Combinando técnicas de análise estática e avaliação dinâmica para avaliação de código em ambientes de aprendizagem de programação

Gilvani Schneider[1]
Patrícia Augustin Jaques[1]

Resumo: O aprendizado da programação tem se revelado desafiador para a maioria dos alunos do ensino superior. A estratégia de ensino empregada é geralmente o *learning by doing*, envolvendo a aplicação de vários exercícios de programação. Essas atividades são facilitadas por ferramentas que possibilitam aos alunos a submissão da solução dos exercícios pela internet e automatizam o processo de correção. Este trabalho apresenta um mecanismo de *feedback*, que combina técnicas de análise estática e avaliação dinâmica de código, e um mecanismo de cadastro de casos de teste pelo docente, que automatiza a geração de casos de teste pela validação da assinatura das classes a serem submetidas, agilizando assim a realização dessa atividade pelo professor. Este estudo foi integrado a um ambiente virtual de aprendizagem para apoiar as disciplinas de programação, que contém funcionalidades para cadastro de projetos de programação e submissão de soluções pelos alunos, chamado Feeper. Uma avaliação deste trabalho foi realizada com uma amostra de conveniência de treze estudantes de graduação e pós-graduação em informática que usaram Feeper para desenvolver projetos de programação por três semanas. Após esse período, foi aplicado um questionário com questões fechadas cujas respostas seguiam a escala Likert de concordância. Os resultados da avaliação sugerem que as melhorias propostas e implementadas no Feeper aumentaram a qualidade do auxílio dado pela ferramenta aos alunos durante a programação de suas soluções.

Palavras-chave: Análise estática de código. Avaliação dinâmica de código. Ensino de programação.

Abstract: *Learning to program has proved to be a challenge for most higher education students. The learning strategy adopted by teachers is usually learning by doing, involving the application of various programming exercises. These activities are usually facilitated by tools that allow students to submit their solutions online and also automate the assessment process. The proposed work presents an improved feedback mechanism that combines static analysis and dynamic evaluation of code. This work also provides a mechanism for test cases registration, which is able to automate the generation of test cases that validate the signature of the classes to be submitted. The proposed work was integrated into Feeper: a virtual learning environment to support programming courses, which contains features for programming problems registration and submission of code solution by students. An evaluation of the proposed work was conducted with a convenience sample of 13 undergraduate and graduate students in computer science, who used Feeper to develop code solutions during three weeks. After this period, students filled in a questionnaire whose answers followed the Likert scale. The evaluation results suggest that the proposed work improved the quality of the feedback provided by Feeper for students.*

Keywords: *Dynamic evaluation of code. Programming teaching. Static analysis of code.*

[1] Programa de Pós-Graduação em Computação Aplicada, Universidade do Vale do Rio dos Sinos, *campus* São Leopoldo, Av. Unisinos, 950 – São Leopoldo, RS, Brasil
{gilschneider90@gmail.com; pjaques@unisinos.br}

http://dx.doi.org/10.5335/rbca.2015.5362

1 Introdução

As disciplinas de programação abordam um dos conceitos básicos da área de tecnologia da informação (TI), e são ministradas geralmente no primeiro e no segundo semestres dos cursos dessa área. Tais disciplinas visam ensinar aos alunos a escrita de instruções que possam ser interpretadas pelo computador e convertidas posteriormente em programas executáveis.

A dinâmica de ensino nessas disciplinas normalmente envolve ministrar os conceitos e regras de programação em aula com aplicação de exercícios para fixar o conteúdo. Essa estratégia de ensino e aprendizagem é conhecida como *learning by doing* [1]. O nível de complexidade dos exercícios é incrementando gradualmente, de modo que não seja ignorada nenhuma etapa importante no aprendizado do aluno [2].

Para atingir bons resultados, é necessária a aplicação de um grande número de exercícios. No entanto, quanto maior o número de exercícios aplicados, maior a carga de trabalho do professor. Como consequência, há uma menor disponibilidade de tempo para o professor interagir com os alunos e o aumento do prazo de correção dos exercícios. Com isso, o professor demorará mais para obter uma visão geral sobre a situação dos estudantes, atrasando assim ações remediadoras, caso os estudantes não estejam adquirindo o aprendizado desejado [3].

Diante desse quadro, tornam-se necessários a abordagem e o desenvolvimento de novas estratégias que agilizem o processo de aplicação dos exercícios, facilitem a comunicação entre professor e alunos e realizem a correção automática dos exercícios, fornecendo *feedback* instantâneo e de qualidade para os alunos, para auxiliar assim o entendimento dos conteúdos ministrados em aula.

Muitas vezes, ambientes virtuais de aprendizagem (AVAs) são empregados para apoiar as disciplinas de ensino de programação, sejam elas presenciais ou a distância [2], [4]-[9]. Os AVAs são sistemas *web* que visam agilizar e facilitar a interação entre alunos e professores, permitindo que o professor poste conteúdos, exercícios, notas, notícias e afins, e possibilitando ao aluno o acesso aos dados postados pelo professor, a possibilidade de entrar em contato com o professor e a submissão para avaliação da resolução dos exercícios.

Seguindo essa tendência, no primeiro semestre de 2014, pesquisadores da Universidade do Vale do Rio dos Sinos (Unisinos) desenvolveram e implantaram um AVA para apoiar o ensino/aprendizagem de programação no ensino superior, denominado Feeper (disponível em: <http://feeper.unisinos.br>.). Entre as principais e mais importantes funcionalidades desse ambiente estão a correção automática dos exercícios e o *feedback* personalizado e instantâneo para o aluno após a submissão da solução do exercício [7].

Segundo Alves e Jaques [7], a ferramenta demonstrou resultados positivos após testes realizados na própria universidade. No entanto, ela ainda apresentava funcionalidades que necessitavam melhorias. No quesito interface e usabilidade, as principais deficiências eram a baixa usabilidade e a alta onerosidade da funcionalidade de cadastro de testes. No quesito correção automática dos exercícios, as principais deficiências eram o fato de o módulo corretor estar altamente acoplado ao sistema acessado por alunos e professores, a ausência de políticas de segurança a serem aplicadas sobre o código do aluno a ser executado, assim como o fato de a ferramenta realizar somente a avaliação dinâmica do código. Quando a solução do aluno apresentava erros de compilação, o sistema apresentava a mensagem gerada pelo compilador, enquanto que na avaliação dinâmica, quando a solução do aluno apresentava um resultado diferente do esperado, era apenas apresentada uma mensagem de retorno definida pelo professor.

No trabalho proposto, o processo de correção do Feeper foi aprimorado; o módulo corretor foi desacoplado do sistema acessado pelos professores e alunos, evitando assim que a velocidade de utilização do sistema seja impactada pelo processo de correção dos exercícios. Além disso, foi implementado um mecanismo de correção próprio, passível de personalização e capaz de fornecer mais detalhes sobre os problemas encontrados na solução do aluno. Além disso, o *feedback* apresentado aos alunos foi aperfeiçoado, pois, além das mensagens definidas pelo professor, tornou-se possível a apresentação de explicações mais detalhadas sobre o porquê dos erros encontrados na avaliação dinâmica da solução do aluno. O *feedback* fornecido aos alunos também passou a contar com mensagens provenientes da análise estática da solução.

A proposta deste trabalho é parte integrante dos esforços de aprimoramento do Feeper, visando transformá-lo em um sistema tutor inteligente (STI). Dessa forma, além de novas funcionalidades, também foram implementadas melhorias em funcionalidades já existentes do sistema. Como diferencial, propõe-se um

mecanismo que apresenta um *feedback* resultante da combinação de mensagens provenientes da análise estática e avaliação dinâmica das soluções dos alunos. Além disso, também apresenta um mecanismo de cadastro de exercícios capaz de automatizar a geração dos testes a partir da assinatura das classes da solução, agilizando assim a realização dessa atividade pelo professor. Outro resultado do presente trabalho é um mecanismo que analisa as soluções dos alunos, apresentando o percentual de similaridade entre elas, facilitando assim a identificação de plágios entre os códigos.

O desenvolvimento deste artigo está organizado da seguinte forma: a segunda seção apresenta os conceitos básicos necessários para a compreensão deste trabalho, a terceira cita os trabalhos relacionados ao assunto aqui proposto, a quarta descreve o trabalho desenvolvido. Por fim, a quinta seção apresenta a avaliação e os resultados obtidos e a sexta apresenta as considerações finais deste trabalho.

2 Conceitos básicos

Nesta seção serão apresentados os conceitos básicos relevantes para a compreensão do trabalho, tais como ensino de programação, ambientes virtuais de aprendizagem, avaliação dinâmica de código e análise estática de código.

2.1 Ensino de programação

A programação é uma das bases da maioria dos cursos de TI. Ela é abordada em disciplinas introdutórias e continua sendo utilizada durante toda a vida acadêmica dos estudantes. Independente da ramificação da área de TI que o estudante pretenda se especializar, o entendimento da programação é sempre importante [3].

A programação envolve a escrita de um conjunto de regras que possam ser interpretadas por uma máquina. Ela desenvolve várias habilidades cognitivas dos estudantes, pois, ao idealizar soluções para problemas de programação, o estudante desenvolve habilidades de abstração e interpretação de problemas bem como a criatividade. Após escrever o conjunto de regras, o estudante desenvolve o raciocínio lógico e emprega conhecimentos matemáticos [10].

A dinâmica utilizada no ensino da programação é a do *learning by doing*. Uma das maneiras mais eficientes de se adquirir conhecimento em novas áreas é por meio da resolução de problemas específicos. Nessa dinâmica, os alunos são desafiados a atingir objetivos propostos pelo professor. Tais objetivos requerem a compreensão e aplicação dos conceitos ensinados em aula [11]. Durante seus esforços para atingir esses objetivos, o próprio aluno é capaz de identificar e corrigir fraquezas e falhas no seu aprendizado. Porém, essa técnica requer a aplicação de vários exercícios para que todos os conceitos envolvidos sejam abordados progressivamente, minimizando a carga de novos conceitos em cada exercício e evitando sobrecargas de conteúdo e frustações para o aluno [11].

Programar exige habilidades básicas de leitura, interpretação, lógica e matemática. No entanto, é cada vez mais comum encontrarmos alunos que concluem o ensino médio e ingressam no ensino superior sem uma base adequada nas disciplinas de Português e Matemática. Como consequência disso, apresentam sérias dificuldades na leitura e interpretação bem como na resolução de problemas matemáticos simples [10]. A união da grande gama de habilidades necessárias na programação, com a crescente falta de habilidades básicas dos alunos que ingressam no ensino superior, faz com que o desempenho dos estudantes nas disciplinas específicas de programação seja preocupante.

Estudos multi-institucionais apontam sérias deficiências no aprendizado dos alunos aprovados nas disciplinas de programação, além da incidência de um alto nível de desistência [12]. Muitas universidades relatam um índice de desistência em torno de 30% a 40%. No Brasil, conforme apontado por Gomes, Henriques e Mendes [9], essa realidade não é diferente. Segundo Giraffa e Móra [13], o índice de desistência na disciplina de Programação I (primeiro semestre) em uma universidade privada de Porto Alegre, no período de 2012 a 2013, ficou em torno dos 38%.

O alto índice de desistência nos cursos superiores faz com que o número de profissionais disponíveis no mercado de TI brasileiro diminua cada vez mais. González [14] previu que no Distrito Federal e nos estados de São Paulo, Rio de Janeiro, Paraná, Rio Grande do Sul, Minas Gerais, Bahia e Pernambuco, para o ano de 2014, haveria um déficit de 45 mil profissionais na área de TI. Esse déficit de profissionais faz com que empresas multinacionais tenham que repensar decisões de investir no mercado de TI brasileiro. Considerando que,

segundo González [14], esse mercado representa atualmente 4,5% do PIB brasileiro, ações que visam aumentar o índice de concluintes na área de TI são justificáveis pelo fato de que o aumento de profissionais disponíveis na área de TI permitirá maiores investimentos, fortificando e diversificando assim a economia brasileira.

2.2 Ambiente virtual de aprendizagem e sistemas tutores inteligentes

Ambientes virtuais de aprendizagem têm sido empregados para apoiar as disciplinas de ensino de programação, sejam elas presenciais ou a distância [2], [4]-[9]. Contudo, por não possuírem inteligência, os AVAs não são capazes de fornecer assistência individualizada aos estudantes. Eles se caracterizam basicamente por fornecer mecanismos de interação entre professores, ser um portal de disponibilização de material e, em áreas específicas, realizar a correção automatizada das soluções dos alunos.

De outro modo, os sistemas tutores inteligentes empregam técnicas e ferramentas da inteligência artificial para prover assistência individualizada. Para tanto, um STI é capaz de identificar o nível de conhecimento e estados afetivos do estudante, construindo um modelo de aluno. Com base nesse modelo, o STI propõe exercícios que melhor se adaptem às necessidades de aprendizado do aluno bem como fornece dicas e *feedback* (verifica se a solução dada é correta) sobre as soluções do estudante [15].

Como já foi mencionado, atualmente o Feeper caracteriza-se como um AVA, e este trabalho é parte integrante dos esforços desenvolvidos com o objetivo de transformá-lo em um STI. Desse modo, este trabalho contribui para que o Feeper seja capaz de extrair informações mais precisas do conhecimento do aluno a fim de prover assistência individualizada.

2.3 Avaliação automática de código

Atualmente, vários AVAs e STIs da área de ensino de programação suportam a avaliação automática das soluções dos exercícios. Essa validação geralmente ocorre por meio da aplicação de casos de teste no sistema alvo, validando as pós-condições e os resultados obtidos.

Caso de teste é o nome dado a um conjunto de instruções elaboradas para testar um *software*. O caso de teste é composto basicamente por dados de entrada, pré-condições de execução e ações a serem executadas no sistema alvo, resultados esperados e pós-condições de execução. Para que um sistema seja considerado correto, ele deve sempre satisfazer as pós-condições e trazer os resultados esperados dado o conjunto de entradas e pré-condições.

Além de avaliar a solução por meio da avaliação dinâmica e informar o resultado ao aluno, alguns AVAs e STIs de ensino de programação também se utilizam de técnicas de análise estática para aprimorar o *feedback* apresentado para o aluno.

2.3.1 Avaliação dinâmica de código

A avaliação dinâmica do código é a forma de avaliação comumente utilizada em AVAs ou STIs. Essa avaliação é possível por meio de componentes como Juízes Online, da implementação de mecanismos próprios de validação ou da utilização de *frameworks* de testes automatizados como o JUnit [16].

Esse tipo de avaliação tem como objetivo garantir que o código do aluno seja capaz de cumprir os objetivos propostos pelo professor no exercício. Para isso, é necessário que o código do aluno seja compilado, empacotado e executado, gerando os resultados corretos e atendendo as pós-condições estabelecidas. Esse tipo de avaliação é amplamente utilizada em AVAs e STIs de programação, entre esses podemos citar o WebCat [17], o BOSS [18] e o Javatool [19].

2.3.2 Análise estática de código

A análise estática do código não requer sua execução. Esse tipo de procedimento analisa a estrutura da solução com o objetivo de verificar a existência de más práticas de programação que comprometam a qualidade do código, como o desrespeito aos padrões de nomenclatura de classes e variáveis, os métodos muito complexos, as variáveis não usadas, entre outros problemas. A análise estática também identifica erros comuns de

programação, que geralmente ocasionam comportamentos inesperados durante a execução do código, tais como conversões de dados para tipos não compatíveis, *loops* infinitos, comparação incorreta de valores, etc.

Esse tipo de análise é geralmente feito com a utilização de ferramentas que o automatizam e retornam os problemas encontrados, pode-se destacar, entre essas ferramentas, Checkstyle, PMD e FindBugs. Esse tipo de análise é menos comum em AVAs ou STIs. Entre os principais sistemas que utilizam esse tipo de verificação, pode-se ressaltar o WebCat [20].

3 Trabalhos relacionados

Com o objetivo de reduzir a carga de trabalho do professor, Johnson [21] desenvolveu o AVA SpecCheck, que suporta a avaliação dinâmica de código e gera os casos de teste automaticamente por meio de uma solução padrão fornecida pelo professor. Os dados de teste usados para geração dos casos de teste são fornecidos na solução padrão por meio de uma estrutura de anotações em forma de comentários. O SpecCheck foi utilizado durante quatro semestres em uma universidade, e seu uso permitiu *feedback* instantâneo para os alunos, porém, não foi comprovada uma diminuição significativa na carga de trabalho do professor.

Weragama e Reye [22] utilizam como estudo de caso um STI para ensino da programação capaz de avaliar exercícios de construção de páginas *web* na linguagem PHP. Os autores afirmam que uma das maiores dificuldades na construção de um STI é o módulo de correção automática dos exercícios, pois um exercício dificilmente terá somente uma estrutura de solução. Para tentar resolver esse problema, os autores enfatizam que a verificação da solução do aluno deve ser realizada com base nos objetivos a serem atingidos e não na estrutura da solução. Para isso, os autores desenvolveram uma estrutura de verificação que permite definir e validar os objetivos para cada chamada do código do aluno. No entanto, o cadastro de casos de teste nessa estrutura requer muito conhecimento técnico, o que dificulta a realização dessa tarefa pelo professor. Para tentar remediar essa dificuldade, os autores criaram um repositório de exercícios, e o professor pode selecionar o exercício a ser aplicado para os alunos, porém, isso elimina a possibilidade de personalização dos exercícios pelo professor.

Moreira e Favero [8] propõem uma ferramenta integrada ao Moodle que implementa uma metodologia de avaliação dos exercícios de programação por meio de regressão linear baseada em n-gramas. O objetivo dessa abordagem é avaliar a solução do aluno por meio da estrutura e complexidade da solução. O diferencial dessa abordagem é que, mesmo que a solução do aluno atinja os objetivos especificados no exercício, ele não tira nota máxima até que a solução esteja estruturalmente otimizada. Com isso o aluno poderá identificar melhorias em sua solução e submeter uma novo resultado, visando aumentar a nota. É necessário prover vinte soluções do exercício para treinamento dos n-gramas. Somente após isso, a ferramenta é capaz de avaliar a estrutura da solução do aluno, algo que pode ser muito trabalhoso bem como pode limitar a capacidade de personalização dos exercícios.

Allowatt e Edwards [23] propõem uma abordagem na qual, além da solução do exercício, o aluno deve prover um conjunto de casos de teste para testar a solução. Os autores acreditam que a escrita de casos de teste ajuda a desenvolver as habilidades de programação dos estudantes. O aluno deve escrever os casos de teste utilizando o *framework* JUnit. Como estudo de caso, os autores utilizam o AVA WebCat. Nesse trabalho, foi desenvolvido um *plug-in* no qual o aluno pode fazer o *upload* da solução e dos casos de teste diretamente do ambiente de desenvolvimento para o WebCat. Os autores relatam que o trabalho desenvolvido foi aplicado em sua universidade durante dois anos e meio. Como resultado, foi obtida uma redução de 28% nos erros em códigos submetidos pelos alunos.

Com o objetivo de agilizar o processo de correção dos exercícios e prover um *feedback* instantâneo e personalizado, Alves e Jaques [7] desenvolveram o AVA Feeper, ambiente que permite que os alunos realizem a submissão de suas soluções de exercícios pela internet. Com o auxílio da ferramenta CodeJudge, categorizada como Juiz Online e incorporada ao Feeper, esse sistema realiza a avaliação estática da solução e provê *feedback* ao aluno. O Feeper foi utilizado durante dois meses na Unisinos, e permitiu o aumento do número de exercícios realizados pelos alunos bem como uma visualização mais clara do andamento do aprendizado dos alunos. Ele possibilitou também a diminuição da carga de trabalho do professor com a correção dos exercícios, porém gerou atividades extras manuais e onerosas, como o cadastro de casos de teste para a correção automática.

O principal diferencial da proposta deste trabalho em relação aos demais é a criação de um mecanismo que agilize o processo de cadastro de exercícios, com geração automática de testes de assinatura, visando diminuir a carga de trabalho do professor. Outro diferencial é a combinação da avaliação dinâmica do código

com a análise estática, provendo assim um *feedback* mais rico e que auxilia o aluno na identificação e resolução de erros na solução submetida. O trabalho desenvolvido é descrito em mais detalhes na próxima seção.

4 Trabalho desenvolvido

Esse trabalho busca aprimorar o *feedback* provido aos alunos pelo AVA Feeper, desenvolvido por Alves e Jaques [7]. O *feedback* provido pela ferramenta foi aprimorado por meio da reformulação da avaliação dinâmica e da combinação dessa avaliação com técnicas de análise estática de código.

Todas as mensagens resultantes da avaliação dinâmica passaram a contar com a mensagem personalizada pelo professor e uma mensagem gerada pelo sistema, que mostra a representação exata do erro. Da mesma forma, as mensagens resultantes da análise estática apresentam possíveis causas para os erros encontrados e indicadores de qualidade da solução do aluno.

Além disso, o trabalho proposto busca estender a funcionalidade de cadastro de exercícios a fim de facilitar e agilizar o desenvolvimento da atividade pelo professor bem como implementar uma funcionalidade que facilite a identificação de plágios entre as soluções dos alunos.

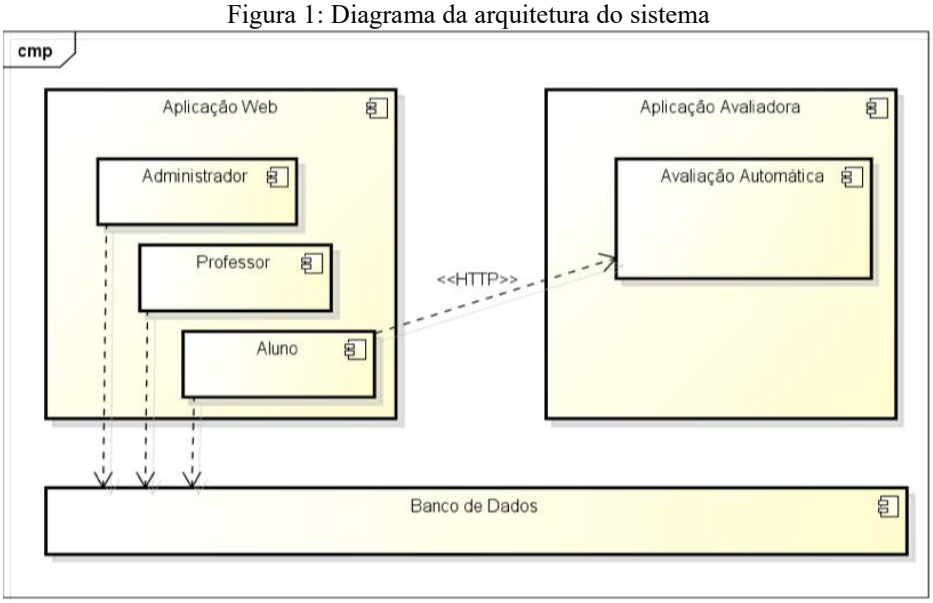
Uma descrição detalhada do Feeper, com imagens e vídeos, pode ser visualizada na URL http://feeper.unisinos.br. Atualmente, o ambiente encontra-se disponível apenas para uso dos cursos de informática fornecidos pela universidade dos autores.

4.1 Arquitetura do sistema

Conforme Alves e Jaques [7], o Feeper foi originalmente desenvolvido como uma aplicação para a internet dividida em quatro módulos. Dentre os quatro módulos, três são controlados por usuários, sendo eles aluno, professor e administrador. O quarto módulo é utilizado no processo de avaliação automática dos exercícios e é controlado pelo próprio sistema, sendo acionado cada vez que uma solução é submetida para avaliação. O trabalho proposto realizou alterações nos módulos operados pelo professor e pelos alunos e reimplementou o módulo controlado pelo sistema.

A Figura 1 ilustra a nova arquitetura do sistema, exibindo os módulos envolvidos e o banco de dados. Nessa estrutura, os módulos administrador, professor e aluno estão inseridos no componente aplicação *web*. Esse componente engloba a interface de usuário do sistema, a camada que controla a operação do sistema e as regras de negócio envolvidas bem como os mecanismos de acesso e escrita no banco de dados.

Figura 1: Diagrama da arquitetura do sistema



Fonte: elaboração dos autores.

Conforme ilustrado na Figura 1, o módulo de avaliação automática está inserido no componente aplicação avaliadora. Esse componente encontra-se isolado, rodando em um servidor diferente e recebendo requisições por meio do protocolo Hypertext Transfer Protocol (HTTP). Essa estrutura evita que o processo de correção impacte na performance do sistema bem como que possíveis ataques consigam derrubar o sistema ou roubar dados da base de dados do sistema por meio da submissão de código malicioso pelo módulo aluno.

Tanto a aplicação *web* como a aplicação corretora encontram-se hospedadas em servidores de um serviço *web* de aluguel de máquinas na nuvem, com a tecnologia Jelastic de escalonamento automático, permitindo que o sistema conte com recursos de processamento adequados à demanda ou diminua esses recursos em um período de baixa atividade.

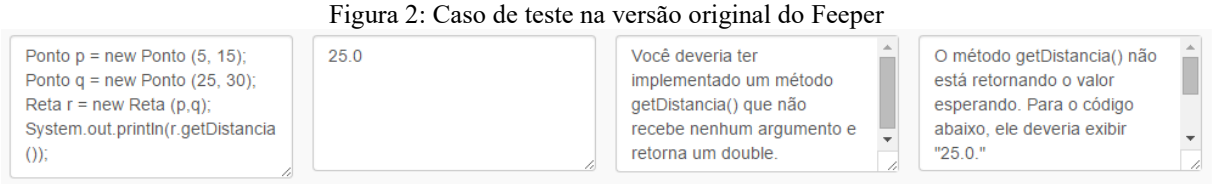
4.2 Funcionalidade de cadastro de exercícios

As alterações na funcionalidade cadastro de exercícios visam diminuir a carga de trabalho do professor, automatizando parcialmente e, dessa forma, agilizando o processo de cadastro de casos de teste. O primeiro passo no aprimoramento dessa funcionalidade foi a implementação de um mecanismo que possibilite ao professor carregar classes ou interfaces para o sistema. Com esse mecanismo, tornou-se possível a disponibilização de classes auxiliares, que poderão ser automaticamente incluídas na solução do aluno e personalizadas por ele, conforme necessário. Esse mecanismo também possibilita que o professor carregue classes ou interfaces que lhe auxiliarão no cadastro de casos de teste, deixando-as invisíveis para os alunos.

Com o intuito de agilizar o cadastro de casos de teste e, consequentemente, o cadastro dos exercícios, foi criado um mecanismo que gera automaticamente casos de teste. Esses casos de teste verificam se a solução do aluno atende a todos os requisitos de assinatura definidos no exercício. O mecanismo produz casos de teste que testam a assinatura de construtores, métodos de acesso, de modificação de atributos e demais métodos das classes da solução do aluno, utilizando como base uma classe ou interface carregada pelo professor. Ele gera casos de teste que verificam se a classe encontrada na solução do aluno apresenta a mesma estrutura da classe carregada pelo professor.

A Figura 2 ilustra a estrutura do caso de teste na versão do Feeper, desenvolvida por Alves e Jaques [7]. O caso de teste era composto pelas mensagens de *feedback*, pelas instruções de teste, que consistiam no código Java a ser executado para testar a solução do estudante, e pelo resultado esperado. O programa comparava a saída desejada com a saída do programa do estudante.

Figura 2: Caso de teste na versão original do Feeper



Fonte: Alves e Jaques, 2014 [7].

Para tornar o cadastro de dados de teste menos dependente da linguagem de programação utilizada e facilitar a execução de várias validações em cada caso de teste, a estrutura do caso de teste foi alterada. Ele passou a ser composto pelas mensagens de *feedback* e por passos estruturados, divididos em cinco tipos:

a) atribuição: executa uma atribuição a uma variável. A atribuição pode envolver um valor literal (tipo primitivo), uma instanciação de novo objeto, o retorno de uma chamada de método, ou o resultado de uma operação lógica ou aritmética;

b) execução: executa uma instrução sem guardar resultado. A instrução geralmente é a chamada de método de um objeto previamente instanciado;

c) verificação: operação utilizada para verificar se a solução atingiu um determinado objetivo por meio de uma comparação de dois valores. Essa instrução é capaz de comparar: duas variáveis; uma variável e um valor literal; uma variável e o retorno da chamada de método de um objeto previamente instanciado; um valor literal e o retorno da chamada de método de um objeto

previamente instanciado; uma variável e dados exibidos na tela; um valor literal e dados exibidos na tela;

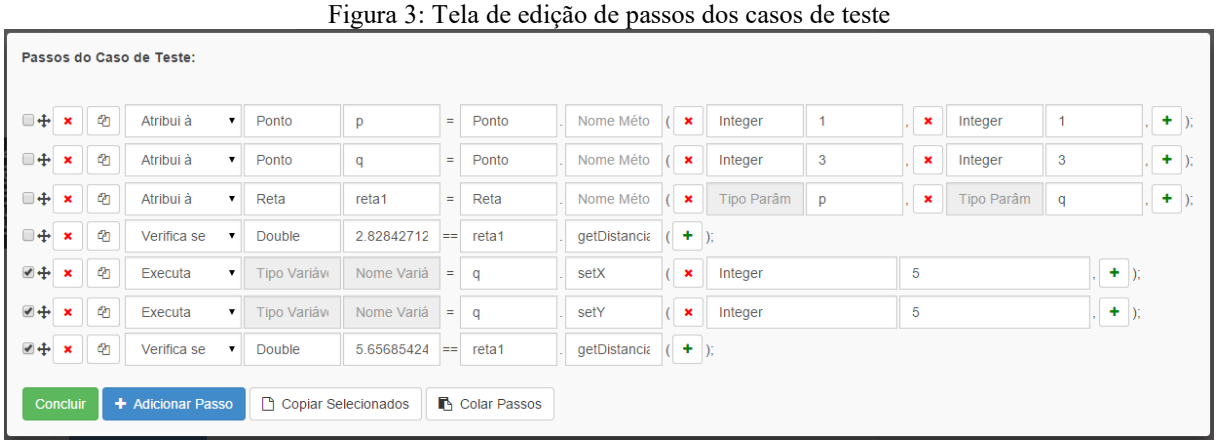
d) abertura de laço: operação utilizada para execução de laços de repetição no caso de teste. Com base em um critério selecionado, compara duas instruções e executa o laço enquanto o critério for verdadeiro. Essa instrução é capaz de utilizar como critério de comparação os mesmos critérios da operação verificação;

e) fechamento de laço: operação que determina o fim de um laço, sendo possível que um caso de teste tenha laços com outros laços internos.

A estrutura de passos descrita, além de facilitar o processo de cadastro dos casos de teste, estabelece uma estrutura de dados genérica menos dependente da linguagem de programação utilizada. Isso permite que a ferramenta seja empregada para correção automática de soluções codificadas em diferentes linguagens de programação. Isso é possível porque essa estrutura de dados é repassada ao módulo de correção automática, sendo ele o responsável pela interpretação dos dados e montagem das instruções de teste na linguagem de programação utilizada.

A Figura 3 ilustra a tela de edição dos passos dos casos de teste, na qual foram implementadas opções de autocompletar nos campos de preenchimento dos dados. Além disso, o trabalho proposto, utilizando como base uma classe ou interface carregada pelo professor, realiza a inclusão automática da lista de parâmetros de um método assim que esse método é inserido no campo que define o método a ser executado. Desse modo, o professor não precisará incluir os parâmetros e apenas precisará determinar o valor desse parâmetro na chamada do método.

Figura 3: Tela de edição de passos dos casos de teste



Fonte: captura de tela do Feeper.

Também foram implementadas opções como copiar e colar casos de teste, reordenar casos de teste, duplicar casos de teste, copiar e colar passos, reordenar passos e duplicar passos.

Uma alteração efetuada na tela de cadastro de exercícios, que melhorou seu desempenho e usabilidade, foi a utilização de um *Controller* construído com o *framework* AngularJS. Esse *framework* permite uma abordagem distinta da tradicional arquitetura Model-View-Controller (MVC), pois transfere a camada controladora para o lado do cliente, enquanto que no MVC tradicional essa camada encontra-se no lado do servidor.

Essa abordagem permite que as ações realizadas pelo usuário sejam processadas em seu próprio navegador, reduzindo assim o número de requisições ao servidor e o tempo de espera de atualização da página. Com a utilização dessa abordagem, essa tela comunica-se com o servidor apenas para carregar sua estrutura inicial, recuperar e persistir os dados na base de dados.

4.3 Módulo de avaliação automática

Como mencionado na seção anterior, o módulo de avaliação automática da versão original do Feeper somente suportava a validação de dados que eram passados a uma *stream* de saída. Essa limitação estava presente, pois, segundo Alves e Jaques [7], a versão original do Feeper utilizava o Juiz Online para realizar a avaliação.

Para a proposta deste trabalho, optou-se por desenvolver um módulo de avaliação próprio. Essa opção foi escolhida, principalmente, para fornecer um *feedback* mais detalhado aos alunos e dar suporte às validações múltiplas em cada caso de teste. O módulo implementado, além dos erros de compilação e da mensagem definida pelo professor, permitiu a apresentação de uma representação exata do problema ocorrido durante a avaliação dinâmica (ver seção 4.3.1) e de alertas provenientes da análise estática do código (ver seção 4.3.2).

A Figura 4 ilustra um *feedback* gerado pelo módulo de avaliação e apresentado ao aluno no qual consta uma mensagem de erro, um alerta indicativo da possível causa do erro e um alerta de problemas na qualidade do código.

Figura 4: Erros e alertas apresentados aos alunos

	Linha	Mensagem
✖ Erro		O método toString da classe Ponto não está retornando o resultado no formato (x, y), por exemplo (2, 4). Verifique este método, certifique de inserir um espaço entre a vírgula e o valor de y! Mensagem do Sistema expected:<[(45, 78)]> but was:<[Ponto@7ab2bfe1]>
⚠ Alerta	36	Mensagem do Sistema A classe Ponto define tostring();mas deveria ser toString(), a letra S deve ser maiúscula
⚠ Alerta		Mensagem do Sistema A propriedade: Ponto.z foi declarada mas nunca foi utilizada, considere a possibilidade de removê-la

Fonte: captura de tela do Feeper.

Além das mudanças no *feedback* gerado, o módulo de avaliação foi desacoplado do sistema acessado pelos usuários. O trabalho proposto transformou esse módulo em um serviço executado em uma aplicação isolada e acionada por requisições assíncronas por meio do protocolo de comunicação HTTP. A utilização dessa abordagem reduz o acoplamento e aumenta a coesão da aplicação. Além disso, deve-se considerar que o processo de avaliação automática consome muitos recursos do servidor, o que prejudicava o desempenho da aplicação na versão original do Feeper, algo que não ocorre com a estrutura utilizada neste trabalho.

Outra vantagem dessa estrutura é que, em conjunto com os casos de teste com passos estruturados, tornou-se mais fácil a adaptação da ferramenta para tornar possível a sua utilização com outras linguagens de programação. Isso se deve ao fato que os dados estruturados são repassados ao módulo corretor no formato XML, e cabe a esse as tarefas de interpretar os dados, preparar o ambiente para os testes, montar as instruções de teste na estrutura da linguagem de programação utilizada e executar os testes com todos os recursos tecnológicos necessários.

4.3.1 Avaliação dinâmica

Para melhorar o *feedback* provido pela ferramenta ao aluno, além das mensagens personalizadas pelo professor, o módulo de correção passou a exibir uma representação exata dos erros que ocorrem durante a avaliação dinâmica. Desse modo, o Feeper passou a fornecer evidências concretas sobre os problemas encontrados na solução do aluno.

Na versão do Feeper desenvolvida por Alves e Jaques [7], para os casos em que a solução do aluno não apresenta o resultado esperado, na avaliação dinâmica, somente a mensagem definida pelo professor era apresentada. Na versão gerada pela proposta deste trabalho, é apresentada a mensagem definida pelo professor seguida da representação exata do erro gerada pelo sistema. Essa representação pode ser a descrição de um erro em tempo de execução na solução submetida ou a comparação do resultado obtido com o resultado esperado.

A apresentação dessa representação dos erros tornou-se possível porque o módulo de avaliação implementado por este trabalho converte cada caso de teste cadastrado pelo professor em um caso de teste do *framework* JUnit. Após a criação dos casos de teste, eles são executados, e o resultado da execução é apresentado ao aluno.

As mensagens personalizadas têm um caráter pedagógico e visam esclarecer dúvidas relacionadas ao conteúdo ministrado em aula, sem se fixar no conteúdo técnico dos erros que podem ocorrer durante o processo de correção. A combinação de mensagens definidas pelo professor e das mensagens geradas pelo JUnit auxilia o aluno a solucionar os problemas do código submetido, pois fornece um retorno focado no conteúdo ministrado em aula e ao mesmo tempo objetivo, mostrando a descrição exata do problema. Como exemplo dessas descrições, destacam-se as comparações entre o valor esperado e o valor obtido e a descrição e localização de exceções ocorridas na solução do aluno durante a execução dos testes.

4.3.2 Análise estática

O módulo de avaliação automática implementado pela proposta deste trabalho também realiza a análise estática da solução do aluno, enriquecendo ainda mais o retorno provido. Essa análise executa a varredura da solução do estudante em busca de más práticas de programação, potenciais erros de programação e padrões de código que geralmente resultam em um comportamento inesperado da solução.

Entre os problemas de qualidade mais recorrentes entre os alunos de programação, e que são detectados pela análise estática, pode-se destacar o desrespeito às convenções de nomenclatura de classes, métodos e variáveis, a instanciação de variáveis nunca utilizadas e o encapsulamento de exceções sem tratamento.

A análise estática também emite alertas que podem auxiliar o aluno a encontrar erros durante o processo de correção da solução. Entre esses alertas estão o alerta para leitura de variáveis não instanciadas, comparação incorreta de dados, conversões de dados impossíveis de serem efetuadas e recursividade infinita ou laços infinitos.

Para realizar a análise estática, o módulo de avaliação utiliza a ferramenta FindBugs [24]. A solução do aluno é considerada correta caso atinja os objetivos definidos na especificação do exercício e validados na avaliação dinâmica. No entanto, o sistema emitirá alertas referentes aos problemas de qualidade e mostrará uma mensagem convidando o aluno a resolver esses problemas. Os alertas emitidos também estarão visíveis ao professor e caberá a ele definir o impacto dos problemas sobre a nota do aluno.

4.4 Funcionalidade de auxílio à detecção de plágio

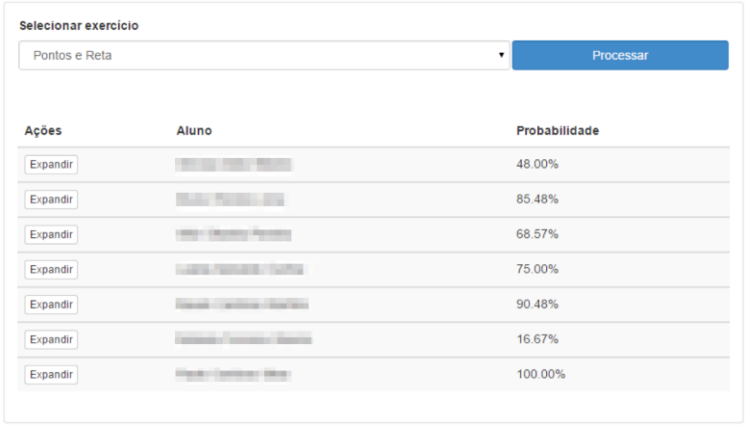
Com o intuito de auxiliar o professor na identificação de plágios entre as soluções dos alunos, foi implementada uma funcionalidade que analisa as soluções e calcula o percentual de similaridade entre elas. Para calcular a similaridade entre as soluções, foi utilizada a ferramenta Plaggie [25].

O Plaggie consiste em uma aplicação Java, desenvolvida por pesquisadores da Helsinki University of Technology, com o propósito de auxiliar na detecção de plágio nos exercícios aplicados aos alunos de programação. Entre as principais vantagens da utilização da ferramenta, destaca-se o fato de ela realizar a análise sem a necessidade de enviar as soluções para um serviço na internet e de ter código aberto [25].

A busca por plágio em exercícios de programação não pode consistir simplesmente em uma análise textual, pois desse modo mudanças simples no código seriam suficientes para mascarar casos de plágio. O Plaggie suporta a análise de vários arquivos em cada solução e realiza a busca utilizando o algoritmo Greedy String Tiling, proposto por Michael Wise. Esse algoritmo converte os arquivos das soluções em uma sequência de *tokens*, cada *token* conterá uma linha do arquivo. Após isso, essas sequências de *tokens* são comparadas aos pares para determinar a semelhança de cada par. Durante cada comparação, o algoritmo tenta cobrir um *token* com partes do outro. A percentagem do primeiro *token* de que pode ser coberta por partes do segundo *token* representa a similaridade entre eles [26].

A Figura 5 ilustra a tela de apresentação dos resultados dessa análise. Nessa tela, é possível analisar o percentual de similaridade entre as soluções e comparar as classes das soluções, facilitando assim a identificação de possíveis plágios.

Figura 5: Apresentação dos resultados da análise de plágio no Feeper



Fonte: captura de tela do Feeper.
Nota: os nomes dos estudantes foram ocultados para manter o anonimato.

5 Avaliação

Para evidenciar e analisar os impactos das mudanças propostas e realizadas no Feeper, foi realizada uma avaliação que contou com a participação de uma amostra por conveniência de treze alunos de graduação e pós-graduação voluntários dos cursos de tecnologia da informação de uma universidade sem fins lucrativos da Grande Porto Alegre. Os estudantes foram recrutados por meio de uma mensagem de divulgação na lista de discussão dos cursos de graduação em informática da universidade. Na avaliação, a versão estendida do Feeper foi utilizada pelos alunos voluntários durante três semanas.

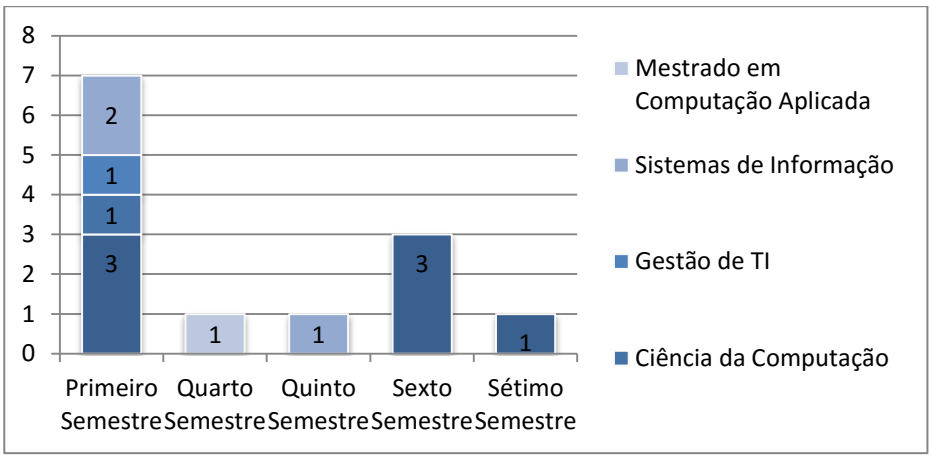
A cada semana, foi disponibilizado um projeto de programação no Feeper para os alunos desenvolverem e submeterem a solução. Totalizando, três projetos de programação foram disponibilizados durante a avaliação. Os alunos implementaram seus projetos de programação em Java, uma vez que essa é a linguagem de programação adotada pelos cursos de graduação em informática da universidade. Foi selecionado um professor voluntário para acompanhar a avaliação, que recebeu a tarefa de analisar as funcionalidades implementadas e incorporadas ao Feeper, bem como de acompanhar a aplicação dos exercícios e analisar o desempenho dos alunos.

Inicialmente, foi aplicado um questionário aos alunos participantes, mapeando seu perfil acadêmico e grau de conhecimento em programação. Depois disso, os alunos utilizaram a ferramenta durante um período de três semanas, e todo o processo foi acompanhado pelo professor voluntário. No fim desse período, foi aplicado um novo questionário para os alunos e outro para o professor com o objetivo de verificar a percepção deles sobre a ferramenta.

O questionário aplicado aos alunos no início da avaliação serviu para identificar o aluno. Os estudantes informaram a idade, o curso, os níveis que estavam cursando, se tinham conhecimento prévio em programação (anterior à entrada na universidade) e se têm experiência profissional em programação. Caso o aluno tivesse conhecimento prévio ou experiência, era verificado em quais linguagens de programação. A Figura 6 apresenta a distribuição por curso e semestre dos alunos que participaram da avaliação.

De acordo com as respostas do questionário, foi possível observar que os alunos que participaram da avaliação estavam na faixa de idade entre 18 e 34 anos. Foi possível identificar também que somente um aluno ainda não estudara e nem trabalhara com programação, três alunos ainda não tinham estudado programação e cinco alunos ainda não tinham trabalhado com programação. Todos os cinco alunos que ainda não tinham trabalhado com programação estavam no primeiro semestre do curso.

Figura 6: Distribuição dos alunos por semestre e curso



Fonte: elaboração dos autores com base no questionário dado aos alunos.

Após os alunos responderem ao questionário, foi iniciada a utilização da ferramenta. Esse processo teve a duração de três semanas, e durante esse período foi liberado um exercício de programação a cada semana. Os exercícios aplicados abordaram conceitos de programação ministrados nas disciplinas de programação de primeiro semestre da Unisinos. Os conceitos abordados pelos exercícios foram orientação a objetos, listas, *arrays* e conceitos de herança e polimorfismo.

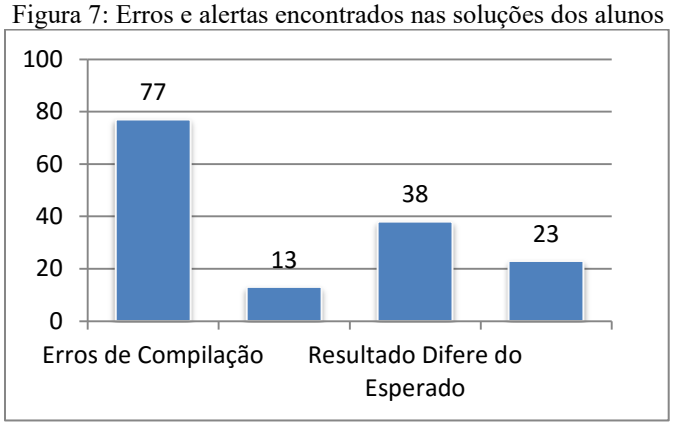
Entre os treze alunos voluntários que responderam ao questionário inicial, nove chegaram a utilizar o Feeper, com o qual puderam resolver os exercícios e trocar mensagens com o professor, visando resolver suas dúvidas. O Feeper também permite que o professor troque mensagens com o aluno, caso identifique que ele está tendo dificuldade. Durante essa avaliação, houve quatorze interações entre professor e aluno, das quais 75% foram motivadas por dúvidas nos exercícios e quanto ao uso da ferramenta; as demais 35% foram motivadas por erros de compilação.

Ao final da avaliação, dentre os nove alunos que utilizaram o Feeper, obteve-se um total de 21 resoluções de exercício bem sucedidas, três mal sucedidas e três ocorrências de exercícios sem tentativa de solução. Durante a avaliação, os alunos submeteram 153 tentativas de solução, totalizando uma média de aproximadamente seis tentativas para cada aluno por exercício. Dessas tentativas, 77 apresentaram erros de compilação, treze tinham diferenças em relação à especificação do exercício, 38 apresentaram um resultado diferente do esperado na avaliação dinâmica, quatro não tinham erros, mas apresentavam problemas de qualidade, e 21 revelaram-se soluções sem erros ou problemas de qualidade.

Com base nos dados descritos, é possível perceber que os problemas mais impactantes nas soluções foram os erros de compilação e casos em que a solução apresenta um resultado diferente do esperado durante os testes. O trabalho proposto aprimorou o *feedback* provido pelo Feeper quando a solução produz um resultado diferente do esperado ou não está em conformidade com a especificação do exercício, pois tornou possível a apresentação do resultado esperado comparado com o resultado obtido, a descrição e localização de exceções ocorridas durante a validação, e apresentou alertas provenientes da análise estática que apresentam possíveis causas para problemas encontrados na solução dos alunos.

A Figura 7 apresenta o total de mensagens geradas pelo módulo de avaliação e apresentadas pelo Feeper aos alunos. A primeira coluna representa o número de erros de compilação nas soluções dos alunos, a segunda representa o número de erros resultantes da não conformidade da solução com a especificação do exercício, a terceira coluna representa o número de resultados produzidos pelas soluções e que diferem do especificado no exercício, e a quarta coluna representa o número de mensagens provenientes da análise estática das soluções.

Figura 7: Erros e alertas encontrados nas soluções dos alunos



Fonte: base de dados do Feeper.

Ao fim do período da avaliação, foi aplicado um questionário para os alunos e outro para o professor com o objetivo de avaliar a nova versão do Feeper na visão deles. As perguntas desses questionários foram montadas utilizando a escala de Likert de concordância, com alternativas com peso de um a cinco, além de um campo aberto para críticas e sugestões. Os dados foram analisados com a medida por moda (Mo) ou a resposta mais frequente, assim como a média (μ).

Dentre os nove alunos que utilizaram a nova versão do Feeper, oito responderam ao questionário da pesquisa. A grande maioria dos alunos acredita que o uso de ferramentas que apoiam o aprendizado da programação tem muita utilidade (Mo=5; μ=4,9). Além disso, a maioria dos alunos acredita que o Feeper facilitaria o seu aprendizado nas cadeiras de programação (Mo=5; μ=4,7). Em um contexto geral, a utilidade do Feeper envolvendo todas suas funcionalidades foi avaliada como boa (Mo=4; μ=4,5).

De acordo com a pesquisa, a maioria dos alunos acredita que o *feedback* provido pelo Feeper auxiliou na resolução dos exercícios (Mo=4; μ=4,1). Os alunos julgaram como muito útil as mensagens que o trabalho proposto adicionou no *feedback* provido pelo Feeper (Mo=4; μ=4,4). A qualidade do *feedback* provido pelo Feeper ao aluno foi avaliada de qualidade intermediária, tendendo para bom (Mo=4; μ=3,9). Assim, a proposta deste trabalho conseguiu um aumento de três décimos de ponto na nota do *feedback* provido pelo Feeper, quando comparada com a nota obtida na avaliação conduzida por Alves e Jaques [7] (Mo=4; μ=3,6).

No campo de críticas e sugestões, os alunos solicitaram melhorias no editor de código e nas mensagens do compilador. No editor de código, foi solicitada a compilação das classes em tempo de codificação, evitando assim que o aluno envie o exercício para correção com erros de compilação. Nas mensagens do compilador, foi solicitada a análise da possibilidade de traduzi-las para a língua portuguesa.

No questionário aplicado ao professor, foi solicitado que ele avaliasse as funcionalidades implementadas pelo trabalho proposto na escala Likert, também lhe foi disponibilizado um campo para justificar as notas e apresentar críticas e oportunidades de melhoria. O professor julgou importante a utilização de ferramentas que apoiem o ensino da programação e atribuiu nota quatro para o Feeper. O professor enfatizou a importância da implementação de técnicas de gamificação na ferramenta, permitindo aos alunos que aumentem sua pontuação ao ajudar os colegas a resolver os exercícios.

Para a funcionalidade de cadastro de casos de teste, o professor atribuiu nota três. Como justificativa para a nota, o professor julgou útil a funcionalidade que automatiza a geração dos casos de teste que validam a assinatura das classes, porém relatou que encontrou dificuldades de compreensão em relação ao seu funcionamento.

Em relação ao *feedback* provido pelo Feeper ao aluno, o professor atribuiu nota quatro. O professor sugeriu a implementação de uma funcionalidade de análise dos erros submetidos pelos alunos e a apresentação de uma lista de colegas que já conseguiram solucionar tal erro. Desse modo, os alunos poderão interagir e buscar resolver os problemas nas soluções.

Para a funcionalidade de auxílio de detecção ao plágio, o professor atribuiu nota três. A funcionalidade, em alguns casos, calculou a similaridade das soluções como 100%, sendo que as soluções dos alunos não eram

completamente iguais; esses casos são conhecidos como falso positivo. Também é possível melhorar a comparação de classes das soluções. A comparação implementada apresentou algumas deficiências na comparação de classe pequenas com partes muito similares.

6 Conclusão

O presente trabalho objetivou propor melhorias no AVA Feeper, desenvolvido por Alves e Jaques [7], a partir da reformulação da avaliação dinâmica e da combinação dessa avaliação com técnicas de análise estática de código, de modo a enriquecer o *feedback* provido ao aluno. Para o professor, o trabalho propôs um mecanismo capaz de automatizar a criação de parte dos dados de teste, tornando os casos de teste menos dependentes da linguagem de programação utilizada no exercício e aumentando a portabilidade da ferramenta. Ademais, foi proposto um mecanismo capaz de evidenciar a ocorrência de plágio entre as soluções dos alunos.

Assim, as mensagens resultantes da avaliação dinâmica, além da mensagem definida pelo professor, passaram a contar também com uma mensagem gerada pelo sistema. Essa mensagem podia representar o resultado obtido em comparação ao esperado ou a representação e localização do erro ocorrido durante a execução dos testes. Em paralelo a isso, as mensagens resultantes da análise estática apresentam também possíveis causas para os erros encontrados e indicadores de qualidade da solução do aluno.

Para o professor, foi implementado um mecanismo que, com base em uma estrutura definida por ele, automatizou a geração de casos de teste que validam a estrutura da solução do aluno. Também foi implementado um mecanismo capaz de analisar a similaridade das soluções dos alunos evidenciando para o professor a ocorrência de plágio.

Este trabalho foi integrado ao ambiente virtual de aprendizagem Feeper, desenvolvido para apoiar as disciplinas de programação, e que contém funcionalidades para cadastro de projetos de programação e submissão de soluções pelos alunos. Uma avaliação deste estudo foi realizada com uma amostra por conveniência de treze estudantes de cursos de graduação em informática, que usaram o Feeper para desenvolver projetos de programação durante três semanas. Após as três semanas, foi aplicado um questionário com questões fechadas cujas respostas seguiam a escala Likert de concordância. A pesquisa indicou que as melhorias implementadas no Feeper pelo trabalho proposto aumentaram a capacidade da ferramenta de auxiliar os alunos na resolução dos exercícios, e que de fato elas foram úteis para os utilizadores da ferramenta.

Com a avaliação realizada, também foram identificadas oportunidades de melhorias da ferramenta. Uma das oportunidades de melhoria identificada é a implementação de um mecanismo de compilação das classes do aluno em tempo de codificação. Essa funcionalidade é importante visto que os erros de compilação representam uma grande parcela dos erros encontrados nas soluções dos alunos. Outra oportunidade de melhoria, que também foi identificada por Alves e Jaques [7], é a implementação de técnicas de gamificação, que permite que o sistema e os alunos interajam, procurando resolver suas dúvidas, seria, então, atribuída uma pontuação para o aluno a cada vez que ele auxiliasse um colega.

7 Agradecimentos

Esta pesquisa foi suportada financeiramente pelo órgão brasileiro de fomento à pesquisa CNPq.

8 Referências

[1] LESGOLD, A. M. The nature and methods of learning by doing. *American Psychologist*, American Psychological Association, Wahington, v. 56, n. 11, p. 964-973, 2001.

[2] YUSOF, N.; ZIN, N. A. M.; ADNAN, N. S. Java Programming Assessment Tool for Assignment Module in Moodle E-learning System. *Procedia - Social and Behavioral Sciences*, s. l., v. 56, p. 767-773, 2012.

[3] BRITO, S. R. et al. Computer Supported Collaborative Learning for helping novice students acquire self-regulated problem-solving skills in computer programming. In: WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING, AND APPLIED COMPUTING, 7., 2011, La Plata. *Proceedings...* La Plata: Universidad Nacional de La Plata, 2011. p. 65-73.

[4] ROCHA, P. S. et al. Ensino e aprendizagem de programação: análise da aplicação de proposta metodológica baseada no sistema personalizado de ensino. *Renote Revista Novas Tecnologias na Educação*, Porto Alegre, v. 8, n. 3, 2010. Disponível em: <http://www.seer.ufrgs.br/index.php/renote/article/view/18061>. Acesso em: 3 mai. 2016.

[5] CHAVES, J. O. M. et al. Integrando Moodle e Juízes Online no apoio a atividades de programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 24, 2013, Campinas. *Anais...* Campinas: Universidade Estadual de Campinas, 2013. p. 244254.

[6] SIROTHEAU, S. et al. Aprendizagem de iniciantes em algoritmos e programação: foco nas competências de autoavaliação. In: WORKSHOP DE INFORMÁTICA NA ESCOLA, 17, 2011, Aracaju. *Anais...* Aracaju: Universidade Federal de Alagoas, 2011. p. 750-759.

[7] ALVES, F. P.; JAQUES, P. Um ambiente virtual com *feedback* personalizado para apoio a disciplinas de programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 25, 2014, Dourados. *Anais...* Dourados: Universidade Federal da Grande Dourados, 2014. p. 1078-1082.

[8] MOREIRA, M. P.; FAVERO, E. L. Um ambiente para ensino de programação com *feedback* sutomático de exercícios. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 17, 2009, Bento Gonçalves. *Anais...* Bento Gonçalves: Universidade Federal do Rio Grande do Sul, 2009. p. 429-438.

[9] GOMES, A.; HENRIQUES, J.; MENDES, A. Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologia*, Monte da Caparica, v. 1, n. 1, p. 93-103, 2008.

[10] PAULA, L. Q. de; PIVA JR., D.; FREITAS, R. L. A importância da leitura e da abstração do problema no processo de formação do raciocínio lógico-abstrato em alunos de computação. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 17, 2009, Bento Gonçalves. *Anais...* Bento Gonçalves: Universidade Federal do Rio Grande do Sul, 2009. p. 2687-2690.

[11] MERRILL, D. et al. Tutoring: guided learning by doing. *Cognition and Instruction*, Santa Monica, v. 13, n. 3, p. 315-372, 1995.

[12] KINNUNEN, P.; MALMI, L. Why students drop out CS1 course?. In: INTERNATIONAL WORKSHOP ON COMPUTING EDUCATION RESEARCH, 2., 2006, New York. *Proceedings...* New York: ACM Press, 2006. p. 97-108.

[13] GIRAFFA, L. M. M.; MÓRA, M. C. Evasão na disciplina de algoritmo e programação: um estudo a partir dos fatores intervenientes na perspectiva do aluno. In: CONFERÊNCIA LATINO AMERICANA SOBRE O ABANDONO NA EDUCAÇÃO SUPERIOR, 3, 2013, Cidade do México. *Proceedings...* Cidade do México: Universidade Nacional Autônoma do México, 2013. Disponível em: <http://www.alfaguia.org/www-alfa/images/ponencias/clabesIII/LT_1/ponencia_completa_136.pdf>. Acesso em: 3 mai. 2016.

[14] GONZÁLEZ, S. *Procuram-se profissionais de TI*. 2013. Disponível em: <http://www.brasscom.org.br/brasscom/Portugues/detNoticia.php?codArea=2&codCategoria=26&codNoticia=400>. Acesso em: 9 set. 2014.

[15] WOOLF, B. P. *Building Intelligent Interactive Tutors: Student-centered strategies for revolutionizing e-learning*. Burlington: Elsevier, 2009.

[16] BECK, K.; GAMMA, E. *JUnit - About*. 2014. Disponível em: <http://junit.org/index.html>. Acesso em: 10 out. 2014.

[17] EDWARDS, S. H.; QUINONES, M. A. P. Web-CAT. *ACM SIGCSE Bulletin*, New York, v. 40, n. 3, p. 328, 2008.

[18] JOY, M.; GRIFFITHS, N.; BOYATT, R. The boss online submission and assessment system. *Journal on Educational Resources in Computing*, New York, v. 5, n. 3, p. 2-28, 2005.

[19] MOTA, M. P.; PEREIRA, L. W. K.; FAVERO, E. L. Javatool: uma ferramenta para ensino de programação. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 17, 2009, Bento Gonçalves, *Anais...* Bento Gonçalves: Universidade Federal do Rio Grande do Sul, 2009. p. 127-136.

[20] EDWARDS, S. H. Teaching software testing. In: ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, 18, 2003, New York. *Proceedings...* New York: ACM Press, 2003. p. 318-319.

[21] JOHNSON, C. SpecCheck. In: ACM ANNUAL CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 17, 2012, New York. *Proceedings...* New York: ACM Press, 2012. p. 186-191.

[22] WERAGAMA, D.; REYE, J. Analysing student programs in the PHP Intelligent Tutoring System. *International Journal of Artificial Intelligence in Education*, New York, v. 24, n. 2, p. 162-188, 2014.

[23] ALLOWATT, A.; EDWARDS, S. IDE Support for test-driven development and automated grading in both Java and C++. In: OOPSLA WORKSHOP ON ECLIPSE TECHNOLOGY EXCHANGE, 1., 2005, New York. *Proceedings...* New York: ACM Press, 2005. p. 100-104.

[24] HOVEMEYER, D. H.; PUGH, W. W. *FindBugs™ Manual*. University of Maryland, 2014. Disponível em: <http://findbugs.sourceforge.net/manual/index.html>. Acesso em: 16 out. 2014.

[25] AHTIAINEN, A.; SURAKKA, S.; RAHIKAINEN, M. Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises. In: BALTIC SEA CONFERENCE ON COMPUTING EDUCATION RESEARCH, 7, 2006, New York. *Proceedings...* New York: ACM, 2006. p. 141-142.

[26] LUKE, D. et al. Software plagiarism detection techniques: a comparative study. *International Journal of Computer Science and Information Technologies*, Duluth, v. 5, n. 4, p. 5020-5024, 2014.