Uma nova metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação baseada em testes de unidade

Thiago F. Noronha[1], **Natã Goulart**[2],
Matheus A. Pacheco[3], **Gabriel G. A. Barbosa**[3]

[1]Departamento de Ciência da Computação, UFMG
Av. Antônio Carlos, 6627 - 31270-010 - Belo Horizonte - MG - Brazil

[2]Mestrado em Modelagem Matemática e Computacional - CEFETMG
Av. Amazonas 7675 - 30.510-000 - Belo Horizonte - MG - Brasil

[3]Centro de Pesquisas e Desenvolvimento em Engenharia Elétrica, UFMG
Av. Antônio Carlos, 6627 - 31270-010 - Belo Horizonte - MG - Brazil

tfn@dcc.ufmg.br, natan@dppg.cefetmg.br,

gabriel.guedesaz@gmail.com, matheuspacheco100@gmail.com

Abstract. *This paper proposes a new methodology for developing systems that automatically correct computer programming exercises based on unit tests. Using this methodology, students have each unit of their code evaluated just after they program it, without the need of professor intervention. In the case of some piece of code (procedure or function) is incorrect, an error message is displayed to the student, accompanied by an explanation that may help him to implement that piece of code correctly. A case study in classes of Algorithms and Data Structures of the Federal University of Minas Gerais showed that the unit tests helped the students to rapidly identify and correct the errors in their codes without the interference of the professor.*

Resumo. *Este trabalho propõe uma nova metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação baseada em testes de unidade. Utilizando esta metodologia, os alunos tem cada unidade de seu código avaliada imediatamente após a sua codificação sem a necessidade de intervenção do professor. No caso de alguma unidade (procedimento ou função) estar incorreta, uma mensagem de erro é exibida para o aluno, acompanhada por uma explicação que pode ajudá-lo a implementar aquela unidade corretamente. Um estudo de caso em turmas de Algoritmos e Estruturas de Dados da UFMG mostrou que o uso dos testes de unidade permitiu que os alunos rapidamente identificassem e corrigissem os erros de seu código sem a ajuda do professor da disciplina.*

1. Introdução

Com a adoção de várias universidades federais ao programa de Reestruturação e Expansão das Universidades Federais - REUNI, espera-se um grande aumento no número de alunos por turma nas disciplinas de ensino de programação de computadores. A estimativa

é que o número médio de alunos por turma em disciplinas como Algoritmos e Estrutura de Dados I e II da Universidade Federal de Minas Gerais - UFMG chegue a aproximadamente 100 alunos. Estas disciplinas são oferecidas em diversos cursos de graduação em instituições de todo o país. Na UFMG, vários cursos oferecem estas disciplinas em sua grade curricular, dentre os quais podemos citar: Ciência da Computação, Engenharia de Controle e Automação, Engenharia da Produção, Engenharia Elétrica, Estatística, Matemática Computacional e Sistemas de Informação.

O ensino de programação de computadores exige dos alunos a resolução de um grande número de exercícios, a fim de que eles desenvolvam o raciocínio lógico sequencial necessário para solucionar problemas computacionais. A correção destes exercícios demanda muito tempo do professor da disciplina. Com o crescimento do número de alunos nas disciplinas de ensino de programação, a correção manual de todos os exercícios se tornará impraticável.

A abordagem mais empregada atualmente para correção automática de exercícios de programação é baseada nos chamados *Juízes Online* (do inglês *Online Judge*) [Cheang et al. 2003]. Nesta abordagem, o aluno escreve um programa de computador que soluciona um problema computacional e submete seu programa via Internet para um servidor de testes. Este servidor trata o programa do aluno como uma ”caixa preta”, aplicando-o a vários casos de teste. Se o resultado do programa do aluno for igual ao resultado esperado, o juiz *online* atesta que o programa do aluno está correto. Uma vez que o programa do aluno não produz o resultado esperado, o juiz *online* não tem como indicar qual é o ponto do programa que está errado, nem sugerir uma possível solução. Além disso, nenhuma pré-condição pode ser avaliada dentro do programa do aluno. Mesmo que o enunciado do exercício exija que o aluno utilize um determinado algoritmo ou estrutura de dados específica, isto não pode ser avaliado pelo juiz *online*. Uma outra desvantagem desta abordagem, está na necessidade da compra e manutenção de um servidor de testes que demanda recursos financeiros e de pessoal técnico da universidade, o que impede sua utilização em centros de ensino com poucos recursos financeiros. Além disso, se por alguma razão o servidor estiver congestionado ou inoperante, os alunos não podem avaliar os seus exercícios. Surge então a necessidade de se estudar outros mecanismos alternativos ou complementares para correção automática de exercícios de programação.

Este trabalho propõe uma nova metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação baseada em testes de unidade (do inglês *unit tests*). A Seção 2 apresenta uma revisão bibliográfica sobre os sistemas utilizados para avaliar algoritmos codificados em determinada linguagem de programação e suas aplicações em sistemas de correção automática de exercícios de programação. A Seção 3 descreve a metodologia para o desenvolvimento de sistemas de correção automática de listas de exercícios de programação proposta neste trabalho. Na Seção 4 é discutido o estudo de caso dessa metodologia aplicada em cursos de Algoritmos e Estruturas de Dados da UFMG. Por fim, a Seção 5 apresenta as considerações finais sobre este trabalho.

2. Trabalhos Relacionados

Esta seção apresenta uma revisão dos principais sistemas e tecnologias que podem ser aplicadas no ensino de linguagens de programação e em outros cursos *online*.

2.1. Juízes Online

Os sistemas baseados em juízes *online* são capazes de receber o código fonte de um programa codificado em uma linguagem de programação qualquer e verificar se este programa apresenta, para determinadas entradas pré-definidas, as saídas esperadas. Estes sistemas foram inicialmente concebidos para uso em concursos e competições de programação [ICPC 2010]. Nos últimos anos porém, instituições de ensino têm utilizado e adaptado estes sistemas para o ensino de linguagens de programação em cursos técnicos e superiores [Kurnia et al. 2001, Kurnia et al. 2008]. Estes sistemas estão sendo integrados a sistemas de ensino e acompanhamento *online* chamados *Learning Management System - LMS* [Woo et al. 2008]. Um dos sistemas LMS mais utilizado para o acompanhamento acadêmico e também em cursos a distância é o *Moodle* [MOODLE 2010], que faz uso das amplas possibilidades da Internet para o ensino presencial ou a distância.

Atualmente, existem propostas de adicionar aos juízes *online* ferramentas com o objetivo de identificar a cópia dos trabalhos que são submetidos. A detecção desta cópia, conhecida na literatura como plágio, é uma função comum nos sistemas de avaliação utilizados no meio acadêmico. Dentre as opções de detecção de plágios disponíveis, podemos citar um *plugin* desenvolvido para o *Moodle* baseado no sistema chamado *Measure Of Software Similarity - MOSS* [Schleimer et al. 2003] que está disponível em [Schleimer et al. 2010].

Dentre as aplicações práticas de sistemas baseados em juízes *online* encontradas na literatura podemos citar o uso do sistema de correção automática que é utilizado na escola de computação da Universidade de Singapura, descrito em [Cheang et al. 2003]. Nesta universidade, até o final da década de 90, os trabalhos acadêmicos na área de ensino de linguagens de programação eram corrigidos manualmente. A partir do ano 2000, esta e outras instituições [Kurnia et al. 2001, Kurnia et al. 2008] começaram a utilizar ferramentas de correção automática de listas de exercício de programação.

Um sistema de correção automática baseado em juízes *online* exige que as saídas geradas pelos programas sigam um padrão e formato rígido. Caso a saída não esteja formatada exatamente igual a do gabarito esperado, o sistema pode considerar que o programa não resolve o problema corretamente. Se por exemplo, o programa espera como saída dados formatados em duas colunas, o resultado pode ser considerado errado caso o aluno use espaços ao invés de tabulações.

2.2. Testes de unidade

Testes de unidade (do inglês *unit tests*) são amplamente utilizados atualmente no desenvolvimento e teste de sistemas comerciais. Eles tem este nome porque cada teste de unidade testa uma unidade lógica do programa independentemente das outras. Em programação estruturada esta unidade lógica pode ser definida como um procedimento ou função [Dustin et al. 1999]. Ao longo do desenvolvimento e manutenção do software, se alguma mudança resultar num erro em uma determinada função os testes relacionados a esta função falham e o erro pode ser imediatamente identificado. Podem existir vários

testes para uma mesma unidade de código, cada um deles avaliando uma propriedade que deve ser atendida por aquela unidade.

Os testes de unidade fazem parte de um conjunto de testes chamados de testes de *caixa branca* [Misra 2003]. Diferentemente dos testes de *caixa preta*, onde o sistema recebe uma entrada e após o processamento gera uma saída, os testes de caixa branca permitem a avaliação do funcionamento interno do sistema. Os testes de unidade são utilizados para verificar se o *software* faz corretamente e adequadamente todos os detalhes especificados no projeto ou, no caso do uso acadêmico, no enunciado do exercício. Na prática, como são testes que irão avaliar os detalhes da implementação, são codificados normalmente pelo próprio desenvolvedor que tem o conhecimento da lógica utilizada. Em sistemas grandes e complexos formados pelas camadas de lógica, dados e interface com o usuário, os testes de unidade devem verificar todos os possíveis caminhos do código, avaliar telas, menus, mensagens de usuário devidamente formatadas, validar campos e verificar se as exceções são devidamente codificadas.

Segundo [Dustin et al. 1999], testes de caixa branca como os testes de unidade geralmente utilizam cerca de 60% do esforço e custo dos testes em um sistema, por outro lado, pode garantir um aumento de cerca de 40% na qualidade do *software* desenvolvido. Devido ao alto custo gerado na elaboração dos testes de unidade, é importante a utilização de algumas ferramentas e metodologias para o seu desenvolvimento e aplicação. Com o objetivo de melhorar a produtividade dos desenvolvedores de testes de unidade foram criados *frameworks* que auxiliam no desenvolvimento e no controle dos testes.

3. Metodologia

A metodologia utilizada para implementar corretores automáticos de listas de exercícios proposta neste trabalho consiste em quatro etapas. Primeiramente, seleciona-se um conjunto qualquer de exercícios relacionados a um tópico dentro do programa da disciplina. Em seguida, cabe ao professor da disciplina especificar qual a interface de cada função que deve ser implementa pelos alunos, especificando os seus parâmetros de entrada e saída. O próximo passo consiste em programar um ou mais testes de unidade para testar cada propriedade que deve ser atendida por cada função. Por fim, é solicitado ao aluno que ele implemente as funções das listas de exercício da forma como especificado no enunciado da questão. Como cada teste de unidade verifica uma única propriedade da função, quando esta falha, é possível exibir para o aluno uma mensagem de erro precisa sobre o que está errado em sua implementação.

Para facilitar a implementação dos testes de unidade foi utilizado o *framework* para desenvolvimento de testes de unidade conhecido como *Google Test*. *Frameworks* são bibliotecas de *software* que auxiliam na implementação de sistemas que tem naturezas semelhantes. Dentre os principais *frameworks* utilizados para a codificação de testes de unidade estão o *JUnit* [JUNIT 2010, Hamill 2004] para programação em *Java* e o *Google Test* [Google Test 2010] para programação em *C++*. O *Google Test* é um *framework* que pode ser compilado e executado em várias plataformas como *Windows, Linux, Mac OS X*, entre outras. Ele é desenvolvido como um projeto de software aberto e distribuído gratuitamente sob os termos da licença *New BSD License* [BSD 2010]. Esta biblioteca dispões de várias, funções, classes e macros em *C++* que permitem a implementação rápida e fácil de teste de unidade. Dentre estas ferramentas, a mais utilizadas são co-

nhecidas como *assertions*, que verificam uma determinada condição da função testada. Quando esta condição não é satisfeita, é gerada automaticamente uma mensagem de erro que pode ser personalizada para cada tipo de erro gerado, informando, por exemplo, os dados de entrada e a saída esperada.

A Figura 1 apresenta o gabarito do exercício que consiste em implementar uma função que recebe um vetor `v` com `n` elementos e retorna a média dos elementos de `v`. Na Linha 2, a variável auxiliar `soma` é inicializada. No laço das linhas 3-5, o somatório dos elementos de `v` é acumulado na variável `soma`. Por fim, a média dos elementos de `v` é retornada na Linha 6. Um erro muito comum neste exercício ocorre quando os alunos escrevem `i <= n`, na Linha 3, em vez da forma correta `i < n`. Uma vez que os vetores em `C++` são indexados de 0 a $n - 1$, isto resulta no acesso ilegal a posição de memória `v[n]`.

Este erro é verificado pelo teste de unidade apresentado na Figura 2. Nas linhas 2 e 3, é criado um vetor `v` com $n = 3$ elementos que será utilizado no teste. Entretanto, para verificar se o aluno não está acessando a posição `v[n]`, um elemento a mais é alocado para este vetor. O valor retornado pela função implementada pelo aluno é armazenado na variável `atual` na Linha 4, enquanto o valor que a função retornaria se estivesse errada é calculado e armazenado na variável `inesperado` na Linha 5. Caso os valores das duas variáveis sejam iguais, uma mensagem de erro é mostrada ao aluno indicando que sua função pode estar acessando a posição inválida `v[n]`. A Figura 3 mostra a mensagem de erro que é exibida ao aluno quando o teste de unidade falha. Note que este teste de unidade testa exclusivamente este tipo de erro. Outros testes de unidade devem ser implementados para testar outras propriedades que devem ser satisfeitas pela função que calcula a média de um vetor. Note também que os detalhes sobre a teoria e implementação de testes de unidade não são apresentados aos alunos, uma vez que esta teoria está além do escopo do curso de ensino de programação de computadores e exige conhecimentos ainda não ensinados para os alunos.

Cada lista de exercícios é disponibilizada na forma de um pacote de arquivos que é composto de (i) um arquivo com o enunciado das questões, (ii) um arquivo com as interfaces das funções que devem ser implementadas pelos alunos (e.g. vetores.h), (iii) um arquivo que contém um ou mais testes para cada função da lista (e.g. vetores_test.h), além da (iv) biblioteca do *Google Test*, necessária para compilar os teste de unidade. Cabe ao aluno escrever o arquivo que contém a implementação de todas as funções referentes a lista de exercícios (e.g. vetores.cc). Aos alunos são dadas duas opções de compilação. No modo `Main`, o programa do aluno é executado normalmente. Já no modo `EasyTesting` o programa do aluno não é executado. Em vez disso, todos os testes de unidade são executados e um relatório é exibido para o aluno com a lista das funções que passaram ou não passaram nos testes (Figura 3). Para cada teste que falha, uma mensagem de erro é exibida acompanhada por uma explicação que pode ajudar o aluno a solucionar erro.

4. Resultados e discussões

Um estudo de caso foi realizado em turmas de Algoritmos e Estruturas de Dados II da UFMG em 2010 e 2011. Estas turmas continham até 70 alunos matriculados. Foram desenvolvidas listas de exercício com os respectivos testes de unidade para os seguintes

```
1  float Media(float v[], int n) {
2    float soma = 0;
3    for (int i = 0 ; i < n ; i++) {
4      soma += v[i];
5    }
6    return soma / n;
7  }
```

Figura 1. Uma possível solução para a função que calcula a média de um vetor `v` com `n` elementos.

```
1  TEST_F(Teste, Testa_acesso_a_uma_posicao_invalida_do_vetor_na_funcao_Media) {
2    int n = 3;
3    float v[3 + 1] = {4.0, 8.0, 14.0, 1000.0};
4    float atual = Media(v, n);
5    float inesperado = (4.0 + 8.0 + 14.0 + 1000.0) / n;
6    ASSERT_NE(atual, inesperado)
7      << "-------------------------------------------------------------------\n"
8      << "Erro na funcao \"float Media(float v[], int n)\".                  \n"
9      << "-------------------------------------------------------------------\n"
10     << "  Sua função pode estar acessando a posição inválida v[n].         \n"
11     << "-------------------------------------------------------------------\n";
12 }
```

Figura 2. Teste de unidade que verifica se a função que calcula a média de um vetor não acessa um posição inválida do vetor.

```
vetores
[==========] Executando 5 testes 1 test case.
[----------] Iniciando testes.
[EXECUTANDO] Testa_acesso_a_uma_posicao_invalida_do_vetor_na_funcao_Media

-------------------------------------------------------------------
Erro na funcao "float Media(float v[], int n)".
-------------------------------------------------------------------
  Sua função pode estar acessando a posição inválida v[n].
-------------------------------------------------------------------

[  FALHOU  ]
[EXECUTANDO] Testa_funcao_Media_de_um_vetor_unitario
[    OK    ]
[EXECUTANDO] Testa_funcao_Media_com_mais_de_um_valor
[    OK    ]
[EXECUTANDO] Testa_funcao_Media_com_numeros_nulos
[    OK    ]
[EXECUTANDO] Testa_funcao_Media_numeros_positivos_e_negativos
[    OK    ]
[----------] Finalizando testes.
[==========] Foram executados 5 testes 1 test case.
[ PASSARAM ] 4 testes.
[ FALHARAM ] 1 teste. Veja a baixo a lista de testes que falharam:
             Testa_acesso_a_uma_posicao_invalida_do_vetor_na_funcao_Media
```

Figura 3. Mensagem de erro exibida quando o teste de unidade apresentado na Figura 2 falha.

tópicos referentes a disciplina: (i) procedimentos e funções, (ii) instruções condicionais e operadores lógicos, (iii) comandos de repetição, (iv) vetores, (v) matrizes, (vi) registros, (vii) recursão, (viii) listas encadeadas, (ix) pilhas, (x) filas, (xi) árvores binárias de busca, (xii) tabelas de dispersão, entre outros. Estas listas de exercício e os respectivos testes de unidade estão disponíveis para a comunidade acadêmica através do projeto de código aberto EasyTesting (`http://code.google.com/p/easytesting`).

As listas de exercício foram disponibilizadas semanalmente após cada tópico da matéria ser abordado em sala de aula. Nenhuma informação sobre como os testes de unidade foram implementados foi apresentada aos alunos. Foi informado apenas como executar os testes e consultar as mensagens de erro. Após a aplicação das listas de exercícios observou-se que o uso de testes de unidade foi um agente motivador para o aprendizado dos alunos, que tiveram menos dificuldade na resolução das listas de exercícios, pois tiveram um retorno imediato sobre a corretude dos seus programas ou possíveis pontos de falha em suas implementações. Isto permitiu que os alunos pudessem solucionar os erros em suas questões e avançar para os próximos tópicos sem a necessidade de aguardar instruções do professor.

Em aulas de laboratório, a correção automática dos exercícios possibilitou que o professor da disciplina pudesse dar uma atenção adequada para uma turma com grande número de alunos, uma vez que os erros mais fáceis e mais comuns puderam ser solucionados pelos alunos apenas com as mensagens de erro exibidas pelos testes de unidade. Isto permitiu ao professor concentrar seu tempo em auxiliar os alunos apenas na solução dos problemas mais complexos. Já fora da sala de aula, a correção automática dos exercícios permitiu que o professor gastasse menos tempo na correção de listas de exercícios e investisse este tempo na preparação das aulas e em outras atividades acadêmicas. Além disso, não foi necessária a compra nem a manutenção de um servidor de testes, uma vez que os testes de unidade são compilados e executados junto com o código do aluno.

Dentre os pontos negativos observados neste estudo de caso pode-se citar: (i) a metodologia não permite o teste da função principal de cada programa (*main()* em C++). Para a verificação desta função pode-se fazer uso complementar de sistemas como os juízes *online*. Além disso, (ii) durante a implementação dos testes de unidade pode ocorrer que determinados casos de teste não sejam codificados, dando a impressão ao aluno que sua codificação incorreta esteja correta. Este é um problema inerente a todos os sistemas de teste de *software* que neste caso pode ser contornado confrontando o aluno com o gabarito da questão após o encerramento da atividade. Ao surgir uma dúvida por parte do aluno, a divergência pode ser esclarecida e um novo teste de unidade que trata este caso pode ser adicionado. Por fim, uma outra desvantagem desta abordagem, é que as interfaces das classes e funções a serem implementadas são previamente definidas no enunciado do exercício e não podem ser alteradas pelos alunos. Isto não é um problema em cursos básicos de programação e podem ser contornados em cursos avançados com a adoção de trabalhos práticos complementares.

5. Considerações Finais

Neste trabalho foi proposta uma nova metodologia para implementação de sistemas de correção automática de listas de exercícios de programação baseada em testes de unidade. O *framework Google Test* foi utilizado na construção de testes de unidade para listas

de exercícios sobre diversos temas relacionados ao ensino de programação em cursos de graduação em computação e em engenharia. Através da aplicação deste método em turmas da disciplina Algoritmos e Estruturas de Dados da UFMG, observou-se que o método auxilia no ensino de linguagens de programação, principalmente para turmas com um grande número de alunos. Isto reduziu consideravelmente o esforço do professor da disciplina na correção de exercícios e o tempo gasto por ele tirando dúvidas dos alunos nas aulas de laboratório. O uso dos testes de unidade permitiu que os alunos rapidamente identificassem e corrigissem os erros de seu código sem a ajuda do professor da disciplina.

Trabalhos futuros podem estudar a utilização em conjunto da abordagem proposta neste trabalho com outras baseadas em sistemas de juízes *Online* e com ferramentas de verificação de cópias de trabalho para uma avaliação completa de listas de exercícios.

Referências

BSD (2010). New bsd license. http://www.opensource.org/licenses/bsd-license.php (referência on-line).

Cheang, B., Kurnia, A., and Oon, A. L. W. (2003). On automated grading of programming assignments in an academic institution. *Computers and Education*, 41:121–131.

Dustin, E., Rashka, J., and Paul, J. (1999). *Automated Software Testing - Introduction, Management and Performance*. Addison-Wesley, Reading.

Google Test (2010). http://code.google.com/p/googletest/w/list (referência on-line).

Hamill, P. (2004). *Unit Test Frameworks - Tools for High-Quality Software Development*. O'Reilly Media, Reading.

ICPC (2010). Acm-icpc. http://cm2prod.baylor.edu/welcome.icpc (referência on-line).

JUNIT (2010). http://www.junit.org/ (referência on-line).

Kurnia, A., Lim, A., and Oon, W.-C. (2001). Online Judge. *Computers and Education*, 36:299–315.

Kurnia, A., Malafiejsk, M., and Noinsk, T. (2008). Application of an Online Judge and Contester System in Academic Tuition. *Advances in Web Based Learning in ICWL 2007*, 4823-2008:343–354.

Misra, S. (2003). Evaluating four white-box test coverage methodologies. In *Evaluating four white-box test coverage methodologies*, volume 3, pages 1739 – 1742 vol.3.

MOODLE (2010). http://moodle.org/ (referência on-line).

Schleimer, S., Wilkerson, D. S., and Aiken, A. (2003). Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, New York, NY, USA. ACM.

Schleimer, S., Wilkerson, D. S., and Aiken, A. (2010). Moss plugin. http://code.google.com/p/sunner-projects/w/list (referência on-line).

Woo, Y.-H., Hong, S.-W., and Kim, S.-B. (2008). A study on the self-directed learning management system. In *NCM '08: Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management*, pages 119–122, Washington, DC, USA. IEEE Computer Society.