Tratando Especificação de Requisitos com Estudantes Iniciantes de Programação

Andréa Mendonça[1], Danielle Chaves[2], Dalton Guerrero[1], Evandro Costa[1]

[1]Coordenação de Pós-Graduação em Ciência da Computação –
Universidade Federal do Campina Grande (UFCG)
Caixa Postal 10.106 – 58.109-970 – Campina Grande – PB – Brasil

[2]Departamento de Sistemas e Computação –
Universidade Federal do Campina Grande (UFCG)

{andrea,danielle,dalton}@dsc.ufcg.edu.br, ebc@gmail.com

Abstract. *In this paper, we report the application of a methodology of programming teaching to novice students that aims to develop the skills for problems understanding and requirements specification. According to this methodology, students are confronted with ill-defined problems and they must specify, with a client, what the program must do. The requirements inspection, using Defects-Based Reading Technique, revealed that the percentage of requirements documented by students was higher than 88% and that the strategy to document test cases minimized the type of defects related to ambiguous and contradictory requirements.*

Resumo. *Neste artigo apresentamos a aplicação de uma metodologia de ensino de programação para iniciantes que tem por objetivo desenvolver as habilidades dos estudantes para entendimento de problemas e especificação de requisitos. Segundo esta metodologia, os estudantes são confrontados com problemas mal definidos e cabe a eles especificarem, junto a um cliente, os requisitos que o programa deve atender para resolver o problema proposto. A inspeção dos requisitos, por meio da técnica Defects-Based Reading, revelou que o percentual de requisitos documentados pelos estudantes foi superior a 88% e que a estratégia de documentar casos de testes minimizou os tipos de defeitos referentes a requisitos ambíguos e contraditórios.*

1. Introdução

Pesquisadores têm advertido a comunidade de educação em computação para as dificuldades dos estudantes com entendimento de problemas e especificação de requisitos, dentre elas, citamos: dificuldades em identificar e eliminar ambigüidades em especificações de software [Blaha et al. 2005], perceber os diferentes níveis de abstração requeridos na resolução de um problema [Hazzan 2008], identificar as informações relevantes para um projeto de software e comunicá-las adequadamente [Eckerdal et al. 2006].

Essas dificuldades, vivenciadas pelos estudantes no decorrer de sua formação, refletem, posteriormente, em sua prática profissional. Conn [Conn 2002] evidencia os problemas que os estudantes enfrentam ao ingressarem na indústria de software e chama atenção para as deficiências com especificação de requisitos.

Embora sejam muitos os relatos que notificam as dificuldades dos estudantes com especificação de requisitos, esta atividade é tratada no currículo dos cursos de computação apenas nas disciplinas de Engenharia de Software ou Análise de Sistemas, geralmente localizadas próximo ao final da graduação. As disciplinas iniciais, por sua vez, focam no desenvolvimento do raciocínio lógico e construção de programas.

Essa estrutura curricular não apenas retarda o desenvolvimento dessas habilidades, como também contraria a forma canônica de resolução de problemas, na qual, primeiramente, entende-se o problema e somente depois soluciona-o.

Preocupados com esta problemática e confiantes de que melhorias na formação podem ser obtidas se a especificação de requisitos for tratada desde o início da graduação, nós concebemos e aplicamos uma metodologia de ensino de programação para iniciantes, denominada *Programação Orientada ao Problema* (POP), que tem por objetivo desenvolver as habilidades para entendimento de problemas e especificação em conjunto com as habilidades de programação.

Neste artigo, nós relatamos a metodologia adotada, sua implementação em uma disciplina de programação para iniciantes e os resultados obtidos com a especificação de um problema mal definido.

2. Programação Orientada ao Problema (POP)

De forma sucinta, POP consiste em propor aos estudantes problemas mal definidos[1] e cabe a eles especificarem, junto a um cliente, os requisitos que o programa deve atender para resolver o problema proposto. Em POP, o cliente assume dupla função: (i) *cliente* para responder aos questionamentos dos estudantes com relação aos requisitos; e (ii) *tutor* para orientar os estudantes na realização das atividades. Em função disso, o chamamos de *cliente-tutor*.

Ao serem confrontados com um problema mal definido, os estudantes devem realizar uma leitura exploratória do enunciado buscando extrair as informações relevantes e identificar aquelas que são ambíguas, contraditórias, irrelevantes, incorretas e/ou informações que foram omitidas. Essa atividade possibilita aos estudantes ter um entendimento inicial do problema, identificar alguns requisitos que o programa deve atender e registrá-los no documento de especificação.

À medida que os requisitos vão sendo identificados, os estudantes devem trabalhar para refinar sua compreensão do problema. Ou seja, eles devem analisar mais detalhadamente os requisitos a fim de descobrir, por exemplo, suas restrições.

Uma das estratégias utilizadas para detalhamento dos requisitos, consiste no estabelecimento de um diálogo entre estudantes e cliente-tutor que vai se tornando, progressivamente, mais estruturado, conforme ilustrado na Figura 1. Inicialmente, os estudantes começam a interação de maneira livre – perguntas e respostas (a), e, posteriormente, vão dialogando na forma de casos de testes de entrada e saída (b) e na forma de *asserts*[2] (c). Em POP, os testes (entrada e saída e *asserts*) são entendidos como uma pergunta que questiona e/ou confirma com o cliente-tutor os requisitos e restrições do problema. Por estarem escritas de forma mais sucinta e estruturada, os testes removem as ambigüidades e contradições, muitas vezes comuns no diálogo em linguagem natural.

[1] Problemas que possuem, intencionalmente, em seu enunciado, ambigüidades, contradições, falta de informações, informações irrelevantes e/ou incorretas.

[2] Um comando da linguagem Python que permite a criação de testes automáticos.

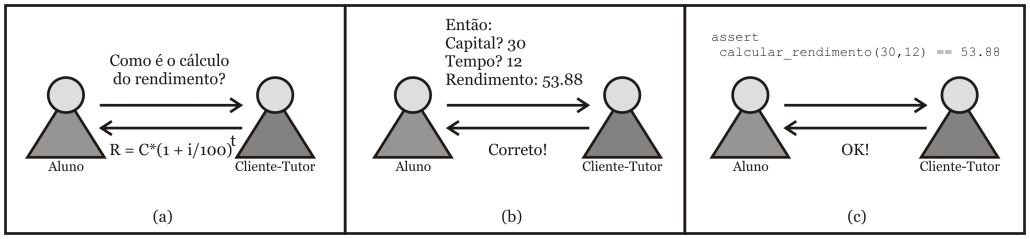

Figura 1. Progredindo do diálogo livre para um diálogo mais estruturado.

É comum que requisitos sejam descobertos e/ou melhor detalhados à medida que os estudantes vão construindo a solução do problema (programa). A implementação contribui para o enriquecimento da interpretação e análise dos requisitos e realimenta as interações para a especificação. Em POP, esse desenvolvimento iterativo e incremental é levado em consideração.

Diferente das metodologias tradicionais de ensino de programação que focam apenas no *espaço da solução*[3], POP propõe o desenvolvimento das habilidades de especificação (*espaço do problema*) em conjunto com as de programação, proporcionando aos estudantes vivenciarem "as idas e vindas" naturais no processo de resolução de problemas. Ao exercitarem as atividades previstas em POP, os estudantes trabalham na produção de três artefatos – documento de especificação, programa e casos de testes. Neste artigo, apresentamos a análise dos documentos produzidos pelos estudantes para a especificação de um problema mal definido.

Essa metodologia tem sido adotada, desde o segundo semestre acadêmico de 2008, na disciplina de programação para iniciantes do curso de Ciência da Computação da Universidade Federal de Campina Grande (UFCG). Python é a linguagem de programação adotada na disciplina.

3. Aplicação de POP com Alunos Iniciantes de Programação

Nesta seção, nós descrevemos a aplicação de POP com os estudantes. Por uma limitação de espaço, apresentaremos apenas um dos problemas propostos, cujo enunciado é descrito na Seção 3.1. Posteriormente, descrevemos a forma como POP vem sendo aplicada na disciplina de programação (Seção 3.2).

3.1. O Problema Proposto

Enunciado. *Em função da crise financeira mundial tem crescido os investimentos na poupança programada, pois é um investimento rentável e com baixíssimo risco. Um professor de administração financeira da UFCG deseja simular investimentos na poupança programada para ensinar seus alunos a driblarem a crise. Faça um programa que atenda a solicitação desse professor, considerando que a poupança rende 5% ao mês, sendo tempo e capital programados pelo investidor com isenção total do imposto de renda.*

Este problema apresenta, intencionalmente, falta de informações, ambigüidades e informações irrelevantes.

Falta de informação Não são informados no problema quais são as entradas, saídas e como elas devem ser formatadas; quais os cálculos que devem ser efetuados e as restrições que o programa deve obedecer. Também não está claro se a poupança tem rendimento fixo;

[3]Aprendizagem de uma linguagem de programação, desenvolvimento do raciocínio lógico e construção de programas.

Ambigüidades A expressão *"sendo tempo e capital programados pelo investidor com isenção total do imposto de renda"* refere-se ao fato de que a poupança programada é um investimento que tem isenção do imposto de renda? ou, que somente pode aplicar na poupança o investidor isento do imposto de renda?

Informações irrelevantes A frase inicial que fala da crise mundial é desnecessária para o entendimento e resolução do problema. Ela foi inserida apenas para deixar o problema mais interessante.

3.2. Aplicação de POP na Disciplina de Programação

POP foi aplicada em duas turmas da disciplina de programação para iniciantes do curso de Ciência da Computação da UFCG. As duas turmas totalizavam 70 alunos, os quais foram divididos em grupos compostos por 5 ± 2 estudantes.

Para cada grupo foi designado um cliente-tutor. Cada cliente-tutor recebeu, previamente, orientações sobre como proceder. Foi entregue a cada cliente-tutor uma *especificação* e um conjunto de *casos de testes de referência*. Com bases neles, os clientes-tutores respondiam aos questionamentos dos estudantes.

O *Google Docs* foi o editor de texto colaborativo utilizado para registrar as especificações dos requisitos. Além disso, listas de discussões foram criadas no *Google Groups* para permitir interações entre estudantes e cliente-tutor fora do horário de aula. Ambos os recursos eram restritos aos membros do grupo e seu respectivo cliente-tutor.

A especificação dos requisitos foi realizada em grupo e a implementação do programa realizada individualmente. Desta forma, os estudantes poderiam reforçar a aprendizagem da sintaxe da linguagem, o desenvolvimento do raciocínio lógico e a construção de programas, ficando livres para definir o *design* do código. Além disso, ao final, os estudantes poderiam verificar como programas construídos de diferentes maneiras atendiam a mesma especificação.

POP foi executada da seguinte forma: primeiramente, cada grupo teve uma reunião presencial de duas horas com o cliente-tutor, nessa reunião foi produzida uma *versão inicial do documento de especificação*. Com base nessa especificação inicial, os estudantes, individualmente, trabalharam na construção de um programa para atendê-la. Ao construírem essa versão do programa, surgiram novos questionamentos que foram tratados via lista de discussão e permitiram atualização do documento de especificação. Um segundo encontro presencial foi estabelecido entre cliente-tutor e estudantes para produzir uma *versão final do documento de especificação*. Neste caso, os estudantes, reunidos com seu respectivo grupo, utilizaram os programas e casos de testes para confirmarem e descobrirem novos requisitos. Após esta data, cada estudante trabalhou na versão final de seu *programa* para atender a especificação. Além disso, produziram um conjunto de casos de testes, escritos em *asserts*, para atestar a qualidade do programa produzido.

4. Avaliação das Especificações Produzidas pelos Estudantes

Nesta seção, apresentamos os procedimentos adotados para análise das especificações, reportamos e discutimos os resultados obtidos.

4.1. Procedimentos

Para avaliar os documentos de especificação produzidos pelos estudantes, nós utilizamos uma técnica de inspeção de requisitos denominada técnica de leitura baseada em defeitos (*Defect-Based Reading* – DBR) [Porter et al. 1995].

Para definir os tipos de defeitos que deveriam ser observados, nós adotamos uma versão simplificada da taxonomia de defeitos[4] definida por Shull, Rus e Basili [Shull et al. 2000] que é apresentada na Tabela 1.

[4]Um *defeito* é uma falha no requisito do sistema que pode levar o *software* a comportar-se de uma maneira indesejada [Shull 1998].

Tabela 1. Taxonomia de Defeitos.

Tipo	Descrição
Informação Omitida	Alguma informação que foi omitida no documento de especificação.
Informação Ambígua	Alguma informação que foi descrita no documento de especificação de forma ambígua, causando duplo sentido.
Informação Contraditória	Duas informações que se contradizem mutuamente ou expressão ações que não podem ser ambas corretas.
Informação Incorreta	Alguma informação que consta no documento de requisitos, mas não descreve corretamente o problema.
Informação irrelevante	Alguma informação que consta no documento de requisitos, mas não é importante, necessária ao entendimento do problema.

Cada documento foi inspecionado por duas pessoas. Na primeira etapa de avaliação, cada inspetor avaliou individualmente os documentos e contabilizou os defeitos observados. Após a avaliação individual, foi realizada uma avaliação conjunta e nos casos em que houve divergência no número e tipo de defeitos identificados, os inspetores entraram em consenso, produzindo uma avaliação final para cada documento de especificação.

4.2. Resultado das Avaliações

Conforme definido no documento de especificação de referência, o programa para resolver o problema proposto (Seção 3.1) teria que atender a 17 requisitos.

Na inspeção dos documentos avaliamos se esses requisitos tinham sido documentados e também se eles tinham sido documentados corretamente, isto é, não apresentavam os tipos de defeitos mencionados na Tabela 1.

A Figura 2 apresenta as porcentagens de requisitos documentados e de requisitos documentados corretamente pelos estudantes das Turmas 1 e 2, respectivamente. A Turma 1 foi dividia em 8 grupos e a Turma 2 em 7. Todos os grupos das duas turmas documentaram mais de 80% dos requisitos. Sendo que 7 grupos documentaram 100% dos requisitos (Grupos 3, 4 e 7 da Turma 1 e Grupos 1, 2, 4 e 7 da Turma 2). Desses grupos que documentaram 100% dos requisitos, apenas o grupo 3 da Turma 1 e o grupo 7 da Turma 2 documentaram todos os requisitos corretamente. No caso do grupo 1 da Turma 2, dos 100% de requisitos documentados, apenas 76,4% deles foram documentados corretamente. Esse grupo apresentou a porcentagem mais baixa de requisitos corretamente documentados. Há casos que merecem destaque, por exemplo, o grupo 3 da Turma 2 documentou apenas 88,2% dos requisitos, sendo que todos eles foram documentados corretamente.

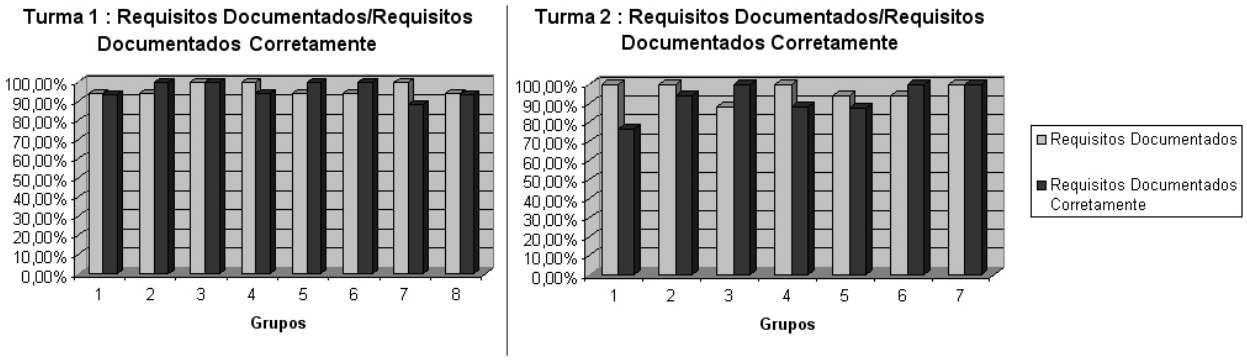

Figura 2. Turma 1 e 2 – Requisitos Documentados e Requisitos Documentados Corretamente.

A Figura 3 apresenta a porcentagem dos tipos de defeitos encontrados nos documentos de especificação das Turmas 1 e 2. Nas duas turmas, os defeitos mais freqüentes foram *informação omitida* e *informação incorreta*. No que diz respeito ao tipo de defeito *informação incorreta*, verificamos que foi comum os estudantes documentarem, erroneamente, o tipo de mensagem que deveria ser exibida em caso de erro na leitura dos valores de entrada, conforme exemplificado no Quadro 1.

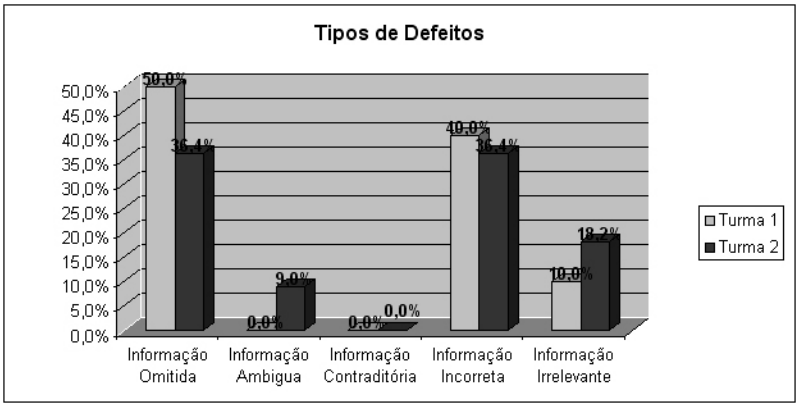

Figura 3. Tipos de Defeitos.

Quadro 1: Turma 1 – Grupo 7: Especificação Incorreta – Mensagem de Erro.

Mensagem que deve ser exibida para entrada inválida de Tempo
Mensagem de erro esperada pelo cliente: Período de tempo de 02 a 48 meses
Mensagem de erro especificada pelo grupo: Tempo mínimo de 2 meses

O registro de casos de testes de entrada e saída no documento de especificação minimizou a porcentagem de outros tipos de defeitos, como por exemplo, *ambigüidades* e *contradições*. Isso ocorreu porque os casos de testes são descritos de maneira mais estruturada e conseguem explicitar vários requisitos de maneira concisa. A Figura 4 mostra como dois casos de testes foram utilizados para especificar diferentes requisitos.

Ainda tratando da especificação de requisitos por meio de testes, a Figura 5 apresenta a porcentagem de requisitos cobertos pelos casos de testes de entrada e saída documentados pelos estudantes. Na Turma 2, essa estratégia foi adotada por apenas 3 grupos, cujos testes de cada grupo cobriram mais de 50% dos requisitos. Na Turma 01, essa estratégia foi mais utilizada e os grupos conseguiram cobrir mais de 70% dos requisitos.

4.3. Discussão

A situação expressa no Quadro 1 evidencia a necessidade de alertar os estudantes para o detalhamento dos requisitos não funcionais. No caso do tipo de defeito *informação omitida*, verificamos que alguns grupos ao interagirem, via lista de discussão, não atualizaram o documento de especificação, confiando no registro que havia sido feita na lista.

Uma dificuldade manifestada pelos estudantes foi a criação dos testes automáticos com uso de *asserts*. Muitos estudantes garantiram a qualidade do código baseando-se nos testes de entrada e saída, descritos no documentados de especificação.

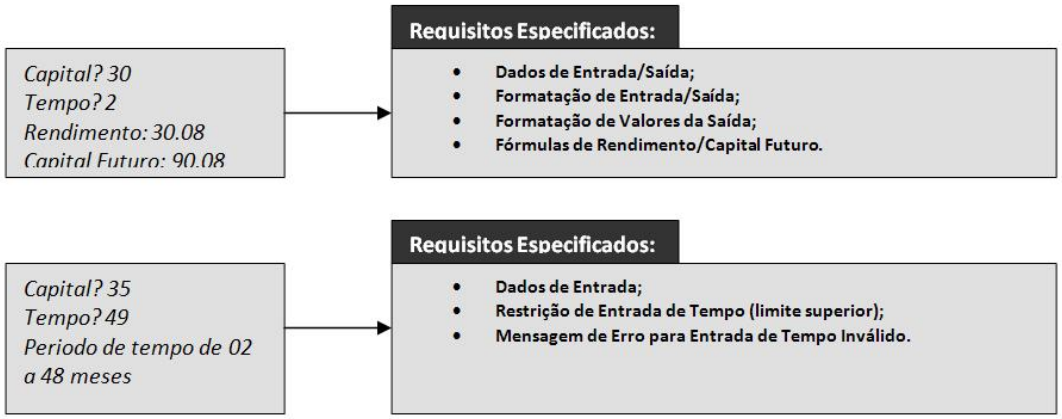

Figura 4. Turma 1 – Grupo 2: Casos de Testes.

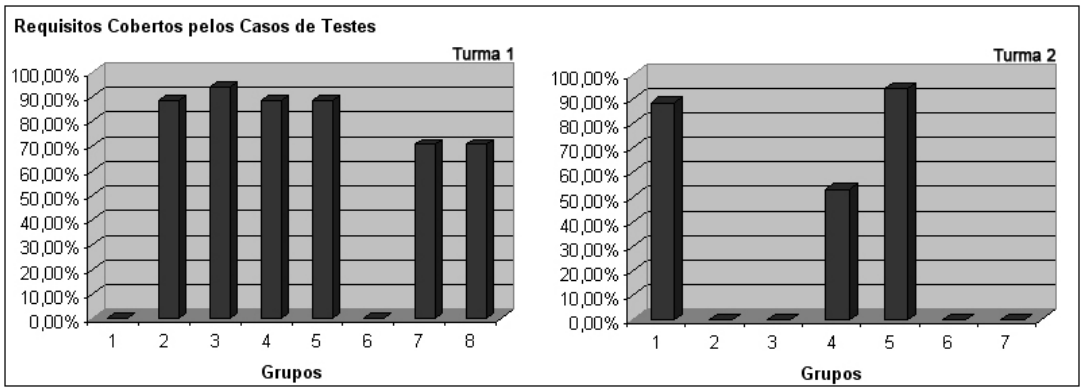

Figura 5. Requisitos cobertos pelos casos de testes de entrada e saída.

Pelos resultados obtidos, nossa avaliação com respeito a POP é de que essa metodologia tem possibilitado resultados promissores no desenvolvimento das habilidades dos estudantes para lidar com problemas mal definidos.

Envolver os estudantes, desde o início da graduação, em atividades de entendimento de problemas e de especificação permite a eles vivenciarem situações mais próximas do mundo real e desenvolverem habilidades que são transferíveis para outras disciplinas do currículo, tais como, Banco de Dados, Análise de Sistemas, Engenharia de Software e Inteligência Artificial.

5. Validade do Estudo e Melhorias de POP

De acordo com a organização da UFCG, alunos repetentes de programação concentram-se em uma turma (Turma 3) a qual recebe assistência mais direcionada. Os resultados reportados nesse artigo sintetizam apenas os dados obtidos das Turmas 1 e 2. Essa organização assegura que a experiência dos alunos repetentes não influenciaram nos resultados reportados.

Para evitar interferências por parte dos clientes-tutores na elaboração do documento de especificação, eles receberam, previamente, as orientações necessárias à aplicação de POP. Estes cuidados aumentam a confiança de que fatores externos não teriam influência sobre os resultados. Várias fontes de dados foram coletadas (programas, documentos de especificação, casos de testes e diálogos) a fim de evitar que julgamentos “subjetivos” interferissem na análise dos resultados.

No que diz respeito à validade externa, esse estudo não teve por objetivo estabelecer generalizações, mas sim adquirir informações que nos permita melhorar a efetividade de POP.

A partir dos resultados obtidos com esse estudo, pretendemos intensificar a utilização de problemas mal definidos na disciplina e melhorar as orientações aos clientes-tutores no que diz respeito a forma de conduzir as atividades em sala de aula e prover *feedback* de acompanhamento aos estudantes.

6. Considerações Finais

Neste artigo, nós apresentamos uma metodologia, denominada *Programação Orientada ao Problema* (POP), e sua aplicação em duas turmas da disciplina de programação para iniciantes do curso de Ciência da Computação da UFCG.

Com a adoção de POP é possível antecipar para o contexto de estudantes iniciantes de programação o desenvolvimento das habilidades para entendimento de problemas e especificação de requisitos. A antecipação destas habilidades tem despertado o interesse da comunidade de Educação em Engenharia de Software que vem discutindo estratégias para minimizar o *gap* entre universidade e indústria de software [Lethbridge et al. 2007]. Colabora também para essa reflexão uma orientação da *Joint Task Force on Computing Curricula* [ACM/IEEE 2004] que diz que muitos conceitos e princípios da Engenharia de Software devem ser ensinados como temas recorrentes durante todo o currículo e a distribuição curricular deve possibilitar a progressiva expansão e aprofundamento desses temas.

Referências

ACM/IEEE (2004). *Software Engineering 2004,Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering*.

Blaha, K., Monge, A., Sanders, D., Simon, B., and VanDeGrift, T. (2005). Do students recognize ambiguity in software design? a multi-national, multi-institutional report. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 615–616, New York, NY, USA. ACM.

Conn, R. (2002). Developing software engineers at the c-130j software factory. *IEEE Softw.*, 19(5):25–29.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., and Zander, C. (2006). Can graduating students design software systems? *SIGCSE Bull.*, 38(1):403–407.

Hazzan, O. (2008). Reflections on teaching abstraction and other soft ideas. *SIGCSE Bull.*, 40(2):40–43.

Lethbridge, T. C., Diaz-Herrera, J., LeBlanc, R. J. J., and Thompson, J. B. (2007). Improving software practice through education: Challenges and future trends. In *FOSE '07: 2007 Future of Software Engineering*, pages 12–28, Washington, DC, USA. IEEE Computer Society.

Porter, A. A., Votta, Jr., L. G., and Basili, V. R. (1995). Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Trans. Softw. Eng.*, 21(6):563–575.

Shull, F., Rus, I., and Basili, V. (2000). How perspective-based reading can improve requirements inspections. *Computer*, 33(7):73–79.

Shull, F. J. (1998). *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. PhD thesis, University of Maryland. Department of Computer Science.