UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GILVANI SCHNEIDER

COMBINANDO TÉCNICAS DE ANÁLISE ESTÁTICA E AVALIAÇÃO DINÂMICA PARA AVALIAÇÃO DE CÓDIGO EM AMBIENTES VIRTUAIS DE APRENDIZAGEM DE PROGRAMAÇÃO

São Leopoldo
2015

GILVANI SCHNEIDER

COMBINANDO TÉCNICAS DE ANÁLISE ESTÁTICA E AVALIAÇÃO DINÂMICA PARA AVALIAÇÃO DE CÓDIGO EM AMBIENTES VIRTUAIS DE APRENDIZAGEM DE PROGRAMAÇÃO

Artigo apresentado como requisito parcial para obtenção do título de Tecnólogo em 2015, pelo Curso de Análise e Desenvolvimento de Sistemas da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Profa. Dra. Patrícia Augustin Jaques Maillard

São Leopoldo

2015

COMBINANDO TÉCNICAS DE ANÁLISE ESTÁTICA E AVALIAÇÃO DINÂMICA PARA AVALIAÇÃO DE CÓDIGO EM AMBIENTES VIRTUAIS DE APRENDIZAGEM DE PROGRAMAÇÃO

Gilvani Schneider*

Patrícia Augustin Jaques Maillard**

Resumo: O aprendizado da programação tem se revelado desafiador para a maioria dos alunos do ensino superior. A estratégia de ensino empregada é geralmente "*learning by doing*", envolvendo a aplicação de vários exercícios de programação. Estas atividades geralmente são facilitadas por ferramentas que possibilitam para os alunos a submissão da solução dos exercícios pela internet e automatizam o processo de correção. Porém, o *feedback* provido por estas ferramentas, muitas vezes, limita-se a certo ou errado, não fornecendo informações específicas que possam auxiliar o estudante a identificar o erro cometido e como solucioná-lo. O trabalho proposto apresenta um mecanismo de *feedback* aprimorado, que combina técnicas de análise estática e avaliação dinâmica de código, apresentando indicadores de qualidade da solução submetida, uma mensagem definida pelo professor e uma descrição técnica dos erros encontrados na solução. O trabalho também apresenta um mecanismo de cadastro de casos de teste pelo docente, que não requer muito conhecimento técnico, e é capaz de automatizar a geração de casos de teste que validam a assinatura das classes a serem submetidas, agilizando assim a realização desta atividade pelo professor. O trabalho proposto 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 do trabalho proposto foi realizada com uma amostra por conveniência de 13 estudantes de cursos de 3º grau em informática, que usaram Feeper para desenvolver projetos de programação por 3 semanas. Após as 3 semanas, foi aplicado um questionário com questões fechadas cujas respostas seguiam a escala Likert de concordância. Os resultados evidenciaram que as funcionalidades implementadas auxiliaram os estudantes na resolução dos exercícios.

Palavras-chave: Correção Automática. *Feedback* Instantâneo. Ensino de Programação. Avaliação Dinâmica. Análise Estática.

1 INTRODUÇÃO

As disciplinas de programação abordam um dos conceitos básicos da área de Tecnologia da Informação (TI), sendo ministradas geralmente no primeiro e segundo semestres dos cursos desta área. Tais disciplinas visam ensinar aos alunos a escrita

* Gilvani Schneider, Graduando em Análise e Desenvolvimento de Sistemas Unisinos.
E-mail: gilschneider90@gmail.com.
** Profa. Dra. Patrícia Augustin Jaques Maillard, Professor Adjunto do PIPCA/UNISINOS.
E-mail: pjaques@unisinos.br.

de instruções que possam ser interpretadas pelo computador e convertidas posteriormente em programas executáveis.

A dinâmica de ensino nestas disciplinas geralmente 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*". (BORGES, 2000). O nível de complexidade dos exercícios é incrementando gradualmente, de modo que não seja ignorada nenhuma etapa importante no aprendizado do aluno. (YUSOF; ZIN; ADNAN, 2012).

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 disto, temos uma menor disponibilidade de tempo para o professor interagir com os alunos e o aumento do prazo de correção dos exercícios. Com este aumento, 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. (BRITO et al., 2011).

Diante deste quadro, se torna necessária a abordagem e 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 e auxiliando 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 à distância. (ALVES; JAQUES, 2014; CHAVES et al., 2013; GOMES; HENRIQUES; MENDES, 2008; MOREIRA; FAVERO, 2009; ROCHA et al., 2010; SIROTHEAU et al., 2011; YUSOF; ZIN; ADNAN, 2012). AVAS são sistemas web que visam agilizar e facilitar a interação entre alunos e professores. O AVA permite que o professor poste conteúdos, exercícios, notas, notícias e afins. Para o aluno, permitem 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 (http://feeper.unisinos.br). Entre as principais e mais

importantes funcionalidades do Feeper estão a correção automática dos exercícios e o *feedback* personalizado e instantâneo para o aluno após a submissão da solução do exercício. (ALVES; JAQUES, 2014).

Segundo Alves e Jaques (2014), 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 do módulo 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 da 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 disto, foi implementado um mecanismo de correção próprio, passível de personalização e capaz de fornecer mais detalhes sobre os problemas encontrados na solução do aluno.

O *feedback* apresentado aos alunos foi aperfeiçoado, pois, além das mensagens definidas pelo professor, o trabalho proposto tornou 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.

O trabalho proposto é parte integrante dos esforços de aprimoramento do Feeper, visando transformá-lo em um Sistema Tutor Inteligente (STI). Dessa forma, além de novas funcionalidades, o trabalho proposto também implementou melhorias em funcionalidades existentes do sistema.

Como diferencial, o trabalho proposto apresenta um mecanismo que apresenta um *feedback* resultante da combinação de mensagens provenientes da análise estática e avaliação dinâmica das soluções dos alunos. Além disto, 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 desta atividade pelo professor, e um mecanismo que analisa as soluções dos alunos, apresentando o percentual de similaridade entre elas, facilitando assim a identificação de plágios entre estas soluções.

1.1 Objetivo

Este trabalho tem como objetivo principal estender o módulo de correção automática do ambiente de aprendizagem Feeper, com o objetivo de aprimorar o *feedback* provido pela ferramenta aos alunos, através da combinação da análise estática e da avaliação dinâmica das soluções. Esta combinação provê mais detalhes sobre os erros encontrados, aumentando a capacidade de auxílio, por parte da ferramenta, ao aluno na resolução dos exercícios.

Os objetivos secundários deste trabalho são estender a funcionalidade de cadastro de exercícios a fim de facilitar e agilizar o desenvolvimento desta atividade pelo professor. Mais especificamente, busca-se agilizar o cadastro dos casos de teste que irão validar as soluções submetidas pelos alunos e torná-los independentes da linguagem de programação utilizada no exercício. Além disso, buscou-se igualmente implementar uma funcionalidade que facilite a identificação de plágios entre as soluções dos alunos.

1.2 Organização do Texto

O restante do artigo está organizado da seguinte forma. A seção 2 apresenta os conceitos básicos necessários para a compreensão deste trabalho. A seção 3 apresenta os trabalhos relacionados, bem como uma tabela comparativa entre estes trabalhos e o trabalho proposto. A seção 4 apresenta o trabalho a desenvolvido. A seção 5 apresenta a avaliação e os resultados obtidos pelo trabalho. A seção 6 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 da 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 pretende se especializar, o entendimento da programação é sempre importante. (BRITO et al., 2011).

A programação é 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. Após isso, ao escrever o conjunto de regras, o estudante desenvolve o raciocínio lógico e conhecimentos matemáticos. (PAULA; PIVA JR; FREITAS, 2009).

A dinâmica geralmente utilizada no ensino da programação é a "*learning by doing*". Uma das maneiras mais eficientes de se adquirir conhecimento em novas áreas é através da resolução de problemas específicos da área. Nesta dinâmica, os alunos são desafiados a atingirem objetivos propostos pelo professor. Tais objetivos requerem a compreensão e aplicação dos conceitos ensinados em aula. (MERRILL et al., 1995).

Durante seus esforços para atingir os objetivos, o próprio aluno é capaz de identificar e corrigir fraquezas e falhas no seu aprendizado. Porém, esta 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, evitando assim sobrecargas de conteúdo e frustações para o aluno. (MERRILL et al., 1995).

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. (PAULA; PIVA JR; FREITAS, 2009). 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. (KINNUNEN; MALMI, 2006). Muitas universidades relatam um índice de desistência em torno de 30 a 40 por cento. No Brasil, essa realidade não é diferente, conforme apontado por Gomes, Henriques e Mendes (2008).

Segundo Giraffa e Móra (2013), o índice de desistência na disciplina de Programação I (1º semestre) em uma universidade privada de Porto Alegre no período de 2012 a 2013 girou em torno dos 38%. González (2013) coloca nos estados de Distrito Federal, São Paulo, Rio de Janeiro, Paraná, Rio Grande do Sul, Minas Gerais, Bahia e Pernambuco já no ano de 2014 haverá um déficit de 45 mil profissionais na área de TI.

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. Este déficit de profissionais faz com que empresas multinacionais tenham que repensar decisões de investir no mercado de TI brasileiro. Considerando que este mercado representa atualmente uma fatia de 4,5 por cento 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 (AVAs) têm sido empregados para apoiar as disciplinas de ensino de programação, sejam elas presenciais ou à distância. (ALVES; JAQUES, 2014; CHAVES et al., 2013; GOMES; HENRIQUES; MENDES, 2008; MOREIRA; FAVERO, 2009; ROCHA et al., 2010; SIROTHEAU et al., 2011; YUSOF; ZIN; ADNAN, 2012). AVAS são sistemas com acesso através da internet que visam agilizar e facilitar a interação entre alunos e professores. 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.

Por outro lado, os STIs empregam técnicas e ferramentas da Inteligência Artificial para prover assistência personalizada. Para tanto, um STI é capaz de identificar o nível conhecimento e estados afetivos do aluno, construindo um modelo de aluno. Com base neste modelo, o STI é capaz de propor exercícios que melhor se adaptam as necessidades de aprendizado do aluno, bem fornecer dicas e *feedback* (verificar se a solução dada é correta) sobre as soluções do estudante. (WOOLF, 2009).

Como citado anteriormente, atualmente o Feeper se caracteriza como um AVA. O trabalho proposto é parte integrante dos esforços desenvolvidos com o objetivo de transformar o Feeper em um STI. Este trabalho contribui para que o Feeper seja capaz de extrair informações mais precisas do conhecimento do aluno, a fim de prover assistência individualizada a este.

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. Esta validação geralmente ocorre através da aplicação de casos de teste no sistema alvo, validando as pós-condições e os resultados obtidos.

Caso de teste é nome dado a um conjunto de instruções elaboradas para testar um software. O caso de teste é composto basicamente pelos dados de entrada, as pré-condições de execução e as ações a serem executadas no sistema alvo, os resultados esperados e as 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 através 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.

2.4 Avaliação Dinâmica de Código

A avaliação dinâmica do código é a forma de avaliação comumente utilizada em AVAs ou STIs. Esta avaliação é possível através de componentes como Juízes Online, da implementação de mecanismos próprios de validação ou da utilização de *frameworks* de testes automatizados como o *JUnit.* (BECK; GAMMA, 2014).

Este tipo de avaliação tem como objetivo garantir que o código do aluno seja capaz de cumprir os objetivos propostos pelo professor no exercício. Para isto, é necessário que o código do aluno seja compilado, empacotado e executado, gerando os resultados corretos e atendendo as pós-condições estabelecidas. Este tipo de avaliação é amplamente utilizada em AVAs e STIs de programação, entre os quais podemos citar o WebCat (EDWARDS; PEREZ-QUINONES, 2008), o BOSS (JOY; GRIFFITHS; BOYATT, 2005) e o Javatool (MOTA; PEREIRA; FAVERO, 2008).

2.5 Análise Estática de Código

A análise estática do código não requer a sua execução. Este 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, entre as quais podemos citar o desrespeito aos padrões de nomenclatura de classes e variáveis, métodos muito complexos, variáveis não usadas, etc... 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...

Este tipo de análise é geralmente feita através da utilização de ferramentas que automatizam esta análise e retornam os problemas encontrados, entre as quais podemos destacar Checkstyle, PMD e FindBugs. Este tipo de análise é menos comum em AVAs ou STIs. Entre os principais sistemas que utilizam este tipo de verificação, podemos destacar o WebCat. (EDWARDS, 2003).

3 TRABALHOS RELACIONADOS

Com o objetivo reduzir a carga de trabalho do professor, Johnson (2012) desenvolveu o AVA SpecCheck. O SpecCheck suporta a avaliação dinâmica de código e gera os casos de teste automaticamente através de uma solução padrão fornecida pelo professor. Os dados de teste usados para geração dos casos de teste são fornecidos na solução padrão através de uma estrutura de anotações em forma de comentários. O SpecCheck foi utilizado durante quatro semestres em sua 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 (2014) 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 este 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 nesta estrutura requer muito conhecimento técnico, o que dificulta a realização desta tarefa pelo professor. Para tentar sanar este problema, os autores criaram um repositório de exercícios onde o professor pode selecionar o exercício a ser aplicado para os alunos, porém isto elimina a possibilidade de personalização dos exercícios pelo professor.

Moreira e Favero (2009) propõem uma ferramenta integrada ao Moodle que implementa uma metodologia de avaliação dos exercícios de programação através de regressão linear baseada em n-gramas. O objetivo desta abordagem é avaliar a solução do aluno através da estrutura e complexidade da solução. O diferencial desta 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 nova visando aumentar sua nota. É necessário prover 20 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 (2005) propõem uma abordagem onde, 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. No trabalho é desenvolvido um *plug-in* onde 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 (2014) desenvolveram o AVA Feeper. O Feeper permite que os alunos realizem a submissão de suas soluções de exercícios através da internet. O Feeper, com o auxílio da ferramenta CodeJudge categorizada como Juiz Online e incorporada ao Feeper, realiza a avaliação estática da solução e provê *feedback* ao aluno. O Feeper foi utilizado durante dois meses na Unisinos e permitiu um aumento do número de exercícios realizados pelos alunos, bem como uma visualização mais clara do andamento do aprendizado dos alunos. O Feeper permitiu também a diminuição da carga de trabalho do professor com a correção dos exercícios, porém gerou atividades completamente manuais e onerosas como o cadastro de casos de teste para a correção automática.

3.1 Tabela Comparativa

É apresentada a seguir uma tabela comparativa dos trabalhos relacionados com o trabalho proposto. A primeira coluna da tabela representa os critérios de comparação e cada uma das colunas a seguir corresponde ao trabalho citado na primeira linha. Os campos marcados com um (X) indicam que o trabalho citado atende o critério da linha correspondente. Caso o campo esteja vazio, o critério da linha correspondente não foi atendido.

Estímulo ao aluno para refinar sua solução e conhecimentos refere-se à capacidade do AVA ou STI de estimular o aluno a revisar aprimorar o código

submetido e consolidar seus conhecimentos. O trabalho de Moreira e Favero (2009) conseguiu estimular o aluno através da avaliação e atribuição de nota para a complexidade e a estrutura da solução do aluno. O trabalho proposto estimula o aluno através da apresentação de más práticas de programação empregadas e de indicadores de qualidade da solução submetida.

Correção automática dos exercícios com feedback instantâneo para o aluno, refere-se ao AVA ou STI utilizado ser capaz de avaliar a solução automaticamente e comunicar o resultado ao aluno. Todos os trabalhos relacionados conseguiram atingir este critério. O trabalho proposto também será capaz de prover *feedback* instantâneo após a submissão da solução.

Avaliação dinâmica do código do aluno refere-se à capacidade do AVA ou STI utilizado de executar o código do aluno e realizar sua validação através de objetivos propostos no exercício. O trabalho de Weragama e Reye (2014) conseguiu isto utilizando uma estrutura de correção criada pelos autores. O trabalho de Allowatt e Edwards (2005) empregou JUnit e o trabalho de Alves e Jaques (2014) usou o Juiz Online CodeJudge para essa tarefa. O trabalho proposto fará a verificação dinâmica com o auxílio do JUnit.

Análise estática do código do aluno refere-se à capacidade do AVA ou STI utilizado de verificar aspectos da estrutura e da qualidade do código do aluno. O trabalho de Moreira e Favero (2009) faz a análise estática através de uma estrutura de verificação construída pelos autores. O trabalho de Allowatt e Edwards (2005) utilizou as ferramentas CheckStyle e PMD para essa tarefa. O trabalho proposto fará a análise estática do código com o auxílio da ferramenta FindBugs.

Auxílio na detecção de plágios refere-se à capacidade do AVA ou STI de analisar as soluções dos alunos em determinado exercício verificando o grau de similaridade entre elas, identificando assim possíveis plágios, cabendo ao professor determinar se o plágio realmente aconteceu. O trabalho de Allowatt e Edwards (2005) permite a instalação de plug-ins que possuem esta funcionalidade. O trabalho proposto analisa a similaridade das soluções com o auxílio da ferramenta Plaggie.

Geração automática de testes de assinatura refere-se à capacidade do AVA ou STI utilizado de gerar automaticamente casos de teste, que verifiquem se a solução do aluno respeitou os requisitos de assinatura de classes definidos pelo professor. No trabalho proposto, basta que o professor carregue para o sistema uma interface das classes da solução para possibilitar a automatização da geração

destes testes. Os autores dos trabalhos relacionados não citaram a existência de funcionalidades similares em seus trabalhos.

Os diferenciais do trabalho proposto em relação aos demais trabalhos são a criação de um mecanismo que agiliza 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.

Quadro 1 - Critérios de Comparação dos Trabalhos

Critérios	(JOHNSON, 2012)	(WERAGAMA; REYE, 2014)	(MOREIRA; FAVERO, 2009)	(ALLOWATT; EDWARDS, 2005)	(ALVES; JAQUES, 2014)	Trabalho Proposto
Estímulo ao aluno para refinar sua solução e conhecimentos.			X			X
Feedback instantâneo para o aluno.	X	X	X	X	X	X
Avaliação dinâmica do código do aluno.		X		X	X	X
Análise estática do código do aluno.			X			X
Auxilio na detecção de plágios				X		X
Geração automática de testes de assinatura						X

4 TRABALHO PROPOSTO

O trabalho proposto busca aprimorar o *feedback* provido aos alunos pelo AVA Feeper, desenvolvido por Alves e Jaques (2014). O *feedback* provido pela ferramenta foi aprimorado através da reformulação da avaliação dinâmica e da combinação desta avaliação com técnicas de análise estática de código.

As mensagens resultantes da avaliação dinâmica passaram a contar com a mensagem personalizada pelo professor e uma mensagem gerada pelo sistema, que apresenta 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.

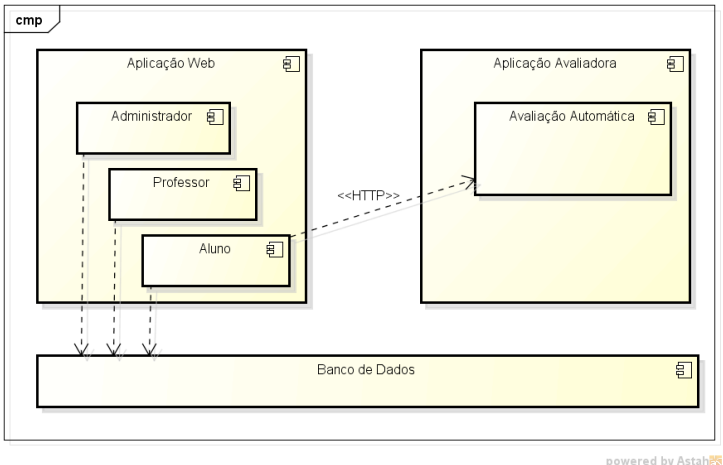
Além disto, o trabalho proposto busca estender a funcionalidade de cadastro de exercícios a fim de facilitar e agilizar o desenvolvimento desta atividade pelo professor e implementar uma funcionalidade que facilite a identificação de plágios entre as soluções dos alunos.

4.1 Arquitetura do Sistema

Conforme Alves e Jaques (2014), o Feeper foi originalmente desenvolvido como uma aplicação para a internet, dividida em três módulos controlados por usuários, sendo eles aluno, professor e administrador. O sistema conta também com um módulo utilizado no processo de avaliação automática dos exercícios, sendo este módulo controlado pelo próprio sistema. O trabalho proposto realizou alterações no módulo operado pelo professor, no módulo operado 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. Nesta estrutura, os módulos Administrador, Professor e Aluno estão inseridos no componente Aplicação Web, ambos com acesso ao banco de dados. O módulo de Avaliação Automática, por sua vez, está inserido na Aplicação Avaliadora, rodando em um servidor diferente e recebendo requisições através do protocolo *Hypertext Transfer Protocol* (HTTP).

Figura 1 - Diagrama da Arquitetura do Sistema



Fonte: Elaborado pelos autores (2015)

Tanto a aplicação Web, como a aplicação corretora, encontram-se hospedadas em servidores da Locaweb com a Tecnologia Jelastic de escalonamento automático, permitindo que o sistema conte com recursos de processamento adequados à demanda ou diminua estes recursos em um período de atividade.

4.2 Funcionalidade de Cadastro de Exercícios

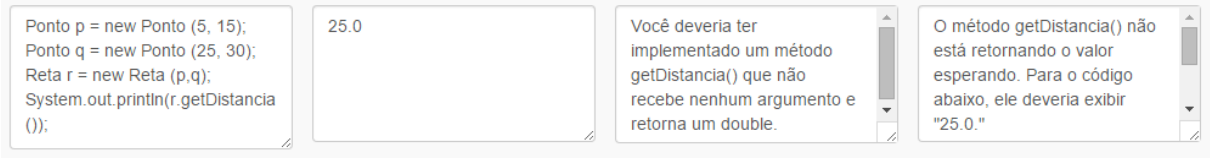
As alterações na funcionalidade de Cadastro de Exercícios visam diminuir a carga de trabalho do professor, automatizando o processo de geração de casos de teste.

O primeiro passo no aprimoramento desta funcionalidade foi a implementação de um mecanismo que possibilita ao professor carregar classes ou interfaces para o sistema. Com este mecanismo, tornou-se possível a disponibilização de classes auxiliares, que poderão ser automaticamente incluídas na solução do aluno e poderão ser personalizadas por ele, conforme necessário. Este mecanismo também possibilita que o professor carregue classes ou interfaces que lhe auxiliarão no cadastro de dados 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, cria automaticamente casos de teste que verificam se a solução do aluno atende todos os requisitos de assinatura definidos no exercício. O mecanismo gera casos de teste que testam a assinatura dos construtores, métodos de acesso, métodos de modificação de atributos e demais métodos das classes da solução do aluno.Este mecanismo utiliza como base uma classe ou interface carregada pelo professor, e é capaz de gerar casos de teste que verificam se a classe encontrada na solução do aluno possui 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 (2014). O caso de teste era composto pelas mensagens de *feedback*, as instruções de teste, que consistiam no código Java a ser executado para testar a solução do estudante, e o 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)

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. O caso de teste passou a ser composto pelas mensagens de *feedback* e por passos estruturados, divididos entre cinco tipos:

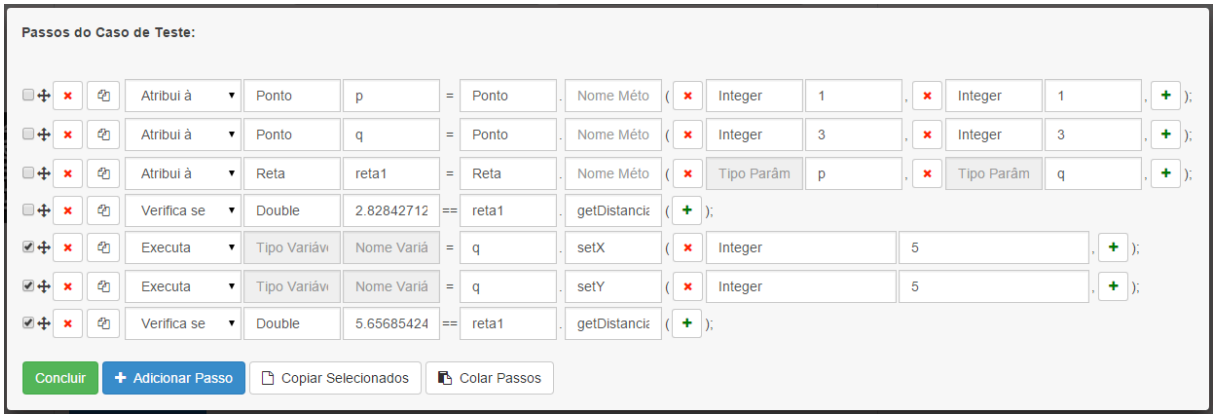
1. **Atribuição**: Recebe uma instrução e atribui o valor resultante a uma variável. A instrução recebida pode representar um valor de tipo primitivo, uma instanciação de novo objeto, uma chamada de método, ou uma operação lógica ou aritmética.
2. **Execução**: Recebe uma instrução a ser executada sem guardar resultado. A instrução recebida geralmente será a chamada de método de um objeto previamente instanciado.
3. **Verificação**: Operação utilizada para verificar se a solução atingiu um determinado objetivo. Esta instrução é capaz de comparar: Duas variáveis; Variável e valor de tipo primitivo; Variável e chamada de método de um objeto previamente instanciado; Valor de tipo primitivo e chamada de método de um objeto previamente instanciado; Variável e dados imprimidos na tela; Valor de tipo primitivo e dados imprimidos na tela;
4. **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. Esta instrução é capaz de utilizar como critério de comparação os mesmos critérios que a operação Verificação;
5. **Fechamento de Laço**: Operação que determina o fim de um laço, sendo possível que um caso de teste possua laços com laços internos.

A estrutura de passos acima 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, o que torna possível a utilização da

ferramenta com diferentes linguagens de programação. Isto é possível, pois esta estrutura de dados é repassada ao módulo de correção automática, sendo ele 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, onde foram implementadas opções de auto completar nos campos de preenchimento dos dados. Nestas opções, foram incluídas: os tipos de dados primitivos; classes ou interfaces carregadas para o sistema; e objetos instanciados em passos anteriores do caso de teste em edição. Além disto, 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 este método é inserido no campo que define o método a ser executado. Deste modo, o professor não precisará incluir os parâmetros manualmente e apenas precisará determinar o valor deste 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. Este 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 esta camada encontra-se no lado do servidor.

Esta 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 desta abordagem, esta 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 citado 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. Esta limitação estava presente, pois, segundo Alves e Jaques (2014), a versão original do Feeper utilizava o Juiz Online CodeJudge para realizar a avaliação.

Durante o trabalho proposto, 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 e de alertas provenientes da análise estática do código.

A Figura 4 ilustra um *feedback,* gerado pelo módulo de avaliação, apresentado ao aluno, no qual constam respectivamente 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 este módulo em um serviço executado em uma aplicação isolada e acionado por requisições assíncronas através do protocolo de comunicação HTTP. A utilização desta abordagem reduz o acoplamento e aumenta a coesão da aplicação. Além disto, devemos 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 desta estrutura é que, em conjunto com os casos de teste com passos estruturados, tornou possível a utilização do sistema com várias linguagens de programação. Isso se deve ao fato que os dados estruturados são repassados ao módulo corretor no formato XML, e cabe a este 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 apresentar uma representação exata dos erros que ocorrem durante a avaliação

dinâmica. Deste 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(2014), nos 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 pelo trabalho proposto, é apresentada a mensagem definida pelo professor seguida da representação exata do erro gerada pelo sistema. Esta 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 desta representação exata dos erros tornou-se possível, pois módulo de avaliação implementado pelo trabalho proposto converte cada caso de teste cadastrado pelo professor em um caso de teste do *framework JUnit*. Após a criação dos casos de teste, os mesmos são executados e o resultado da execução é apresentado para o aluno.

As mensagens personalizadas possuem 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*, auxiliam 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 ao aluno a descrição exata do problema. Como exemplo destas descrições, destacam-se, 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 pelo trabalho proposto também realiza a análise estática da solução do aluno, enriquecendo ainda mais o retorno provido. Esta análise varre a 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 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 estes 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 *loops* infinitos.

Para realizar a análise estática, o módulo de avaliação utiliza a ferramenta FindBugs. (HOVEMEYER; PUGH, 2014). 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 estes 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. (AHTIAINEN; SURAKKA; RAHIKAINEN, 2006).

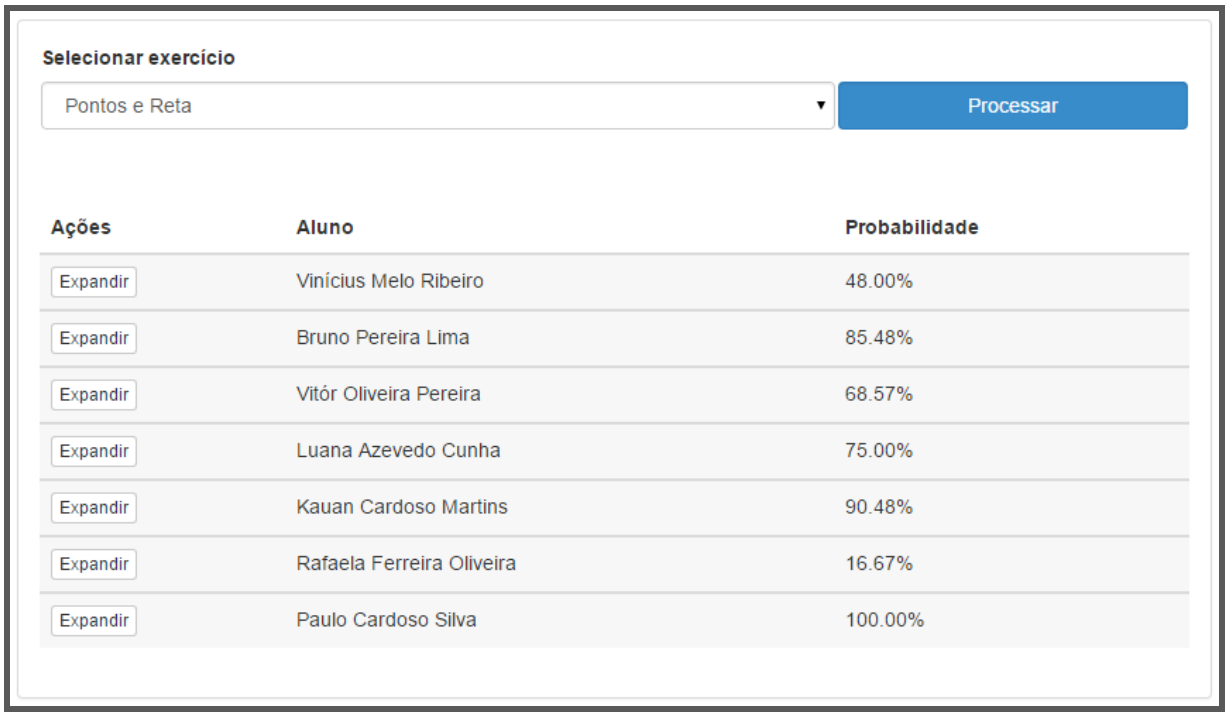
O Plaggie consiste em uma aplicação Java, desenvolvida por pesquisadores de Universidade de Helsinski, com o propósito de auxiliar na detecção de plágio nos exercícios aplicados aos alunos de programação. Entre as principais vantagens da utilização da ferramenta, destaca-se o fato de a ferramenta realizar a análise sem a necessidade de enviar as soluções para um serviço na internet e o fato da ferramenta possuir código aberto. (AHTIAINEN; SURAKKA; RAHIKAINEN, 2006).

A busca por plágio em exercícios de programação não pode consistir simplesmente em uma análise textual, pois deste 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 esta busca utilizando o algoritmo *Greedy String Tiling* (GST), proposto por Michael Wise. Este algoritmo

converte os arquivos das soluções em uma sequência de *tokens*, onde 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. (LUKE et al., 2014).

A figura 5 ilustra a tela de apresentação dos resultados desta análise. Nesta 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
(os nomes dos estudantes foram trocados para fins de anônimato)

5 AVALIAÇÃO

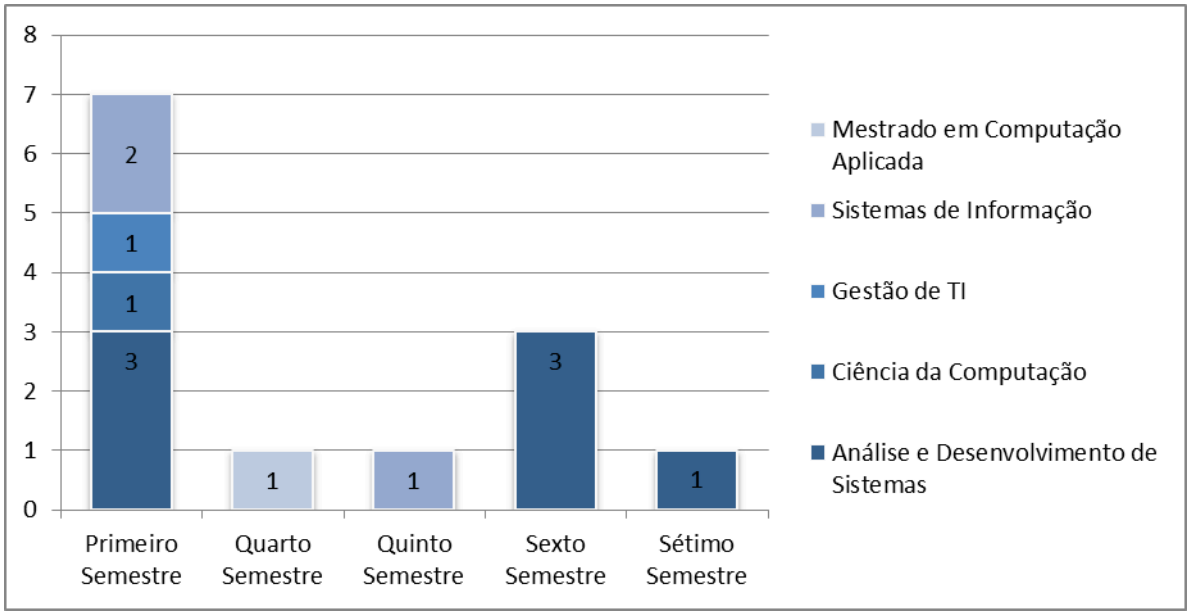
Para evidenciar e analisar os impactos das mudanças realizadas no Feeper pelo trabalho proposto foi realizada uma avaliação, que contou com a participação de uma amostra por conveniência de 13 alunos voluntários dos cursos de tecnologia da Informação da UNISINOS, recrutados através de uma mensagem de divulgação na lista de discussão dos cursos de graduação em informática da UNISINOS. Nesta

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 submeter a solução. Foi selecionado um professor voluntário, que ficou encarregado de analisar as funcionalidades, 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. Após isto, os alunos utilizaram a ferramenta durante um período de três semanas, sendo que todo o processo foi acompanhado pelo professor voluntário. No fim deste período, foi aplicado um novo questionário para os alunos e outro para o professor com o objetivo de verificar a percepcão dos estudantes sobre a ferramenta.

O questionário aplicado aos alunos no início da avaliação serviu para identificar o aluno, onde este informou sua idade, o curso que está cursando, há quantos semestres está neste curso e se cursou ou trabalhou com programação. A Figura 6 apresenta a distribuição por curso e semestre dos alunos que participaram da avaliação.

Figura 6 – Distribuição dos alunos por semestre e curso.



Fonte: Google Forms

De acordo com as respostas deste questionário, foi possível observar que somente um aluno ainda não estudou e nem trabalhou com programação, três alunos ainda não estudaram programação e cinco alunos ainda não trabalharam com programação. Todos os cinco alunos que ainda não trabalharam com programação estão cursando o primeiro semestre de seu curso.

Após responder este questionário, foi iniciada a utilização da ferramenta. Este processo teve a duração de três semanas, sendo que durante este período foi liberado um exercício de programação a cada semana. Os exercícios aplicados abordaram conceitos de programação ministrados nas disciplinas de programação de 1º semestre da UNISINOS. Os conceitos abordados pelos exercícios foram respectivamente orientação a objetos, conceitos de listas e *arrays* e conceitos de herança e polimorfismo.

Entre os treze alunos voluntários que responderam o 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 entre em contato com o aluno caso identifique que ele está passando por dificuldades. Durante esta avaliação, houveram quatorze interações entre professor e aluno, das quais 75% foram motivadas por dúvidas nos exercícios e no uso da ferramenta, e as demais 35% foram motivadas por erros de compilação.

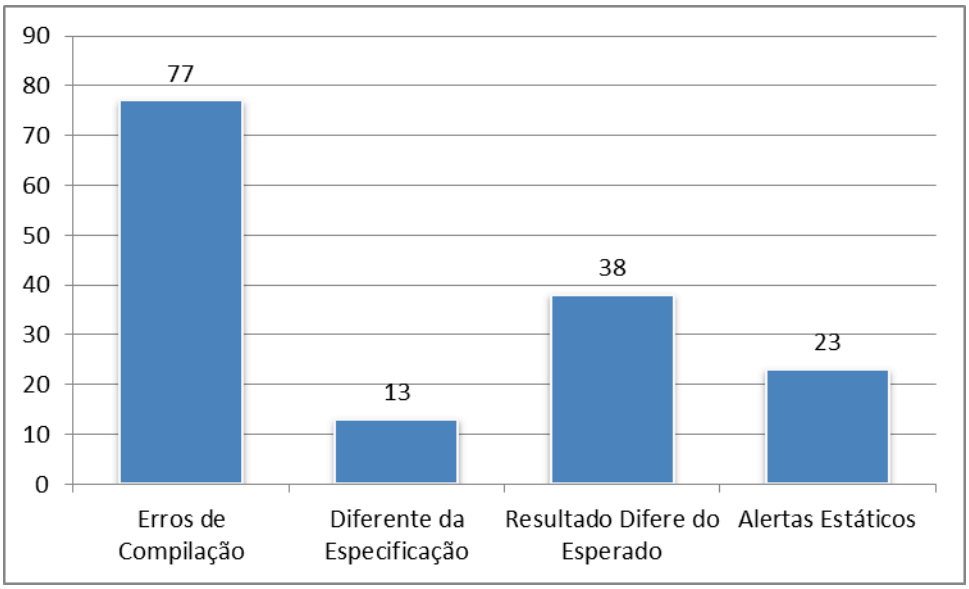
Ao final da avaliação, dentre os nove alunos que utilizaram o Feeper, obteve-se um total de 21 resoluções de exercício bem sucedidas, três mal sucedidas e três ocorrências de exercícios sem tentativa de solução. Durante a avaliação, os alunos submeteram 153 tentativas de solução, totalizando uma média de aproximadamente seis tentativas de cada aluno por exercício. Destas 153 tentativas, 77 apresentaram erros de compilação, 13 possuíam diferenças em relação à especificação do exercício, 38 apresentaram um resultado diferente do esperado na avaliação dinâmica, quatro não possuíam erros mas possuíam problemas de qualidade e 21 representavam soluções sem erros ou problemas de qualidade.

Com base nos dados acima, é possível perceber que os problemas mais impactantes nas soluções dos alunos 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 nos casos em que 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 dinâmica, bem como apresenta 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 do aluno 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 dos alunos e do professor. As perguntas destes questionários foram montadas utilizando a escala de *Likert* de concordância com alternativas com peso de 1 a 5, 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 o questionário da pesquisa. De acordo com a 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). O trabalho proposto 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 (2014) (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 ao professor que este avaliasse as funcionalidades implementadas pelo trabalho proposto na escala *Likert* e também foi disponibilizado um campo para justificar as notas e apresentar críticas e oportunidades de melhoria. O professor julgou importante a utilização de ferramentas que apoiam 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 como 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 funcionamento desta funcionalidade.

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 apresentação de uma lista de colegas que já conseguiram solucionar tal erro. Deste modo os alunos poderão interagir entre si buscando 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; estes 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 trabalho proposto objetivou diversas melhorias no AVA Feeper, desenvolvido por Alves e Jaques (2014). O trabalho proposto reformulou a avaliação dinâmica e combinou esta avaliação com técnicas de análise estática de código, enriquecendo o *feedback* provido ao aluno. Para o professor, o trabalho proposto criou um mecanismo capaz de automatizar a criação de parte dos dados de teste e tornou os casos de teste independentes da linguagem de programação utilizada no exercício. Também foi criado um mecanismo capaz de evidenciar a ocorrência de plágio entre as soluções dos alunos.

As mensagens resultantes da avaliação dinâmica, além da mensagem definida pelo professor, passaram a contar também com uma mensagem gerada pelo sistema. Esta mensagem pode 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 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 pelo professor, 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 plágios nas mesmas.

O trabalho proposto 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 do trabalho proposto foi realizada com uma amostra por conveniência de 13 estudantes de cursos de 3º grau em informática, que usaram Feeper para desenvolver projetos de programação por 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 também foi possível identificar pontos a serem melhorados, por exemplo, a implementação de técnicas de gamificação, a compilaçao da solução do aluno em tempo de codificação e a tradução das mensagens geradas pelo compilador para a língua portuguesa.

6.1 TRABALHOS FUTUROS

Durante o desenvolvimento do trabalho proposto, foram identificadas oportunidades de melhoria que podem ser aplicadas para aperfeiçoar o uso 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.

As mensagens geradas pelo compilador representam outra oportunidade de melhoria, que é a tradução destas mensagens para a língua portuguesa. Foi identificado durante o trabalho proposto, que os alunos possuem certo grau de dificuldade em interpretar as mensagens da língua inglesa geradas pelo compilador.

Outra oportunidade de melhoria, que também foi identificada por Alves e Jaques (2014), é a implementação de técnicas de gamificação, onde o sistema deve permitir que os alunos interajam entre si buscando resolver suas dúvidas, sendo atribuída uma pontuação para o aluno cada vez que ele auxilia o colega.

COMBINING TECHNIQUES OF DYNAMIC ANALYSIS AND STATIC EVALUATION OF CODE IN VIRTUAL ENVIRONMENTS FOR LEARNING PROGRAMMING

Abstract: Programming learning has proved to be a challenge for most higher education students. The teaching strategy employed is usually "learning by doing", involving the application of various programming exercises. These activities are usually facilitated by tools that make it possible for students to submit exercises solution online and automate the assessment process. However, the feedback provided by these tools often limited to right or wrong, and can be enhanced in order to assist students in the development of the activity. The proposed work presents an improved feedback mechanism that combines static analysis techniques and code dynamic evaluation, presenting quality indicators of the submitted solution, a message defined by the teacher and a technical description of errors found in the solution. This work also provides a mechanism for test cases registration, which does not require much technical knowledge and it is able to automate the generation of test cases that validate the signature of the classes to be submitted, speeding up this activity to the teacher. The proposed work was integrated into a virtual learning environment to support programming disciplines, which contains features for registration programming projects and solutions submission by students, called Feeper. An assessment of the proposed work was carried out with a convenience sample of 13 students of 3rd degree courses in computer science, who used Feeper to develop programming projects for 3 weeks. After three weeks, a questionnaire with closed questions whose answers followed the Likert scale of agreement was applied. The results showed that the features implemented helped students in solving exercises.

Keywords: Automatic Assessment. Instantaneous Feedback. Programming Teaching. Dynamic Evaluation. Static Analysis.

REFERÊNCIAS

AHTIAINEN, A.; SURAKKA, S.; RAHIKAINEN, M. **Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises.** *In*: VI Baltic Sea Conference on Computing Education Research. **Anais**...New York: ACM, 2006.

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. **Anais**...New York: ACM Press, 2005.

ALVES, F. P.; JAQUES, P. **Um Ambiente Virtual com Feedback Personalizado para Apoio a Disciplinas de Programação.** *In*: XXV Simpósio Brasileiro de Informática na Educação. **Anais**...Dourados: Universidade Federal Da Grande Dourados, 2014.

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

BORGES, M. **Avaliação de uma metodologia alternativa para a aprendizagem de programação.** *In*: VIII Workshop de Educação em Computação. **Anais**...Campinas: Faculdade Campo Limpo Paulista, 2000.

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. **Anais**...La Plata: Universidad Nacional de La Plata, 2011.

CHAVES, J. O. M. et al. **Integrando Moodle e Juízes Online no Apoio a Atividades de Programação.** *In*: XXIV Simpósio Brasileiro de Informática na Educação. **Anais**...Campinas: Universidade Estadual de Campinas, 22 nov. 2013.

EDWARDS, S. H. **Teaching software testing.** *In*: XVIII annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. **Anais**...New York, New York, USA: ACM Press, 2003.

EDWARDS, S. H.; PEREZ-QUINONES, M. A. Web-CAT. **ACM SIGCSE Bulletin**, v. 40, n. 3, p. 328, 25 ago. 2008.

GIRAFFA, L. M. M.; MÓRA, M. C. **Evasão e Disciplina de Algoritmo e Programação: Um Estudo a partir dos Fatores Intervenientes na Perspectiva do Aluno.** *In*: III Conferência Latino Americana Sobre o Abandono na Educação Superior. **Anais**...Cidade do México: Universidade Nacional Autônoma do México, 2013.

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

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

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

JOHNSON, C. **SpecCheck.** *In*: XVII ACM annual conference on Innovation and technology in computer science education. **Anais**...New York: ACM Press, 2012.

JOY, M.; GRIFFITHS, N.; BOYATT, R. The boss online submission and assessment system. **Journal on Educational Resources in Computing**, v. 5, n. 3, p. 2–28, 1 set. 2005.

KINNUNEN, P.; MALMI, L. **Why students drop out CS1 course?** *In*: International workshop on Computing education research. **Anais**...New York: ACM Press, 2006.

LUKE, D. et al. Software Plagiarism Detection Techniques : A Comparative Study. **International Journal of Computer Science and Information Technologies**, v. 5, n. 4, p. 5020–5024, 2014.

MERRILL, D. et al. Tutoring: Guided Learning by Doing. **Cognition and Instruction**, v. 13, n. 3, p. 315–372, 1995.

MOREIRA, M. P.; FAVERO, E. L. **Um Ambiente para Ensino de Programação com Feedback Automático de Exercícios.** *In*: XVII Workshop sobre Educação em Computação. **Anais**...Bento Gonçalves: Universidade Federal do Rio Grande do Sul, 2009.

MOTA, M. P.; PEREIRA, L. W. K.; FAVERO, E. L. **JAVATOOL: uma ferramenta para ensino de programação.** *In*: XVII Workshop sobre Educação em Computação. **Anais**...Belém do Pará: Universidade Federal de Minas Gerais, 2008.

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*: XVII Workshop sobre Educação em Computação. **Anais**...Bento Gonçalves: Universidade Federal do Rio Grande do Sul, 2009.

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**, v. 8, n. 3, 2010.

SIROTHEAU, S. et al. **Aprendizagem de iniciantes em algoritmos e programação : foco nas competências de autoavaliação.** *In*: XVII Workshop de Informática na Escola. **Anais**...Aracaju: Universidade Federal de Alagoas, 2011.

WERAGAMA, D.; REYE, J. Analysing Student Programs in the PHP Intelligent Tutoring System. **International Journal of Artificial Intelligence in Education**, v. 24, n. 2, p. 162–188, 4 fev. 2014.

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

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**, v. 56, n. 0, p. 767–773, out. 2012.

GLOSSÁRIO

ANGULARJS: Framework para construção de aplicações web.

ASSINATURA: Especificação da estrutura de uma classe.

CLASSE: Arquivos de código fonte das linguagens de programação.

FEEDBACK: Termo inglês que tem o significado de um parecer, um comentário a respeito de determinado assunto.

FEEPER: Nome dado ao sistema desenvolvido por Alves e Jaques (2014), é um acrônimo entre as palavras *feedback* personalizado.

GREEDY STRING TILING: Algoritmo que compara dois textos e encontra partes contidas em ambos textos.

HTTP: Protocolo de comunicação entre aplicações distintas através da Internet.

IDE: Ambiente utilizado durante o desenvolvimento e construção de um sistema computacional.

INTERFACE: Arquivos de código fonte que implementa a estrutura de classes que herdam da interface.

INSTÂNCIA: Na programação orientada a objetos, instância significa é objeto armazenado em memória.

JAVA: Linguagem de programação.

JELASTIC: Serviço de hospedagem de aplicações na nuvem.

JUNIT: Biblioteca para criação de testes para a linguagem de programação *Java*.

LEARNING BY DOING: Termo em inglês que tem o significado de aprender fazendo.

MÉTODO: Função que determina o comportamento de uma classe.

MOODLE: Ambiente Virtual de Aprendizagem.

MVC: Padrão de arquitetura de software.

N-GRAMA: Uma subsequência de n itens construídos a partir de uma determinada sequência.

PHP: Linguagem de programação.

PIB: Produto interno bruto representa a soma de todos os bens e serviços finais produzidos pelo país.

PLUGIN: Componente que adiciona uma determinada funcionalidade a um sistema existente.

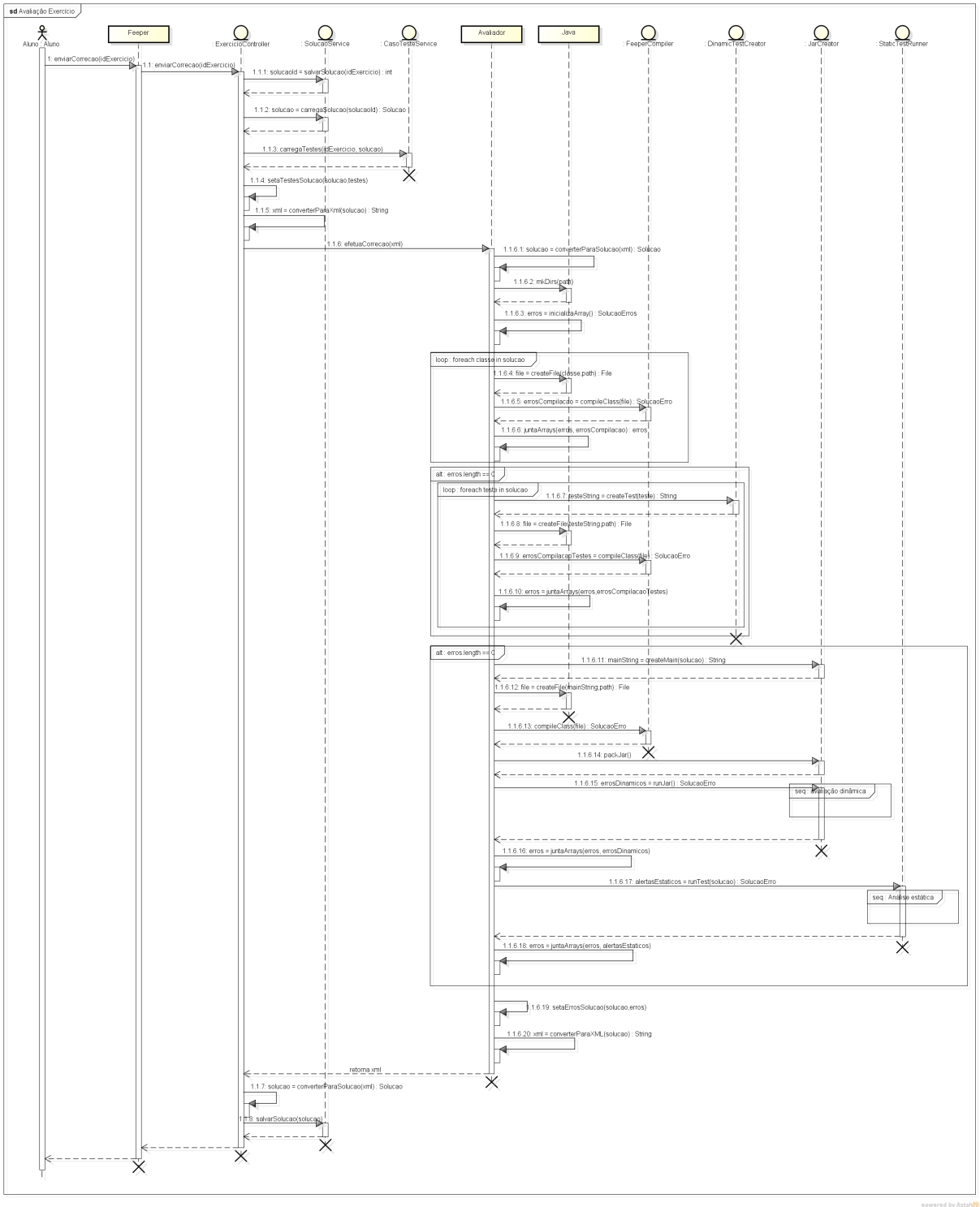
TOKEN: Padrão encontrado dentro de determinado texto.

UPLOAD: Ação de transferir arquivos de um computador para um local remoto.

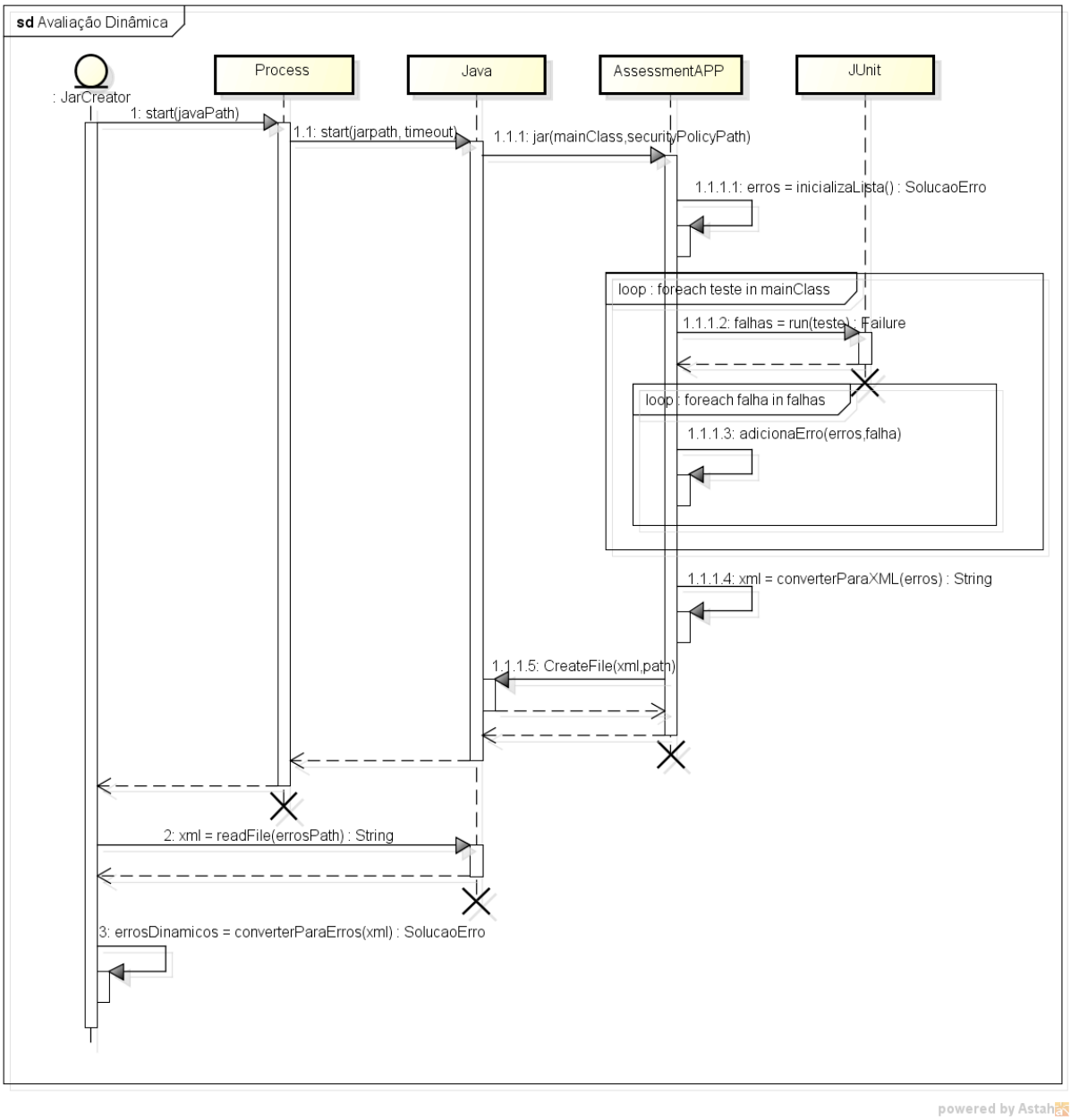
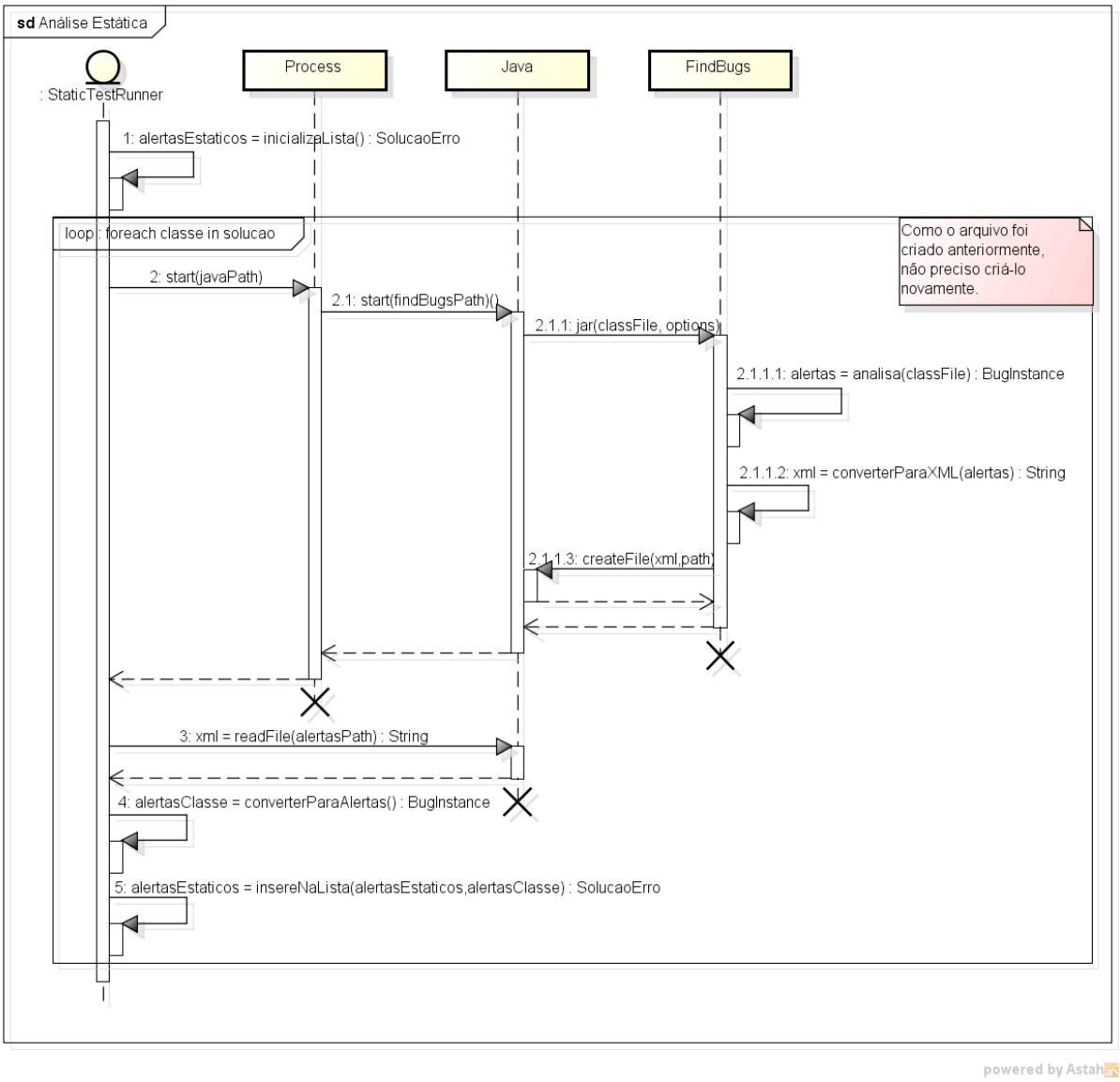
VARIÁVEL: Objeto frequentemente localizado na memória, capaz de reter e representar um valor ou expressão.

WEBSERVICE: Solução utilizada para a integração de sistemas e na comunicação entre aplicações distintas.

APÊNDICE A – Diagrama de sequência do processo de avaliação das soluções



APÊNDICE B – Diagrama de sequência da avaliação dinâmica



APÊNDICE C – Diagrama de sequência da análise estática