
| [Naude et al. 2010] | Medida de similaridade e modelos de regressão linear | Representa códigos por grafos, trata variabilidade, uso de modelos de regressão para predição de notas |

**Tabela 2. Evolução da Análise Estática**

Destacamos, entre os trabalhos de análise estática da Tabela 2, o método de avaliar programas por similaridade de grafos proposto por [Naude et al. 2010]. Nesse trabalho, os programas desenvolvidos por alunos são normalizados a árvores sintáticas abstratas e as variações das soluções como as expressões lógicas, as estruturas de controle condicional e de repetição, são normalizadas. As notas são preditas com base em notas de provas de outras turmas por um modelo de regressão linear, que tem como variáveis independentes os maiores índices de similaridades entre um programa não pontuado e os programas já avaliados. Os resultados de [Naude et al. 2010] indicam alta correlação entre a nota do professor e a nota predita em notas mais altas, mas não nas mais baixas.

| Análise Dinâmico-Estática | | |
|---|---|---|
| **Trabalhos** | **Método** | **Características de Avaliação** |
| [Jackson and Usher 1997] - *Assyst* | Caixa preta, Caixa Branca, Análise estrutural | Avalia corretude, eficiência, estilo, complexidade e adequação a dados de testes |
| [Saikkonen et al. 2001] - *Scheme-robo* | Caixa-preta, Caixa-branca | Avalia corretude por grupo de palavras-chave, estilo de programação, tempo de execução e plágio |
| [Benford et al. 1995] - *Ceilidh* | Métricas de Análise Dinâmica e Estática | Analisa *layout* tipográfico, complexidade estrutural, fraquezas estruturais, características de código-fonte, corretude funcional e eficiência |
| [Wang et al. 2011] - AutoLep | Representação de programas por grafos | Atribui notas, analisa sintaxe, semântica e funcionalidade de programas em Linguagem C. |
| [Vujosevic-Janicic et al. 2013] | Testagem, verificação de software e medida de similaridade de fluxo de controle | Verifica saída correta a partir de entradas fornecidas, suporta várias linguagens de programação, aplica modelos de regressão para predição de notas, é adaptável a diferentes estilos de correção |

**Tabela 3. Evolução da Análise Dinâmico-Estática**

Entre os trabalhos mais completos de análise dinâmico-estática da Tabela 3, destacamos o *Ceilidh* de [Benford et al. 1995] e a estratégia de [Vujosevic-Janicic et al. 2013], que é uma proposta mais recente que combina técnicas de testagem, análise e predição.

Na análise de código do *Ceilidh*, são computadas estatísticas como a média de caracteres por linha, a média de espaços por linha, número de comentários e de variáveis globais. Na complexidade estrutural, o *Ceilidh* analisa itens como número de palavras reservadas, número de estruturas de controle condicional e de repetição, número de chamadas de funções e número de *gotos*. Para verificação das deficiências estruturais, utiliza-se o analisador de código C do Linux chamado *Lint* [Orcero 2000]. As características de código-fonte são verificadas pela análise dos construtos do programa, removendo-se *strings* e comentários, o que pode ser útil na análise de plágios. Já a corretude dinâmica é avaliada submetendo um programa a vários casos de testes e a eficiência dinâmica, pelo número de vezes que cada linha do programa é executada.

A estratégia de [Vujosevic-Janicic et al. 2013] pode ser considerada a proposta mais inovadora porque seus resultados empíricos mostram precisão na predição de notas e alta correlação entre as notas preditas e as notas de professores. Além disso, a solução de [Vujosevic-Janicic et al. 2013] pode ser treinada para adaptar-se a diferentes estilos de correção. Mas, apesar dessas vantagens, a proposta de [Vujosevic-Janicic et al. 2013] herda dos trabalhos anteriores a dependência de oráculos e a carência de técnicas eficazes para tratar a variabilidade de soluções.

### 3.1. A Avaliação Automática de Programação no Brasil

No Brasil, a avaliação automática de exercícios de programação é ainda pouco expressiva e predominam as soluções baseadas em análise dinâmica. As discussões sobre avaliação automática no Brasil chamam à atenção para duas questões [Moreira and Favero 2009]: como avaliar manualmente todos os exercícios dos estudantes para turmas grandes e como fornecer *feedback* em curto prazo.

A Tabela 4 apresenta os principais trabalhos de avaliação automática de programação no Brasil com suas abordagens e principais destaques.

| Avaliação Automática | | |
|---|---|---|
| **Trabalhos** | **Abordagens** | **Destaques** |
| [Dias 2001] | Análise Estática | Reduz código-fonte a uma representação padrão após análise léxica e sintática, duas etapas de avaliação (correção e classificação) |
| [Nunes and Lisboa 2004] | Análise Dinâmica | Testador automático que avalia corretude por correção funcional e de estado das classes comparando classes implementadas em Java contra um conjunto-modelo de classes |
| [Campos and Ferreira 2004] - BOCA | Análise Dinâmica | Avaliação de corretude com base em Entrada/Saída, submissão *online* de exercícios; Versão estendida com envio múltiplo de arquivos: BOCA-LAB [Franca et al. 2011] |
| [Moreira and Favero 2009] | Análise Estática | Representação de programas por métricas da Engenharia de Software para medir complexidade e esforço, predição de notas por modelos de regressão linear múltipla tendo essas métricas como variáveis independentes |

**Tabela 4. Avaliação Automática de Programação no Brasil**

Embora a construção de sistemas de avaliação automática no Brasil ainda seja pouco expressiva, os desafios e caminhos de pesquisas apresentados nas próximas seções podem direcionar o desenvolvimento de avaliadores automáticos de programação no país.

## 4. Desafios da Avaliação Automática de Programação

Embora várias sistemas de avaliação automática de programação tenham sido desenvolvidas nas últimas décadas [Malmi et al. 2002, Ala-Mutka 2005, Romli et al. 2010], são apontados ainda os seguintes desafios:

1. Gerar representações alternativas de programas além das árvores sintáticas abstratas [Xu and Chee 2003, Wu et al. 2007] e dos grafos [Naude et al. 2010].
2. Reduzir a dependência de oráculos (ou gabaritos)
3. Tratar a variabilidade em exercícios de programação [Naude et al. 2010].
4. Estabelecer medições que garantam a confiabilidade dos sistemas de avaliação automática e motivem professores a utilizá-los.
5. Orientar a avaliação por uma perspectiva pedagógica [Ihantola et al. 2010].
6. Desenvolver prevenção contra códigos maliciosos submetidos por alunos.
7. Controlar com maior rigor os plágios de exercícios
8. Garantir *feedback* (atribuição de notas) correto para melhor avaliar alunos.
9. Todos os desafios acima ampliam-se no contexto de *MOOCs (Massive Open Online Courses - Cursos Online Abertos e Massivos)* [Pieterse 2013].

### 4.1. Desafios da Análise Dinâmica

Um requisito essencial apontado por [Ala-Mutka 2005] para os sistemas de análise dinâmica é garantir ambientes seguros de execução através de *sandboxes* ("caixas de areia").

Um exemplo de execução de risco em sistema de avaliação automática é a submissão de programas em *loop* que gravam dados em arquivos. O alto consumo de tempo de processamento e de memória podem causar tão-logo consideráveis danos ao sistema de avaliação e até a paralisação da sua infra-estrutura.

Estratégias de prevenção e de ação devem, portanto, ser criteriosamente desenvolvidas contra códigos de execução perigosa em sistemas de análise dinâmica.

### 4.2. Desafios da Análise Estática

O principal desafio da análise estática é tratar a variabilidade de soluções em códigos-fontes, o que aumenta o trabalho de professores, uma vez que estes precisam fornecer vários modelos de soluções [Rahman et al. 2008]. De acordo com [Naude et al. 2010], as principais causas das variabilidades em códigos-fontes são as seguintes:

- Vários algoritmos para um mesmo problema, como os algoritmos de ordenação.
- Escolha dos identificadores, pois é difícil mapear os nomes de variáveis e fazer a correspondência delas em diferentes programas.
- Decomposição do programa em funções, sentenças e expressões
- Ordenação de sentenças independentes, que é um problema quando, em programas, a ordem dos textos é considerada.
- Multiplicidade de formas, na mesma linguagem de programação, para escrever uma mesma sentença. Isso acontece muito na Linguagem C.
- Identidades algébricas e lógicas
- Várias formas de escrever uma estrutura de controle de repetição.
- Várias formas de escrever estruturas de controle condicional.

### 4.3. Desafios da Avaliação de Análise Dinâmico-Estática

A avaliação de análise dinâmico-estática combina os desafios das avaliações de análise dinâmica e estática, sendo os principais deles a redução de dependência do oráculo e o tratamento da variabilidade em códigos e nas saídas geradas.

Os sistemas de análise dinâmico-estática, por dependerem parcial ou totalmente de oráculos, são considerados semi-automáticos. Dessa forma, os passos mais ousados a serem dados na construção de tais sistemas devem depender minimamente do especialista humano e evoluir na aprendizagem de máquina.

## 5. Caminhos de Pesquisa

Os grandes desafios da avaliação automática de programação atualmente são tratar a variabilidade, reduzir a dependência de oráculos e predizer notas com precisão. Para vencer esses e os outros desafios apresentados, apontamos os seguintes caminhos de pesquisa:

1. Propor soluções inicialmente para o domínio da programação introdutória, uma vez que a maioria das propostas ainda não venceu os desafios nesse domínio.
2. Gerar representações numéricas que melhor sintetizem o código-fonte e itens de execução [Pieterse 2013] para tratamento matemático da análise de código e da predição de notas.
3. Separar classes de soluções por técnicas como o *Clustering*, conforme sugere [Naude et al. 2010], para atacar o problema da variabilidade de soluções.
4. Identificar automaticamente modelos de soluções a partir das classes de soluções do Item 3, o que reduz o esforço de professores na composição de gabaritos.
5. Gerar modelos matemáticos que possibilitem descobrir estilos de correção de diferentes professores, o que motivaria mais o uso de avaliadores automáticos.

6. Desenvolver estratégias de *Sandbox* para execução segura e em massa de programas submetidos em ambientes virtuais. Uma boa ideia é executar as submissões de exercícios de programação em uma rede paralela de máquinas virtuais.
7. Integrar sistemas de avaliação automática a sistemas já existentes de detecção de plágios em exercícios de programação, como o *P-detect* de [Moussiades and Vakali 2005] e vários outros destacados por [Pieterse 2013].
8. Desenvolver estratégias de controle de submissão de códigos maliciosos. Para isso, podem ser utilizados ambientes LMS com técnicas de controle de tempo de execução, tamanho de arquivos e de quantidade de submissões.
9. Flexibilizar o formato de submissões para que alunos especifiquem roteiro de execução (por arquivo *makefile*, por exemplo) e submetam programas, casos de testes e outros arquivos necessários para a execução de suas soluções [Pieterse 2013].
10. Gerar relatórios com uma avaliação clínica e multidimensional do processo de aprendizagem a partir de códigos-fontes. As métricas da Engenharia de Software e ferramentas de visualização de informação podem auxiliar nessa tarefa.

No caso de cursos massivos, em que os desafios da avaliação automática ampliam-se, as propostas de sistemas de avaliação de programação devem considerar os vários fatores de sucessos e de problemas apontados por [Pieterse 2013].

## 6. Considerações Finais

Este trabalho apresentou um sumário do estado da arte da avaliação automática de exercícios de programação no Brasil e no mundo, apontando tendências, desafios e caminhos de pesquisas que orientem o desenvolvimento dos futuros sistemas de avaliação.

Até a presente década, os principais desafios da avaliação automática de programação a serem vencidos são o tratamento da variabilidade de soluções, a redução da dependência de oráculos e a criação de estratégias para cursos massivos.

A contribuição deste trabalho para vencer esses desafios é promover a discussão deles na comunidade científica de interesse e oferecer direções para tornar a avaliação automática de programação funcional, massiva e alinhada com objetivos educacionais nos próximos dez anos no Brasil.

## Referências


Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102.

Benford, S. D., Burke, E. K., Foxley, E., and Higgins, C. A. (1995). The Ceilidh system for the automatic grading of students on programming courses. In *Proceedings of the 33rd annual on Southeast regional conference*, ACM-SE 33, pages 176–182, New York, NY, USA. ACM.

Blumenstein, M. M., Green, S., Nguyen, A. T., and Muthukkumarasamy, V. (2004). Game: A generic automated marking environment for programming assessment.

Campos, C. and Ferreira, C. (2004). Boca: um sistema de apoio para competições de programação. In *XII Workshop de Educação em Computação (WEI) - SBC 2004)*, Salvador, BA.

C.M. Tang, Y. Y. and Poon, C. (2009). An approach towards automatic testing of student programs using token patterns. In *Proceedings of the 17th International Conference on Computers in Education*, pages 118–190, Hong Kong.

Curtis, B., Sheppard, S. B., Milliman, P., Borst, M. A., and Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Trans. Softw. Eng.*, 5(2):96–104.

De Souza, D., Maldonado, J., and Barbosa, E. (2011). Progtest: An environment for the submission and evaluation of programming assignments based on testing activities. In *Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on*, pages 1 –10.

Dias, P. (2001). Avaliação automática de exercícios em sql. Master's thesis, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal.

Ebrahimi, A. (1994). Novice programmer errors: language constructs and plan composition. *International Journal of Human-Computer Studies*, 41(4):457 – 480.

Franca, A., Soares, J., Gomes, D., and G.C.Barroso (2011). Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programacão com integracão ao ambiente moodle. *RENOTE - Revista Novas Tecnologias na Educaccão*, 9(1).

Gerdes, A., Jeuring, J. T., and Heeren, B. J. (2010). Using strategies for assessment of programming exercises. In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, pages 441–445, New York, NY, USA. ACM.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Commun. ACM*, 3(10):528–529.

Hung, S.-l., Kwok, L.-f., and Chung, A. (1993). New metrics for automated programming assessment. In *Proceedings of the IFIP WG3.4/SEARCC (SRIG on Education and Training) Working Conference on Software Engineering Education*, pages 233–243, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co.

Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (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 Calling '10, pages 86–93, New York, NY, USA. ACM.

Jackson, D. and Usher, M. (1997). Grading student programs using assyst. *SIGCSE Bull.*, 29(1):335–339.

Malmi, L., Korhonen, A., and Saikkonen, R. (2002). Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, ITiCSE '02, pages 55–59, New York, NY, USA. ACM.

Moreira, M. P. and Favero, E. L. (2009). Um ambiente para ensino de programação com feedback automático de exercícios. In *XVII Workshop Sobre Educação em Computação (WEI) - CSBC 2009*.

Moussiades, L. and Vakali, A. (2005). Pdetect: A clustering approach for detecting plagiarism in source code datasets. *The computer journal*, 48(6):651–661.

Naude, K. A., Greyling, J. H., and Vogts, D. (2010). Marking student programs using graph similarity. *Computers & Education*, 54(2):545 – 561.

Nunes, I. and Lisboa, M. (2004). Testador automático e método de avaliacão de programas em java. In *XVI Salão de Iniciacão Científica e XIII Feira de Iniciacão Científica*, Porto Alegre, RS.

Orcero, D. S. (2000). The code analyser LClint. *Linux J.*, 2000(73).

Pieterse, V. (2013). Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, CSERC '13, pages 4:45–4:56, Open Univ., Heerlen, The Netherlands, The Netherlands. Open Universiteit, Heerlen.

Rahman, K. A., Ahmad, S., Nordin, M. J., and Maklumat, F. T. D. S. (2008). The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique.

Rees, M. J. (1982). Automatic assessment aids for pascal programs. *SIGPLAN Not.*, 17:33–42.

Romli, R., Sulaiman, S., and Zamli, K. (2010). Automatic programming assessment and test data generation a review on its approaches. In *Information Technology (ITSim), 2010 International Symposium in*, volume 3, pages 1186 –1192.

Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assessment of programming exercises. *SIGCSE Bull.*, 33:133–136.

Truong, N., Roe, P., and Bancroft, P. (2004). Static analysis of students' java programs. In *Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30*, ACE '04, pages 317–325, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Vujosevic-Janicic, M., Nikolic, M., Tosic, D., and Kuncak, V. (2013). Software verification and graph similarity for automated evaluation of students assignments. *Information and Software Technology*, 55(6):1004 – 1016.

Wang, T., Su, X., Ma, P., Wang, Y., and Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course. *Comput. Educ.*, 56(1):220–226.

Wu, W., Li, G., Sun, Y., Wang, J., and Lai, T. (2007). AnalyseC: A Framework for Assessing Students' Programs at Structural and Semantic Level. In *Control and Automation, 2007. ICCA 2007. IEEE International Conference on*, pages 742 –747.

Xu, S. and Chee, Y. S. (2003). Transformation-based diagnosis of student programs for programming tutoring systems. *IEEE Transactions on Software Engineering*, 29:360–384.

Zin, A. and Foxley, E. (1994). Analyse:an automatic program assessment system. *Malaysian Journal of Computer Science*, 7:123–142.