PROGTEST: Ambiente de Submissão e Avaliação de Trabalhos Práticos

Camila K. Della Corte[1], Ana Cláudia Riekstin[1], Marco Aurélio Graciotto Silva[1], Ellen F. Barbosa[1], José Carlos Maldonado[1]

[1]Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brasil

{camila, ana , magsilva, francine, jcmaldon}@icmc.usp.br

Abstract. *Only a small portion of the Computer Science (CS) curriculum is allocated for teaching software testing. Some experiences have suggested that the teaching of testing should begin as early as possible so an adequate culture of testing could be created. In this paper we explore the integration between the teaching of software testing along with the teaching of programming foundations. For this, we propose* PROGTEST *– a Web-based environment for the submission and automatic evaluation of practical programming assignments based on testing activities, aiming at providing an adequate feedback to evaluate the learners' performance concerning programming and testing.*

Resumo. *Somente uma pequena parte dos currículos dos cursos de Ciências da Computação são alocados para o ensino de teste de software. Algumas experiências sugerem que o ensino de teste deveria começar o mais cedo possível. Desta forma, uma cultura adequada de teste de software seria criada. Neste artigo, é explorado a integração do ensino de teste de software e fundamentos de programação. Para isso, é proposto um ambiente para submissão e avaliação automática de trabalhos práticos baseado em atividades de teste* – ProgTest –*, fornecendo um* ***feedback*** *adequado para a avaliação da performance do aprendizado dos conceitos de programação e teste de software.*

1. Introdução

Teste de software é o processo de executar um programa com a intenção de encontrar erros. No contexto de engenharia de software, teste é uma das mais importantes atividades na garantia da qualidade e da confiabilidade do software que está sendo desenvolvido mas, ao mesmo tempo, é um tópico difícil para ensinar ou aprender sem suporte apropriado. Tradicionalmente, teste tem sido ensinado de acordo com a abordagem clássica de desenvolvimento de software somente no final do processo de aprendizagem dos cursos de Ciências da Computação. Além disso, somente uma pequena porção dos currículos do cursos de Ciências da Computação é alocada para a atividade de teste de software em comparação com outras atividades do processo de desenvolvimento de software [Shepard et al. 2001].

Uma das iniciativas que vêm sendo investigadas para amenizar este problema refere-se ao ensino conjunto de conceitos básicos de programação e de teste de software. No entanto, fundamentos de programação não é uma disciplina trivial de ser

ensinada, muitos estudantes têm dificuldades em compreender os conceitos abstratos de programação [Lahtinen et al. 2005] e possuem visões erradas sobre a atividade de programação [Edwards 2004]. Então, a principal mudança que vem acontecendo nos cursos introdutórios de Ciências da Computação é torná-los interessantes e relevantes para os alunos [Hickey 2004].

Uma das iniciativas refere-se a introdução da atividade de teste juntamente com fundamentos de programação. Acredita-se que a introdução da atividade de teste pode ajudar no desenvolvimento das habilidades de compreensão e análise nos estudantes, já que para que a mesma seja realizada é necessário que os alunos conheçam o comportamento dos seus programas [Edwards 2004]. Além disso, experiências recentes têm sugerido que a atividade de teste poderia ser ensinada o mais cedo possível. A idéia é que os alunos que aprendem teste mais cedo podem se tornar melhores testadores e desenvolvedores uma vez que o teste força a integração e aplicação de teorias e habilidades de análise, projeto e implementação [Patterson et al. 2003].

Assim, a aquisição de habilidades o mais cedo possível de conceitos e técnicas de teste pode: (1) melhorar o raciocínio sobre os programas (e sua solução), conduzindo para uma melhor qualidade dos produtos; e (2) induzir e facilitar o uso de teste por todo o processo de desenvolvimento de software, conduzindo a uma melhor qualidade do processo, em contraste com as práticas correntes. Desta forma, a inclusão da atividade de teste de software o quanto antes pode ajudar tanto a aumentar a qualidade do produto final produzido bem como melhorar a qualidade do processo utilizado no seu desenvolvimento. Portanto, a idéia consiste em criar uma "cultura de teste de software" entre os alunos (e desenvolvedores).

No entanto, apesar de todas as vantagens do ensino da atividade de teste, os problemas associados a ele, vão desde a dificuldade de motivar os alunos a realizarem uma boa atividade de teste até a falta de ferramentas de suporte ao ensino de teste em nível introdutório [Patterson et al. 2003].

Dentro do contexto apresentado, o objetivo deste trabalho é mostrar como funciona o ambiente de apoio para submissão e avaliação automática de trabalhos práticos dos alunos, baseado em atividades de teste de software, denominado PROGTEST e como ele pode ser um mecanismo de apoio ao ensino integrado de fundamentos de programação e teste.

Este artigo está organizado da seguinte forma. Na Seção 2 é apresentada uma visão geral dos trabalhos relacionados. Na Seção 3 são discutidas as principais características do ambiente PROGTEST e seus aspectos operacionais. E por fim, na Seção 4, são sintetizadas as contribuições deste trabalho e apresentadas as perspectivas de pesquisa futuras.

2. Trabalhos Relacionados

A seguir é apresentada uma visão geral dos trabalhos relacionados, os quais serviram como base para a condução da presente pesquisa.

2.1. Ambientes de Apoio ao Ensino de Teste de Software

Seguindo a tendência da inclusão da atividade de teste de software o mais cedo possível nos cursos de Ciências da Computação, trabalhos vêm sendo conduzidos a fim de

fornecer suporte ao ensino de teste de software. Nesse sentido, ambientes de apoio estão sendo desenvolvidos em duas perspectivas diferentes. Uma delas é a construção de ambientes de programação pedagógicos que facilitem a construção de casos de teste apoiando a realização da atividade de teste de unidade. Entre os ambientes desenvolvidos, destacam-se o *DrJava* [Allen et al. 2002] e o *BlueJ* [Patterson et al. 2003]. Esses ambientes apóiam o ensino de OO e facilitam a construção de casos de teste no *JUnit* [Massol and Husted 2005], podendo ser utilizados pelos alunos para auxiliar o ensino integrado de fundamentos de programação e de teste de software. Entretanto, é importante ressaltar que tais ambientes não estão integrados a ferramentas de teste.

2.2. Ambientes de submissão de avaliação de trabalhos práticos

Um dos fatores críticos para o sucesso do ensino integrado de teste de software e de fundamentos de programação é promover um *feedback* apropriado e avaliar a desempenho dos estudantes. Neste caso, haverá uma dupla carga de trabalho, uma vez que, tanto os casos de teste quanto o código do programa terão que ser avaliados. Uma solução para esse problema crítico é o desenvolvimento e utilização de ambientes automatizados para a avaliação de trabalhos práticos, o que poderá reduzir a sobrecarga que existe sobre os professores [Edwards 2004]. Além disso, o uso de um ambiente de avaliação automática de trabalhos práticos traz benefícios adicionais em termos de consistência, eficácia e eficiência. Todo programa submetido é analisado no mesmo nível de eficiência e os resultados da avaliação são baseados nos mesmos padrões. A avaliação tem que ser realizada independentemente do professor ou de qualquer outro efeito externo. Um benefício desses ambientes é o fornecimento de relatórios, após a avaliação, de cada programa dos estudantes. Assim, o aluno pode saber como está o seu desempenho [Isong 2001].

Alguns ambientes para submissão e avaliação automática de trabalhos práticos foram propostos no sentido de facilitar o ensino integrado de teste e de fundamentos de programação, como [Goldwasser 2002, Isong 2001, Jones 2001, Korhonen et al. 2002, Saikkonen et al. 2001]. Esses ambientes recebem os programas desenvolvidos pelos alunos (alguns deles também aceitam casos de teste) e realizam comparações do resultado do programa feito pelo aluno com o resultado esperado. Desta forma, ocorre um julgamento se o programa do aluno está correto ou não. A ênfase desses ambientes está na saída dos programas produzidos pelos alunos, não sendo utilizadas ferramentas de teste de análise de cobertura para avaliar os trabalhos entregues.

3. ProgTest: Ambiente de Submissão e Avaliação de Trabalhos Práticos

3.1. Visão Geral

O ambiente PROGTEST é um ambiente baseado na *Web*, na concepção de software livre, para submissão e avaliação automática de trabalhos práticos. Com isso pretende-se fornecer suporte ao ensino integrado de fundamentos de programação e teste de software. O ambiente PROGTEST foi desenvolvido para aceitar programas escritos em Java. Os casos de teste que são submetidos pelos alunos devem estar no formato do JUnit[Massol and Husted 2005] que é padrão dos casos de teste aceitos pela ferramenta de teste JaBUTi [Vincenzi 2004], utilizada para realizar a análise de cobertura dos programas.

Basicamente, as principais características do ambiente são descritas a seguir. Dado um programa P_{Al_i} (fornecido pelo aluno) e seu respectivo conjunto de casos de teste T_{Al_i}

(gerado com base no critério de teste C_K previamente estabelecido - T_{Al_i} é $C_{K-adequado}$), o ambiente, integrado às ferramentas de teste pertinentes, deve ser capaz de:

1. executar o programa P_{Al_i} com os conjuntos de casos de teste fornecidos pelo professor T_{Inst};
2. utilizar o conjunto de casos de teste T_{Al_i} para testar o programa "oráculo" P_{Inst}, fornecido pelo professor ;
3. comparar o comportamento de P_{Inst} executado com o conjunto de casos de testes T_{Inst} em relação a P_{Al_i} executado com o conjunto de casos de testes T_{Inst}; e
4. comparar o comportamento de P_{Inst} executado com o conjunto de casos de testes T_{Al_i} em relação a P_{Al_i} executado com o conjunto de casos de testes T_{Al_i}.

A partir das execuções realizadas, o sistema deve ser capaz de decidir pela aceitação ou não do programa e/ou dos casos de teste fornecidos pelo aluno, sugerindo eventualmente uma "nota" aos trabalhos submetidos, tendo como base os índices de cobertura dos conjuntos de casos de teste utilizados.

3.2. Abordagem para avaliação de programas

O ambiente PROGTEST avalia automaticamente os programas dos alunos da seguinte forma. Para cada trabalho existe uma implementação de referência, o programa "oráculo", e um conjunto de casos de teste que são utilizados para testar essa implementação de referência. O aluno submete ao sistema o código fonte do seu programa escrito em Java e os casos de teste escritos no JUnit. Após a submissão, o sistema compila o programa e os casos de teste do aluno e depois realiza as execuções trocadas utilizando a ferramenta de teste JaBUTi, realizando o teste estrutural, e também o framework JUnit para realização do teste funcional. Como mencionado anteriormente, as execuções ocorrem da seguinte maneira:

- Para cada trabalho:
    1. Executa-se o programa do professor (P_{Inst}) com os casos de teste do professor (T_{Inst}) — P_{Inst} - T_{Inst};
- Para cada aluno:
    1. Executa-se o programa do aluno (P_{Al_i}) com os seus respectivos casos de teste (T_{Al_i}) — P_{Al_i} - T_{Al_i};
    2. Executa-se o programa do professor (P_{Inst}) com os casos de teste do aluno (T_{Al_i}) — P_{Inst} - T_{Al_i};
    3. Executa-se o programa do aluno (P_{Al_i}) com os casos de teste do professor (P_{Inst}) — P_{Al_i} - T_{Inst}.

Para cada uma dessas execuções a ferramenta JaBUTi realiza a análise de cobertura do código com os casos de teste utilizados e fornece os índices de cobertura nos critérios de teste por ela implementados, obtendo dessa forma os resultados do teste estrutural dos programas dos alunos. Durante essas execuções, os casos de teste também são executados no JUnit. Dessa forma, têm-se também os resultados do teste funcional dos programas dos alunos, uma vez que executado um caso de teste, a saída obtida é comparada com a saída esperada.

A nota que o sistema fornece é somente uma sugestão, cabendo ao professor decidir se utiliza a mesma como nota real do trabalho. O professor pode alterar essa nota sugerida pelo sistema assim que desejar.

3.3. Arquitetura da ferramenta

Como o ambiente PROGTEST é um sistema para *Web*, sua arquitetura foi elaborada seguindo o padrão de projeto de arquitetura *Model-View-Controller*. A camada de modelo do ambiente PROGTEST é detalhada na Figura 1 onde são mostrados os módulos que compõem as funcionalidades do ambiente PROGTEST. O ambiente possui basicamente três módulos principais: gerenciamento de cursos, submissão e avaliação.

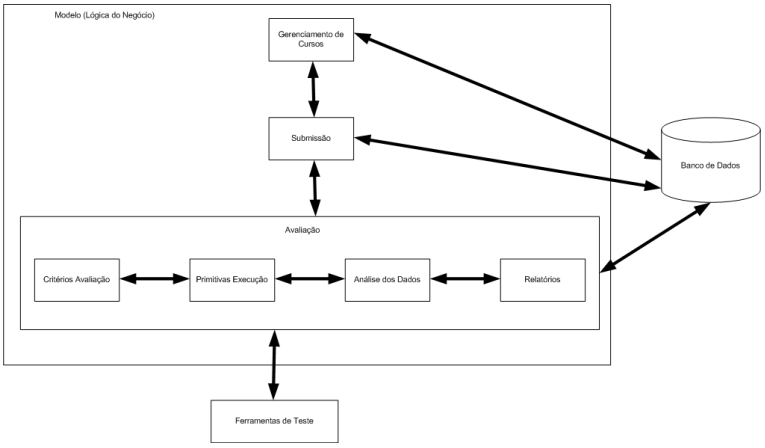

Figure 1. Camada de Modelo do ambiente PROGTEST

O módulo de gerenciamento de cursos é o responsável pelo gerenciamento de usuários (alunos, professores), turmas e trabalhos envolvidos em um determinado curso. O módulo de submissão controla as submissões de trabalhos. O módulo de avaliação é o responsável pelo controle das avaliações dos trabalhos, sendo composto de quatro outros sub-módulos: (1) critérios de avaliação; (2) primitivas de execução; (3) análise dos dados; e (4) relatórios. O módulo de critérios de avaliação é o responsável pela determinação dos critérios de avaliação e os pesos que serão atribuídos aos critérios de teste existentes nas ferramentas de teste, entre outros aspectos. O módulo de primitivas de execução é responsável por compilar os programas e os casos de teste submetidos pelos alunos e depois executá-los em uma ferramenta de teste (neste caso, na ferramenta JaBUTi). No módulo de análise dos dados é realizada uma análise dos resultados obtidos pelo módulo de primitivas de execução levando em consideração os critérios de avaliação definidos no módulo critérios de avaliação. O módulo de relatórios é o responsável por gerar os relatórios de avaliação, tendo como base os resultados da análise dos dados.

3.4. Aspectos Operacionais

A PROGTEST possui dois tipos de usuários: professor e aluno. Cada um desses usuários tem uma visão diferente do sistema, mas com algumas funcionalidades em comum. O professor pode realizar todo o gerenciamento das turmas e dos trabalhos vinculados à ela. Na Figura 2 é mostrado a lista de turmas às quais ele está vinculado onde ele pode pode editar, excluir ou visualizar os dados de uma determinada turma bem como criar uma nova turma.

Na Figura 3 é mostrada a página que contém os dados da turma. No lado esquerdo da página têm-se links para visualizar os alunos matriculados e o quadro geral de notas da turma. A parte principal da página contém a lista de trabalhos já criados para essa turma.

Assim, pode-se editar, excluir, ver as notas da turma para um determinado trabalho já existente. A partir dessa página, pode-se ir para a página de cadastro de novos trabalhos.

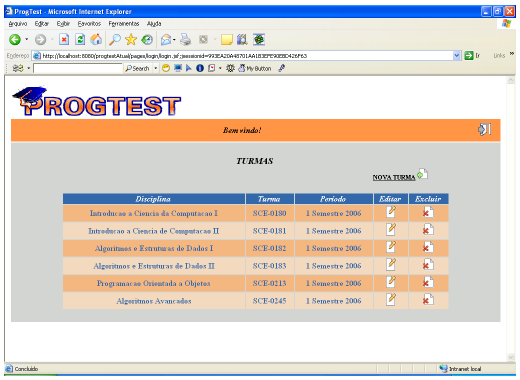

Figure 2. PROGTEST - Lista de Turma Professor

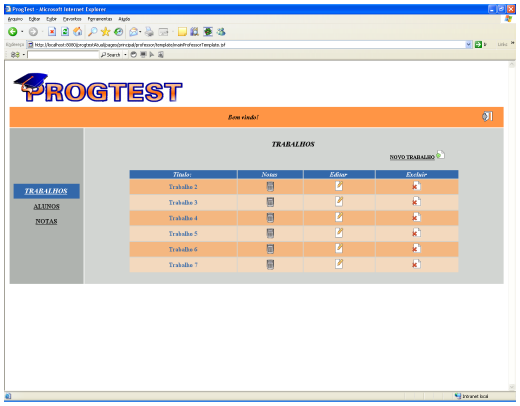

Figure 3. PROGTEST - Lista de Trabalho Professor

Para o cadastro de novos trabalhos é necessário que o professor entre com o título do trabalho, uma breve descrição do que se trata o trabalho, a data inicial que o sistema irá abrir as submissões aos alunos e o data limite do término do trabalho. Cabe ainda ao professor, no momento da criação de um novo trabalho, estabelecer os critérios de avaliação dos trabalhos definindo os pesos que devem ser aplicados aos critérios de teste. Devem ainda ser definidos os pesos que vão gerar a nota sugerida pelo sistema. Esses pesos correspondem ao teste funcional realizado pelo aluno e às execuções. O cadastro dessas informações é mostrado na página da Figura 4.

Cabe ainda ao professor, durante o processo de criação do trabalho, submeter ao sistema a implementação de referência do programa e dos casos de testes referentes ao trabalho. Isso é mostrado na Figura 5 onde tem-se a página que ilustra a submissão da implementação de referência do trabalho. No momento da criação do trabalho o sistema cria a estrutura de diretórios onde serão armazenados os trabalhos submetidos pelos alunos e armazena as informações referentes ao trabalho no banco de dados.

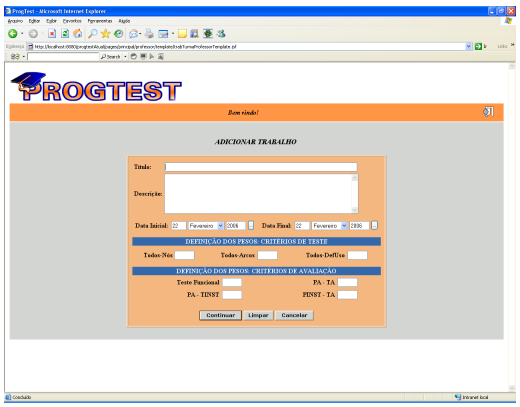

Figure 4. PROGTEST - Cadastro de Trabalhos - Parte 1

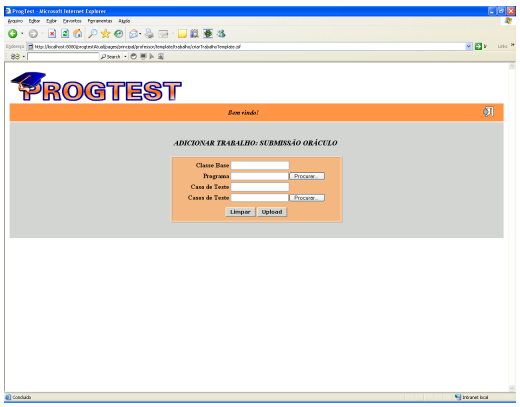

Figure 5. PROGTEST - Cadastro de Trabalhos - Parte 2

Com relação aos alunos, eles podem visualizar as turmas nas quais ele está matriculado, tendo acesso à lista de trabalhos criados para a turma, podendo submeter seus

trabalhos. Na Figura 6(a) é ilustrada a lista de trabalhos de cada turma que os alunos têm acesso. Se um trabalho está aberto para submissões, o aluno pode submeter quantas vezes ele quiser sua solução até o data limite de entrega do trabalho. Primeiramente, o aluno tem acesso aos dados do trabalho e posteriormente à página de submissões onde ele vai fazer o *upload* do seu programa e dos seus casos de teste como mostra a Figura 6(b).

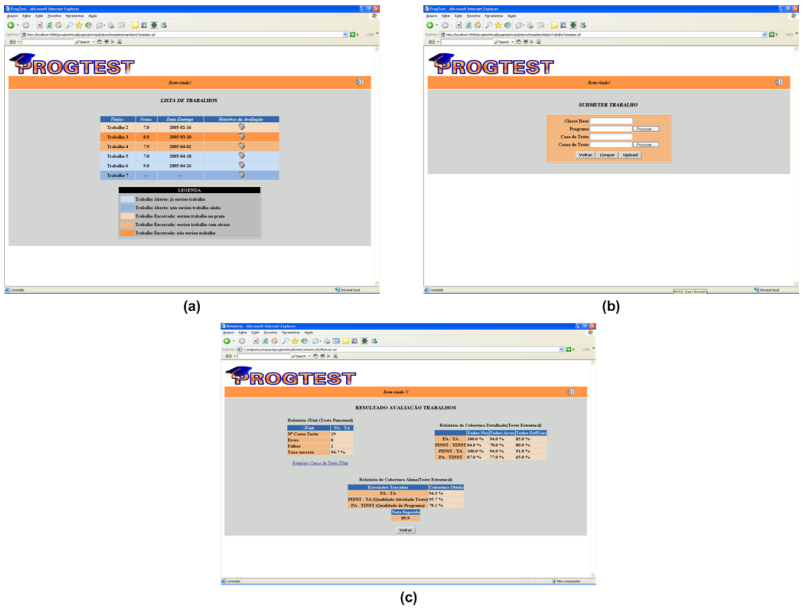

Figure 6. PROGTEST - Submissão e Avaliação de Trabalhos

Logo após a submissão, o aluno pode visualizar o relatório de avaliação do seu trabalho. O resultado da avaliação do trabalho gerado pela PROGTEST (Figura 6(c)) contém três relatórios: (1) relatório do teste funcional; (2) relatório de cobertura do teste estrutural; e (3) relatório com a cobertura geral obtida com as execuções trocadas, cujo valor é calculado com base nos pesos que o professor definiu para os critérios de teste estruturais. Por fim, é apresentada a nota sugerida pelo sistema, calculada com base na cobertura geral obtida com as execuções trocadas e na realização do teste funcional. Por exemplo, considerando a Figura 6(c), na execução do programa do aluno com os casos de teste do professor, o aluno obteve uma taxa de sucesso de 94,7% na realização do teste funcional. No teste estrutural foram obtidas as seguintes coberturas: 87% no critério Todos-Nós; 77% no critério Todos-Arcos; e 65% no critério Todos-DefUsos. A cobertura geral obtida, calculada com base nos pesos definidos pelo professor para os critérios de teste, foi de 78,1%. O relatório mostra ainda a nota sugerida pelo sistema que, nesse caso, é 89.9.

4. Conclusão

Neste trabalho foi investigado a utilização do ambiente para submissão e avaliação automática de trabalhos práticos – ProgTest como mecanismo de apoio ao ensino conjunto de fundamentos de programação OO e teste de software. A principal funcionalidade desse ambiente é aceitar a submissão de trabalhos práticos (programas e casos de teste dos alunos escritos na linguagem Java) e avaliar automaticamente esses trabalhos utilizando, para isso, ferramentas de teste. Foi definida uma estratégia para realizar a avaliação dos

trabalhos dos alunos baseada em critérios de teste. Para colocar em prática essa estratégia, o ambiente foi integrado com a ferramenta de teste JaBUTi.

Um ambiente desse tipo é importante pois avalia todo programa submetido no mesmo nível de eficiência e os resultados da avaliação são baseados nos mesmos padrões. Além disso, é possível reduzir a sobrecarga que existe sobre os professores, uma vez que o sistema avalia automaticamente tanto os casos de teste quanto o código do programa. São ainda fornecidos relatórios, após a avaliação, de cada trabalho dos alunos.

Como trabalhos futuros, ressalta-se a necessidade de aplicar e validar o ambiente PROGTEST, em disciplinas introdutórias de programação. Para isso, um experimento sistemático e controlado vem sendo planejado, envolvendo estudantes de graduação do ICMC/USP. O objetivo é avaliar o uso prático dos mecanismos propostos em cenários reais de aprendizado.

References

Allen, E., Cartwright, R., and Stoler, B. (2002). DrJava: A lightweight pedagogic environment for Java. In *33rd ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'02)*, volume 34, pages 137–141, Cincinnati, Kentucky.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *35th SIGCSE Technical Symposium on Computer Science Education*, pages 26–30, Norfolk, Virginia, USA.

Goldwasser, M. H. (2002). A gimmick to integrate software testing throughout the curriculum. In *33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE'02)*, volume 34, pages 271–275, Cincinnati, Kentucky.

Hickey, T. J. (2004). Scheme-based web programming as a basis for a CS0 curriculum. In *35th SIGCSE Technical Symposium on Computer Science Education*, pages 353–357, Norfolk, Virginia, USA.

Isong, J. (2001). Developing an automated program checker. *The Journal of Computing in Small Colleges (JCSC)*, 16(03):218–224.

Jones, E. L. (2001). Grading student programs - a software testing approach. In *4th Annual Consortium on Small Colleges Southeastern Conference*, volume 16, pages 185–192, Salem, Virginia.

Korhonen, A., Malmi, L., and Saikkonen, R. (2002). Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'02)*, volume 34, pages 55–59, Aarhus, Denmark.

Lahtinen, E., Ala-Mutka, K., and Järvinen, H. (2005). A study of the difficulties of novice programmers. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 14–18, New York, NY, USA. ACM Press.

Massol, V. and Husted, T. (2005). *JUnit em Ação*. Ciência Moderna, 1^a edição edition.

Patterson, A., Kölling, M., and Rosenberg, J. (2003). Introducing unit testing with BlueJ. In *8th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'03)*, Thessaloniki, Greece.

Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assesssment of programming exercises. In *6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'01)*, pages Canterbury, United Kingdom, 133-136.

Shepard, T., Lamb, M., and Kelly, D. (2001). More testing should be taught. *Communications of the ACM*, 44(06):103–108.

Vincenzi, A. M. R. (2004). *Orientação a Objetos: Definição e Análise de Recursos de Teste e Validação*. PhD thesis, ICMC/USP, São Carlos, SP.