Ensino de Programação para Futuros Não-Programadores: Contextualizando os Exercícios com as Demais Disciplinas de mesmo Período Letivo

Leandro S. G. Carvalho, Bruno F. Gadelha, Fabíola G. Nakamura, David B. F. Oliveira, Elaine H. T. Oliveira

Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octávio, 6200 – Coroado I – CEP: 69077-000 – Manaus – AM – Brasil

{galvao,bruno,fabiola,david,elaine}@icomp.ufam.edu.br

Resumo. *Disciplinas de introdução à programação são usualmente ministradas para cursos de graduação em Engenharia e Ciências Exatas com o propósito de desenvolver nos aprendizes algumas habilidades e competências, tais como: raciocínio lógico, capacidade de resolver problemas utilizando associação, generalização, modularização, entre outras. Muitos dos alunos desses cursos costumam não ter motivação para se aplicar à disciplina, já que a programação não será sua atividade profissional fim. Para trabalhar essa dificuldade, uma equipe de professores concebeu uma metodologia de ensino-aprendizagem que considera o conjunto de disciplinas que estão sendo ministradas no mesmo período letivo. A metodologia foi aplicada em quatro turmas, de cursos e professores diferentes, no primeiro período letivo de 2015. Os resultados preliminares indicam um índice de aprovação maior do que em turmas lecionadas no mesmo período onde a metodologia não foi aplicada. A metodologia foi avaliada pelos discentes que, na maioria, aprovaram a dinâmica de aplicação e as atividades propostas.*

Abstract. *Introductory programming courses are part of undergraduate degree programs in engineering and exact sciences with the goal of developing in the students some skills and competencies, such as logical reasoning, problem-solving by means of association, generalization, modularization, among others. Many students of these programs usually have no motivation to engage in the course, since programming is not directly related to their future professional career. To deal with this issue, a group of professors developed a learning methodology that considers the set of courses taken at the same term. The methodology was applied in four classes in different programs and by different professors, in the first term of the 2015 academic year. The preliminary results indicate that the approval rating among students was higher than that of classes in the same term where the methodology was not applied. The methodology was evaluated by the students that, in most cases, approved the dynamic and the proposed activities.*

1. Introdução

Disciplinas de introdução à programação costumam fazer parte do ciclo básico de formação nos currículos de cursos de graduação em Engenharia e Ciências Exatas. Pela falta de maturidade em perceber a importância da disciplina no exercício da sua

atividade profissional, alunos desses cursos normalmente não têm motivação suficiente para se aplicar à disciplina, o que resulta em altos índices de reprovação na mesma.

Para trabalhar essa dificuldade, uma equipe de professores da Universidade Federal do Amazonas concebeu uma metodologia de ensino-aprendizagem dinâmica, que não se limita à exposição de conteúdo, mas que trabalha com diferentes formas de exercícios. Além disso, na elaboração de tais exercícios, os domínios de problemas trabalhados em cada curso são considerados, bem como as disciplinas que estão sendo ministradas no mesmo período letivo. Essa tarefa de elaborar exercícios relevantes e contextualizados é essencial para o sucesso da disciplina e engajamento dos estudantes, ainda que não seja uma tarefa trivial [Feldman e Zelenski 1996, Stevenson e Wagner 2006, e Hundley e Britt 2009]. O objetivo é estimular os alunos a percebam a importância da disciplina em sua formação integral.

Neste artigo, relatamos a metodologia de ensino-aprendizagem concebida, em grande parte baseada nos trabalhos de Píccolo et al. (2010) e Zanini e Raabe (2012), bem como os resultados. Na próxima seção, contextualizamos a disciplina na universidade em que foi lecionada e na Seção 3 apresentamos os trabalhos relacionados. Em seguida, detalhamos a metodologia de ensino-aprendizagem, aplicada em quatro turmas, de cursos e professores diferentes, no primeiro período letivo de 2015. Na penúltima seção, analisamos os resultados obtidos em termos de índice de aprovação, índice de facilidade e poder de discriminação das questões das avaliações, bem como o *feedback* qualitativo dos discentes. Por fim, propomos trabalhos futuros.

2. Contextualização

A disciplina de Introdução à Programação de Computadores (IPC) é ministrada para cerca de 500 alunos do primeiro período de onze cursos de Engenharia e de Ciências Exatas na Universidade Federal do Amazonas. Cerca de metade dos ingressantes é selecionada por meio do Sistema de Seleção Unificada (Sisu) e a outra metade por meio do Processo Seletivo Contínuo (PSC), realizado pela própria universidade ao longo dos três anos do ensino médio.

Como acontece em outras universidades [Píccolo et al. 2010], por se tratar de uma disciplina de serviço, poucos professores se sentem encorajados a ministrar IPC, de modo que ela é normalmente atribuída a professores substitutos. Entretanto, ensinar programação para alunos iniciantes não é uma tarefa trivial, pois eles vêm com uma base matemática insipiente e estão em cursos em que a Computação não é uma atividade fim.

O gráfico da Figura 1 mostra a evolução da taxa de aprovação na disciplina de IPC, isto é, a razão entre o número de alunos aprovados e o número de alunos matriculados, considerando todas as onze turmas ofertadas em 2015. Os decrescentes índices obtidos de 2011 a 2014, bem como o aumento no número de matrículas, motivaram um grupo de professores a reformular a metodologia de ensino, com o propósito de reduzir a taxa de reprovação por motivo de nota.

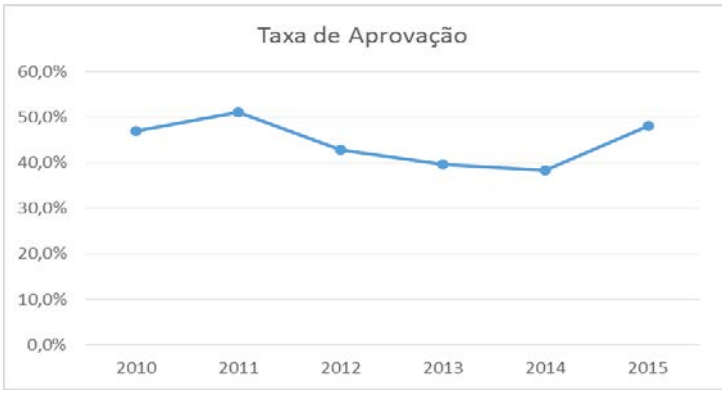

Figura 1 – Evolução da taxa de aprovação na disciplina IPC.

O gráfico apresenta dois pontos destoantes na tendência apontada: os anos de 2010 e de 2015. Em 2010, a Universidade adotou, pela primeira vez, o Sisu como processo seletivo. No Sisu o aluno pode selecionar até 02 (duas) opções de curso, indicando qual a sua ordem de preferência. Tem-se observado que, quando o candidato não possui nota suficiente para ser selecionado em sua primeira opção, ele costuma trocar a ordem de preferência. Só mais tarde, depois de matriculado, ele percebe que o curso não corresponde à sua vocação e o abandona, o que se reflete na taxa de reprovação por frequência. Já no ano de 2015, o aumento na taxa de aprovação pode ser creditado provavelmente à adoção da metodologia aqui apresentada em quatro das onze turmas da disciplina, como veremos nas próximas seções.

O regimento da Universidade não permite que o aluno tranque disciplinas ofertadas para os dois primeiros períodos letivos. Consequentemente, quando o aluno não é aprovado em IPC, ou ele reprova por motivo de nota (quando não atinge média 5, de 0 a 10 pontos), ou por motivo de frequência (quando não frequenta pelo menos 75% das aulas previstas, independentemente do desempenho).

Parte da reprovação pode ser explicada pelo descontentamento do aluno com o curso, o que uma disciplina isolada como IPC não contribui substancialmente. Porém, parte da reprovação pode ser explicada pela falta de uma metodologia de ensino-aprendizagem que os estimulassem a se empenhar.

Para tratar tal problema, fomos buscar na literatura relatos e técnicas que nos fornecessem embasamento para reformular a metodologia de ensino-aprendizagem de formar eficiente e eficaz.

3. Trabalhos relacionados

Segundo Araujo et al. (2013), "no aprendizado de programação, é essencial que os alunos direcionem o seu estudo à codificação e à resolução de exercícios". Os autores identificaram que o estudo regular (diário) é um fator que contribui para o sucesso do aluno na disciplina, mais do que a quantidade de tempo total dedicada ao estudo.

Nesse mesmo sentido, Ihantola et al. (2015) afirmam que "a avaliação contínua durante um curso de programação garante que os estudantes pratiquem bastante, bem como obtenham *feedback* sobre a qualidade de suas soluções". Isso acontece porque a avaliação orienta a aprendizagem dos alunos e serve de *feedback* tanto para o aluno

quanto para o professor sobre o processo de aprendizagem, seja de um tema em específico, seja do curso como um todo.

Contudo, a elaboração de muitos exercícios de programação requer dois tipos de tarefas [França et al. 2011]:

- Gerenciamento de atividades: compreende a elaboração e disponibilização de exercícios, controle da submissão de trabalhos e registro de notas.
- Acompanhamento do aprendizado: compreende a análise de corretude das soluções submetidas e o provimento de *feedback* ao aluno para que este entenda seus erros.

Ambas as tarefas despendem muito tempo do professor e monitores, e não há garantia de que sejam eficientes. Um ambiente virtual de aprendizagem (AVA) típico resolve o problema de gerenciamento das atividades, mas não oferece suporte ao acompanhamento do aprendizado. Para contornar esse problema, muitos cursos de programação lançam mão de sistemas conhecidos como "juízes online".

Juízes online são sistemas que compilam, executam e testam códigos-fonte com base em dados padronizados para julgar se estão corretos. Ihantola et al. (2015) apresentam uma revisão sobre sistemas de avaliação automática – que neste artigo tomamos como sinônimo de juízes online. De acordo com Pieterse (2013), o uso de ferramentas automáticas aumenta a necessidade de uma cuidadosa concepção pedagógica das configurações do exercício e de sua avaliação. Quando os exercícios são avaliados manualmente, é possível contornar problemas causados por enunciados confusos, pontuando-se pela criatividade das soluções. No entanto, esse nem sempre é o caso quando juízes online são utilizados como ferramenta de ensino e avaliação.

Em contrapartida, como apontam Hundley e Britt (2009) e Pieterse (2013), desenvolver exercícios de programação contextualizados e automatizados requer uma atenção especial, a fim de evitar problemas que possam comprometer o êxito dos estudantes. Como apontado por Zanini e Raabe (2012), esse tipo de exercício não é comum, pelo menos em língua portuguesa, o que dificulta a reutilização. Portanto, o docente que se propõe a aderir à risca às premissas aqui colocadas deve dispor de um tempo para planejar questões de enunciado claro, de contextos variados, em diversas versões de mesmo grau de dificuldade para permitir aleatorização, e com exemplos de entradas e saídas que deem um indício do processo de solução. Também requer consulta a livros de Cálculo, Álgebra Linear, Física, Estatística, entre outros, para relembrar conteúdos que já não domina mais, a fim de aplicá-los nos exercícios de programação.

Essa carga de trabalho extra pode ser equivalente ou até mesmo suplantar a carga de trabalho envolvida na correção de atividades no modelo tradicional de ensino de programação [Pieterse 2013]. Para contornar esse efeito colateral, é essencial que uma equipe de professores e tutores divida entre si as tarefas, de modo que, na média, o trabalho seja menor para todos.

4. Descrição da Metodologia de Ensino-Aprendizagem

Com base nos trabalhos discutidos na Seção 3, os autores deste artigo decidiram reformular a metodologia de ensino da disciplina de IPC. Tomou-se como ponto de partida a metodologia proposta por Píccolo et al. (2010), cujos resultados foram bem-

sucedidos em um cenário muito semelhante à da instituição de ensino dos autores. A metodologia adotada apresenta as características descritas nos próximos parágrafos.

A disciplina – que prevê 60 horas de aulas presenciais, sendo 30h de carga teórica e outras 30h de prática – foi remodelada para sete módulos de oito horas (quatro encontros de duas horas), mais um módulo inicial de quatro horas (dois encontros) de motivação e apresentação da disciplina. O conteúdo foi distribuído da seguinte maneira: (1) variáveis; (2) estruturas condicionais; (3) estruturas condicionais aninhadas; (4) estrutura de repetição por condição; (5) vetores; (6) estrutura de repetição por contagem; e (7) matrizes. O oitavo módulo sobre *strings* teve o conteúdo ministrado, mas não foi avaliado. O tópico de "funções" foi ministrado de forma diluída entre os módulos de 5 a 7. Cada módulo tinha quatro aulas de duas horas de duração, na seguinte sequência: aula teórica, laboratório de codificação, laboratório de exercícios, e avaliação parcial. Todas as aulas eram realizadas em laboratório com computadores.

Os exemplos e exercícios trabalhados em cada módulo eram cumulativos. Por exemplo, embora o tópico principal do módulo 6 fosse "estruturas de repetição por contagem", nele eram trabalhados conceitos de vetores e de estruturas condicionais. Por conta disso, as avaliações parciais dos módulos mais avançados tinham maior peso no cômputo da média final na disciplina.

A primeira aula de cada módulo, a aula teórica, era de natureza presencial. Nela, o professor apresentava o conteúdo do módulo à classe, que era estimulada a reproduzir alguns exemplos nos computadores à sua disposição. A segunda e a terceira aula, chamadas de "laboratórios", eram de natureza prática e nelas a presença era facultativa aos alunos. Elas eram acompanhadas apenas pelo tutor, a fim de sanar dúvidas. O desempenho dos alunos nessas atividades contribuía, em menor peso, para a média final na disciplina.

O **laboratório de codificação** era destinado à codificação de problemas cadastrados no juiz online *CodeBench*[1]. Nessa ferramenta, as questões eram corrigidas automaticamente e o aluno recebia *feedback* imediato. Tais questões tinham como objetivo exercitar a habilidade de resolução de problemas, abstração e codificação. O **laboratório de exercícios** era destinado à resolução de questões cadastradas em um AVA semelhante ao *Moodle*[2]. Tais questões eram de cunho mais conceitual ou trabalhavam a habilidade de rastrear código.

A divisão entre laboratório de codificação e laboratório de exercícios tinha finalidade apenas em termos de planejamento. Na prática, os exercícios de ambas as atividades eram liberados para visualização e resolução a partir do dia da aula teórica, com validade até a véspera da avaliação parcial. O aluno ficava à vontade para tirar dúvidas de qualquer uma das atividades durante o tempo de aula com o tutor.

A quarta e última aula do módulo era destinada à avaliação do conteúdo ministrado. As questões da avaliação normalmente eram adaptações de questões trabalhadas no laboratório de codificação. O desempenho dos alunos na avaliação parcial contribuía em maior peso para a média final na disciplina.

---

[1] Disponível em http://codebench.icomp.ufam.edu.br

[2] Disponível em https://moodle.org/

Adotou-se a linguagem Python, pois sua sintaxe é de simples compreensão, exige que o programador organize o código respeitando indentação, possui material didático em português, além de outras vantagens [Menezes 2014]. Adotou-se a IDE Spyder, pelas seguintes características: tem versões para vários sistemas operacionais; possui destacador de sintaxe; incorpora as bibliotecas Numpy e MathPlotLib em seu instalador; e inclui um visualizador de estado das variáveis (critério decisivo quando se lida com aprendizes com baixa capacidade de abstração).

5. Resultados

Nesta seção, apresentamos a taxa de aprovação nas turmas em que a metodologia foi aplicada e nas turmas em que não foi; o índice de facilidade e o índice discriminação das questões elaboradas para as avaliações; e uma análise qualitativa das opiniões dos alunos sobre a metodologia.

A Tabela 1 apresenta os cursos para os quais a disciplina IPC foi ministrada em 2015, destacando-se as quatro turmas que participaram da metodologia. Dos 172 alunos matriculados nelas, 61,6% foram aprovados. Dos 319 alunos matriculados nas outras seis turmas, 40,8% foram aprovados. Essa diferença de mais de 20% parece indicar que a metodologia foi bem-sucedida, pelo menos na sua primeira aplicação, em 2015.

Em seguida, analisamos se essa diferença poderia ser explicada por um enviesamento na atribuição de turmas com alunos melhor preparados no Ensino Médio aos professores que adotaram a metodologia. Verificou-se, no entanto, que a correlação tau de Kendall (pois trabalhou-se com um pequeno conjunto de dados) entre a nota de corte no Sisu e a taxa de aprovação em IPC, independente da metodologia aplicada, resultou em um valor em torno de 0,6 (em uma escala de -1 a +1). Esse resultado sugere, portanto, que, há uma forte correlação positiva entre a nota de corte no Sisu e a taxa de aprovação em IPC que independe da metodologia aplicada. Desse modo, a maior taxa de aprovação pode ser creditada à adoção da metodologia, pelo menos no ano de 2015.

Tabela 1 – Cursos de graduação para os quais foi ministrada a disciplina de Introdução à Programação de Computadores em 2015.

Curso	Nota de corte SISU	Nº de matrículas	Nº de aprovados	Percentual de aprovados
Engenharia Química	703,6	61	43	70,5%
Engenharia Mecânica	694,0	54	43	79,6%
Matemática Bacharelado Diurno	645,3	17	7	41,2%
Matemática Licenciatura Matutino	625,6	40	13	32,5%
Engenharia de Petróleo e Gás	703,8	50	28	56,0%
Engenharia de Produção	682,9	52	33	63,5%
Engenharia de Materiais	666,8	47	23	48,9%
Física Bacharelado Diurno	657,6	25	3	12,0%
Estatística	630,9	56	20	35,7%
Matemática Licenciatura Noturno	626,4	57	18	31,6%
Física Licenciatura Diurno	615,1	32	5	15,6%

Após o fechamento das turmas, analisamos a adequabilidade das questões com relação ao índice de facilidade e o poder de discriminação. Esses critérios são adotados pelo INEP no Exame Nacional de Desempenho dos Estudantes (ENADE) para avaliar as questões propostas em cada exame.

Para o INEP, o índice de facilidade de uma questão está diretamente relacionado com seu percentual de acerto. Questões com baixo percentual de acerto são consideradas difíceis, enquanto as que possuem alto índice de acerto são consideradas fáceis. A Tabela 2 mostra os índices de facilidade adotados pelo comitê do ENADE.

Tabela 2 – Classificação de questões segundo o índice de facilidade. Fonte: INEP (2014)

Índice de Acerto	Classificação
Acima de 86%	Muito fácil
61% a 85%	Fácil
41% a 60%	Médio
16% a 40%	Difícil
Abaixo de 15%	Muito difícil

A Tabela 3 apresenta algumas características das questões elaboradas para a avaliação dos alunos, incluindo o contexto explorado pelo enunciado de cada uma delas. A quarta coluna mostra os índices de facilidade das questões. Relacionando a Tabela 3 com a Tabela 2, percebemos que a maioria das questões foi classificada como fácil ou mediana. Esse alto nível de facilidade é um reflexo da contextualização dos enunciados com as experiências e vivências do aluno. Apenas a questão 6 foi classificada como uma questão difícil, pois envolvia conceitos mais avançados de geometria, que muitos alunos tiveram dificuldade de resolver.

Tabela 3 – Questões usadas para avaliação dos alunos.

Avaliação	Questão	Contexto	Índice de Acerto	Valor de r_{pb}
1	A	Comercial: percentual de gorjeta	77%	0,38
	B	Geometria: volume de recipiente e custo	64%	0,38
2	A	Escolar: aprovação em disciplina	79%	0,37
	B	Matemática: aritmética	67%	0,33
3	A	Físico-química: estado físico de elementos	81%	0,34
	B	Geometria: volume de um sólido não trivial	21%	0,34
4	A	Cálculo: aproximação de séries	77%	0,43
	B	Financeira: aplicação bancária	56%	0,46
5	A	Cálculo: derivada de polinômios	68%	0,40
	B	Estatística: identificação de padrão em vetor	74%	0,39
6	A	Estatística: contagem de ocorrências em vetor	57%	0,45
	B	Frações contínuas	71%	0,43
7	A	Álgebra Linear: sistema de equações lineares	55%	0,39
	B	Álgebra Linear: manipulação de matrizes	51%	0,42

Outra métrica adotada pelo INEP é o poder de discriminação. Dizemos que uma questão possui alto poder de discriminação quando ela é capaz de diferenciar alunos de bom desempenho dos alunos de mau desempenho. A capacidade de discriminação de uma dada questão pode ser quantificada através do coeficiente de correlação ponto-bisserial, que mede a correlação entre uma variável dicotômica (acerto/erro da questão) e uma variável numérica (nota final do aluno). Esse coeficiente é dado por:

$$r_{pb} = \frac{M_Q - M_T}{DP_T}\sqrt{\frac{p}{q}}$$

onde M_Q é a média das notas finais dos alunos que acertaram a questão Q, M_T é a média das notas finais de todos os alunos, DP_T é o desvio padrão das notas finais de todos os alunos, p é o percentual de alunos que acertaram a questão Q, e q é a proporção de alunos que erraram a questão Q ($q = 1-p$).

Tabela 4 – Classificação de questões segundo o poder de discriminação. Fonte: INEP (2014)

Valor de r_{pb}	Classificação
Acima de 0,40	Muito bom
0,30 a 0,39	Bom
0,20 a 0,29	Médio
Abaixo de 0,19	Fraco

A Tabela 4 apresenta as classes de poder discriminatório adotadas pelo INEP. Relacionando-a com a última coluna da Tabela 3, que apresenta os valores de correlação ponto-bisserial alcançadas pelas questões das avaliações em IPC, podemos observar que todas as questões obtiveram um índice de discriminação bom ou muito bom. Dessa forma, podemos concluir que a probabilidade de um aluno ter acertado uma questão esteve diretamente relacionada com seu domínio sobre o tópico de programação explorado na questão, e não com um efeito aleatório.

Ao final da disciplina, foi solicitado que os alunos participassem de uma pesquisa para avaliar a metodologia utilizada através do preenchimento de um questionário. Esse questionário consistia de perguntas sobre o entendimento e a dificuldade na realização das diversas atividades propostas, o grau de interação com professores e tutores, e a opinião do aluno acerca da metodologia utilizada. Neste artigo, serão apresentados somente os resultados relacionados às atividades e à metodologia.

Para analisar os comentários dos participantes nas perguntas abertas, foi usada a técnica descrita no Método de Explicitação do Discurso Subjacente (MEDS) [Nicolaci-da-Costa 2007]. Foi realizada uma análise inter-participantes onde suas respostas foram sistematicamente comparadas em busca de recorrências, emergindo assim as categorias. Parte delas é apresentada na Tabela 5.

Sobre as atividades realizadas (categorias 1, 2, 3 e 4), os alunos perceberam a relação entre o conteúdo ministrado nas aulas presenciais com as atividades cobradas na avaliação. Destacaram também o nível de exigência de cada atividade, como por exemplo, na citação "Estavam na dificuldade adequada: não eram extremamente difíceis, mas podiam ser respondidos com estudo e conhecimento". A maioria dos alunos também destacou a clareza nos enunciados das atividades e a importância das

dicas oferecidas para exemplificar o que se pretendia com a atividade. Porém, alguns alunos discordaram e afirmaram ter problemas com a interpretação de algumas atividades, como ilustram as citações: “Algumas questões eram claras, mas as mais difíceis não eram claras” e “O objetivo da questão era nos confundir, e o objetivo foi realizado com sucesso”. Ainda com respeito às atividades, os alunos perceberam que ao avançar no curso, as atividades ficavam mais complexas como pode se observar na citação: “... apesar das aulas terem sido ministradas com clareza, achei as questões do lab em um nível de dificuldade muito acima do nível da aula” e “... foram ficando cada vez complexos...”.

Tabela 5 – Categorias que emergem da análise inter-participantes do MEDS.

#	Categoria	Ocorrências
1	Coerência das atividades com conteúdo ministrado	24
2	Clareza nos enunciados das atividades	70
3	Dicas nos enunciados das atividades	7
4	Complexidade das atividades	33
5	Qualidade do material de apoio	12
6	Disponibilidade dos sistemas e material de apoio	11
7	Quantidade de exemplos de teste	9

Sobre a qualidade do material de apoio (categoria 5) disponibilizado aos alunos, foi relatado que eram satisfatórios, de linguagem de fácil entendimento e que realmente ajudavam na realização das atividades como relatado na citação: “Havia tipo um "tutorial" explicando detalhadamente questões semelhantes”. A disponibilidade tanto do material de apoio quanto das ferramentas de suporte na disciplina (categoria 6) foi outro ponto de destaque nos relatos dos alunos, que afirmavam: “Porque foi possível acessar os arquivos de apoio no AVA” e “porque podia consultar a apostila e o Spyder”.

Com respeito às atividades de codificação, os alunos destacaram a quantidade de exemplos para testes dos scripts propostos (categoria 7). Alguns alunos relataram dificuldades no entendimento dos enunciados e apontaram a falta de exemplos para os testes, como pode-se observar na citação: “Achei que faltavam mais exemplos para testar o programa”. Outros não relataram a mesma dificuldade, como observa-se na citação: “Por que eram bem explicativos e no material disponibilizado no AVA continha muitos exemplos”.

6. Considerações finais

Aqui foi apresentado um relato de experiência da aplicação de uma metodologia de ensino-aprendizagem em turmas de Introdução à Programação de Computadores. Os resultados preliminares indicaram um índice de aprovação maior que em períodos anteriores. Além disso, as questões interdisciplinares elaboradas se mostraram efetivas tanto pelo seu poder discriminatório quanto pela avaliação dos próprios discentes.

Como trabalhos futuros, desejamos fazer uso de Inventários de Conceito [Caceffo et al. 2016] para aferir o ganho de aprendizagem antes e depois da disciplina. Além disso realizaremos o cruzamento mais detalhado dos dados deste estudo com os de disciplinas concomitantes, a fim de diferenciar entre alunos que se afastam informalmente de todas as disciplinas do curso daqueles que se afastaram apenas da disciplina de IPC.

Referências

Araujo, E. C.; Gaudencio, M.; Menezes, A.; Ferreira, I.; Ribeiro, I.; Fagner, A.; Ponciano, L.; Morais, F.; Guerrero, D. S.; Figueiredo, J. A. (2013). O papel do hábito de estudo no desempenho do aluno de programação. In *Workshop sobre Educação em Computação (WEI 2013)*. Maceió.

Caceffo, R.; Wolfman, S.; Booth, K. S.; Azevedo, R. (2016). Developing a Computer Science Concept Inventory for Introductory Programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (SIGCSE '16). ACM, New York, NY, USA, 364-369.

Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. Relatório Síntese da Área de Ciência da Computação – Enade (2014). Disponível em http://portal.inep.gov.br/enade/relatorio-sintese-2014

Feldman, T. J.; Zelenski, J. D. (1996). The quest for excellence in designing CS1/CS2 assignments. In *ACM SIGCSE Bulletin* (Vol. 28, No. 1, pp. 319-323). ACM.

França, A. B.; Soares, J. M.; Gomes, D. G.; Barroso, G. C. (2011). Um sistema orientado a serviços para suporte a atividades de laboratório em disciplinas de técnicas de programação com integração ao ambiente Moodle. Revista Novas Tecnologias na Educação – RENOTE, 9(1).

Hundley, J.; Britt, W. (2009). Engaging students in software development course projects. In *Proceedings of the Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations* (pp. 87-92). ACM.

Ihantola, P.; Ahoniemi, T.; Karavirta, V.; Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 86-93).

Menezes, N. N. C. (2014). Introdução à Programação com Python. Editora Novatec, 2ª edição.

Nicolaci-da-Costa, A. M. (2007). O Campo da Pesquisa Qualitativa e o Método da Explicitação do Discurso Subjacente (MEDS). In *Psicologia: Reflexão e Crítica*. vol.20 no.1. ISSN: 0102-7972. RS, Porto Alegre.

Píccolo, H. L.; Sena, V. F.; Nogueira, K. B.; Silva, M. O.; Maia; Y. A. N. (2010). Ambiente Interativo e Adaptável para ensino de Programação. In *Workshop sobre Educação em Computação*, pp. 555–566.

Pieterse, V. (2013). Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference* (pp. 45-56). Open Universiteit, Heerlen.

Stevenson, D. E.; Wagner, P. J. (2006). Developing real-world programming assignments for CS1. In *ACM SIGCSE Bulletin* (Vol. 38, No. 3, pp. 158-162).

Zanini, A. S.; Raabe, A. L. A. (2012). Análise dos enunciados utilizados nos problemas de programação. In *Workshop sobre Educação em Computação*, pp. 555–566. Curitiba, PR.