Aprende – um Ambiente Cooperativo de Apoio à Aprendizagem de Programação

Thais Helena Chaves de Castro[1], Alberto Nogueira de Castro Júnior[1], Crediné Silva de Menezes[2]

[1]Departamento de Ciência da Computação – Universidade Federal do Amazonas (UFAM)
Av. Gal. Rodrigo O. J. Ramos, 3000, Bloco N, Mini-Campus
69.077-900 Manaus-AM-Brasil

[2]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, s/n, Centro Tecnológico, Bloco VII
29.060-970 Vitória-ES-Brasil

{thais, albertoc}@dcc.fua.br, credine@inf.ufes.br

Resumo

Este trabalho visa descrever o processo de levantamento de requisitos para a construção de um ambiente cooperativo de apoio à aprendizagem de programação. O processo de levantamento de requisitos ocorreu na Universidade Federal do Amazonas e foi realizado por meio de um experimento prospectivo, o qual utilizou, como sujeitos da aprendizagem, alunos do primeiro período dos cursos de Ciência e Engenharia da Computação.

Abstract

This work describes the requirements acquisition for the construction of a cooperative environment to support the apprenticeship of programming. This process has taken place at Universidade Federal do Amazonas and has been effectively done by a prospective experiment, which has considered, as apprenticeship subjects, first year students from the majors of Computer Science and Computer Engineering.

Palavras-chave: levantamento de requisitos, programação funcional, ensino-aprendizagem.

1. Introdução

Já é um consenso entre muitos membros da comunidade acadêmica (vide discussão em listas e fóruns apropriados) que aprender a programar é uma atividade extremamente difícil, especialmente àqueles que ingressam em um curso de graduação na área de Computação e Informática e são inseridos em uma turma heterogênea, onde alguns estudantes são provenientes de cursos técnicos na área e outros são exímios programadores que, pela ausência de uma educação formal na área em questão, codificam sem saber ao certo o que estão fazendo.

Devido a essa heterogeneidade algumas instituições de ensino superior utilizam o paradigma de programação funcional nas disciplinas do primeiro período, incluídos aqui os cursos de Ciência da Computação e Engenharia da Computação da Universidade Federal do Amazonas e da Universidade Federal do Espírito Santo.

Temos trabalhado em cursos introdutórios nas instituições citadas anteriormente utilizando o paradigma de programação funcional há mais de 12 anos, mas apesar de termos realizado uma anotação a respeito do grau de dificuldade dos estudantes em cada tópico, não possuíamos informações suficientes sobre a adequabilidade do paradigma adotado ao objetivo da disciplina. O que impossibilitou o desenvolvimento de um ambiente de apoio à aprendizagem

de programação, pois não pudemos realizar um levantamento de requisitos baseado em um experimento com uma turma inteira de estudantes iniciantes.

Portanto, apesar de já termos relatado algumas evidências de uma melhoria na aprendizagem de programação com a adoção do paradigma funcional e os tipos de problemas com que trabalhamos [CASTRO et al, 2002], faltava um resultado mais concreto. Ao realizarmos o experimento descrito neste artigo, conseguimos inferir alguns aspectos que ainda não tinham sido considerados com respeito à aprendizagem e, com isso, possibilitou-nos realizar o levantamento de requisitos que desejávamos e, com isso, propormos o desenvolvimento de um ambiente computacional de apoio à aprendizagem de programação.

Um outro fator motivador para a realização do experimento foi o fato de não termos conhecimento de resultados de experimentos com turmas de primeiro período que pudessem fornecer subsídios para se inferir um crescimento significativo no número de estudantes com desempenho satisfatório nas disciplinas iniciais ou mesmo uma redução na taxa de evasão desses cursos.

A fim de compartilharmos nossa experiência como facilitadores da aprendizagem de programação e propormos um ambiente para apoio cooperativo à aprendizagem, iniciamos descrevendo resumidamente, na Seção 2, em que parâmetros o curso de introdução à programação que ministramos em nossas instituições são apoiados. Na Seção 3, relatamos como foi realizado o experimento, de que forma analisamos as informações obtidas e concluímos a Seção com a especificação dos requisitos para um ambiente computacional. Prosseguindo nessa linha, na Seção 4 fornecemos uma descrição do ambiente computacional resultante do levantamento de requisitos da Seção 3, considerando os elementos implementados em um protótipo de ambiente, desenvolvido previamente pelo grupo.

2. Experiência com Aprendizagem de Programação

O objetivo principal de um curso de introdução à programação é que os estudantes aprendam a resolver problemas e depois representem a solução em um programa, seja em linguagem funcional, imperativa ou lógica.

Como citado na Seção anterior, utilizamos o paradigma de programação funcional em nossos cursos introdutórios de programação. Dentre os muitos argumentos que podemos enumerar para a utilização deste paradigma, destacamos: (i) o paradigma funcional é baseado em conhecimento já familiar ao estudante desde o ensino médio (funções e mapeamentos entre domínios); (ii) permite concentrar a atenção na elaboração de soluções e nas descrições formais delas; (iii) o foco da atenção está em "o que fazer", ao invés de "como fazer"; (iv) por dispensar conhecimento dos detalhes de um computador específico, possibilita um número maior de experimentações e aumento da produtividade.

Em nossas atividades utilizamos a linguagem Haskell [THOMPSON, 1999], uma linguagem funcional pura e não restrita. Para as aulas de laboratório, utilizamos uma implementação do Haskell, o ambiente HUGS [JONES et al, 2004] para a plataforma Linux, que é o sistema operacional instalado no laboratório básico dos cursos mencionados.

Além da adoção de um paradigma de programação específico, procuramos elaborar exercícios que sejam comuns ao cotidiano dos estudantes, ao mesmo tempo em que são interessantes. Por exemplo, ao abordarmos o uso de listas e recursão elaboramos uma lista de exercícios somente com o jogo de dominó[1], onde a cada exercício pedimos que os estudantes elaborem funções cada vez mais complexas.

[1] O jogo de dominó que estamos comentando é o que é jogado no norte do país, onde a soma das pedras é relevante e de onde se pode trabalhar com todas as pontas (as laterais também).

Um outro aspecto a considerar é a habilidade de raciocínio lógico, que ajuda o estudante a enxergar melhores caminhos para chegar às soluções dos problemas. No início do curso, trabalhamos o método de resolução de problemas elaborado por Polya [POLYA, 1988] e depois fazemos um paralelo com problemas computáveis. Segundo este método, a idéia geral para se resolver um problema é primeiramente distanciar-se dele, e só então tentar dividi-lo em partes menores, relacionando-as a outros problemas já resolvidos. Esse distanciamento requer uma certa prática e nossa experiência tem mostrado que os estudantes que ingressam nas universidades não estão acostumados a pensar dessa maneira por não terem o hábito de sistematizar seus próprios métodos de resolução de problemas.

3. O Processo de Levantamento de Requisitos

Como resultado de pesquisa anterior dos professores das instituições envolvidas, foi desenvolvida uma arquitetura, chamada SAAP [CASTRO et al, 2002b], que se propunha a atenuar algumas deficiências na aprendizagem de programação, levantadas informalmente com alguns estudantes. A arquitetura SAAP utiliza esquemas de programas escritos em uma metalinguagem de programação como base para comparar estilos de programação em programas desenvolvidos pelos estudantes. Isso para que se possa fornecer o *feedback* adequado a cada classe de problema.

Embora a solução encontrada para representar esquemas de soluções seja eficiente, sentiu-se que os estudantes não se sentiam à vontade para confiar plenamente no ambiente, uma vez que não havia meios de assegurar que o esquema proposto por alguém estivesse adequado ou, ainda, se um estudante que estivesse em um nível de desenvolvimento cognitivo mais avançado desejasse colaborar com o ambiente não poderia, pois caso fosse autorizado haveria o risco de sua solução não estar adequada.

Tendo isso em vista, elaboramos um experimento controlado com duas turmas de estudantes calouros da Universidade Federal do Amazonas, ingressantes nos cursos de Ciência e Engenharia da Computação. Com os alunos de Ciência da Computação, utilizamos como *setting* duas disciplinas: Introdução à Computação, que se propões a ensinar programação utilizando o paradigma de programação funcional; e Construção do Conhecimento, cujo principal objetivo é estudar maneiras de representação e resolução de problemas, com especial atenção à solução criativa de problemas, com base em [GOLEMAN et al, 2001]. Como os alunos de Engenharia da Computação não possuem Construção do Conhecimento em seu currículo, utilizamos como *setting* apenas a disciplina Introdução à Computação, fornecendo o material de resolução de problemas como atividades complementares. A descrição detalhada das atividades planejadas encontra-se na Tabela 1.

Conforme as inadequações detectadas com o uso do protótipo GesTo e as necessidades identificadas pelos professores, no que concerne a sistematizar os processos de aprendizagem de programação nos quais os alunos iniciantes estão inseridos, o experimento realizado pretendia atingir os seguintes objetivos:

- ✓ Avaliar pelo menos um aspecto dentre aqueles considerados como benefício do uso do paradigma funcional – por exemplo, a segurança (autoconfiança) dos alunos no que diz respeito à habilidade na especificação de solução de problemas, visto que ao serem utilizadas estratégias de resolução de problemas, quebra-se a complexidade da solução, tornando-se simples de ser especificada em qualquer linguagem, quer seja uma linguagem puramente matemática ou de programação funcional.
- ✓ Avaliar as dificuldades no que diz respeito à transição do paradigma funcional para o procedimental (no caso dos cursos da UFAM, o paradigma de programação estruturada).

- ✓ Levantar os insucessos com as abordagens, possíveis causas e soluções.

Tabela 1 - Planejamento de Atividades para o Experimento

Descrição do *Setting*	Atividades Previstas
Três turmas da disciplina IC, oferecidas para o primeiro período dos cursos de Bacharelado em Ciência da Computação e Engenharia da Computação.	Registrar a evolução do estudante no que diz respeito às habilidades para resolução de problemas e especificação de soluções. De modo a evitar interferência com *avaliação* do conhecimento construído, sugere-se o uso de instrumentos para medida de atitude (questionários, entrevistas, observação direta, etc) no que diz respeito à segurança do estudante com respeito às habilidades específicas anteriormente mencionadas, aliado à análise de cenários (pelo menos três – ao início, meio e final da disciplina) com problemas *clássicos* de computação.
Uma turma da disciplina Construção do Conhecimento, oferecida para alunos do 1º. Período de Bacharelado em Ciência da Computação, e alguns alunos de outros períodos do curso.	Uma vez que a maioria dos estudantes seria alunos calouros, busca-se identificar os tipos de dificuldades percebidas pelos próprios estudantes, com respeito à metodologia da pesquisa científica, que aborda todas as etapas de resolução de problemas científicos. Isto para que se consiga identificar pontos de ruptura no processo de resolução de problemas de naturezas variadas e se consiga estabelecer um paralelo com o raciocínio e técnicas para resolução de problemas de computação.

3.1 Procedimentos Experimentais

Como é uma necessidade para os processos de organização e análise dos dados coletados, fizemos um levantamento de informações junto aos estudantes, utilizando ferramentas e métodos próprios da análise qualitativa. Este tipo de análise é utilizado, por exemplo, para proceder verificação de consistência através de referências cruzadas ao se utilizar diferentes instrumentos com cada participante, ou estratificação de termos-chave e até relacionamentos entre conceitos utilizados nos documentos.

Uma vez realizado o planejamento dos instrumentos a serem utilizados no experimento, partiu-se a segunda etapa: preparação do conteúdo a ser trabalhado, parcialmente baseado na apostila [MENEZES et al, 2004] desenvolvida pelo nosso grupo de pesquisa. Nesta fase, precisamos também elaborar maneiras de confrontar os diversos instrumentos a serem utilizados.

Os principais instrumentos efetivamente utilizados para coleta de dados foram:

- ✓ Auto-avaliações, em forma de entrevistas;
- ✓ Exercícios de lógica;

- ✓ Debate sobre Programação Funcional, utilizando o método cooperativo da controvérsia acadêmica;
- ✓ Questionário;
- ✓ Provas e listas de exercícios, contendo questões-chave para a avaliação do conhecimento adquirido.

Inicialmente, foi aplicado um questionário inicial, com o objetivo de conhecer o *background* dos alunos, quanto à familiaridade com programação e representações de funções. A compilação das respostas reforçou nossas hipóteses: (a) mais da metade dos alunos, apesar de terem familiaridade com computador, nunca programou; (b) aqueles que já haviam feito um curso técnico em informática, nunca tinham estudado programação funcional; e (c) quase a totalidade apresentou indícios de possuírem conhecimentos rudimentares em manipulação de funções matemáticas.

Essa avaliação inicial confirmou nossas previsões quanto ao *background* dos alunos iniciantes, o que ajudou a sustentar o argumento de que programação funcional era eficiente para uma primeira experiência com programação.

Uma vez estabelecidas as premissas a, b e c, foi elaborada uma primeira avaliação, contendo problemas que exploram o raciocínio lógico-dedutivo. Pretendíamos, com esta avaliação, entender como os alunos resolviam problemas e se eles conseguiam modularizá-los, para só depois chegarem a uma solução. Com a análise das respostas dos alunos percebemos uma dificuldade em modularizar os problemas. Portanto, começamos a trabalhar mais profundamente os métodos de resolução de problemas, em especial o de Polya.

À metade do curso, após termos trabalhado o tópico "estruturas condicionais", realizamos uma entrevista, onde os alunos deveriam avaliar o seu aprendizado até o momento, identificando possíveis causas de insatisfação em relação à disciplina ou aos professores. Entendemos que este instrumento, por ser de caráter intrinsecamente subjetivo, não poderia fornecer parâmetros confiáveis. Por este motivo, precisamos confrontar as respostas dos alunos na entrevista com suas respostas aos problemas da prova, que foi realizada na aula anterior.

As informações que conseguimos extrair a partir desse confronto entre os instrumentos de coleta de dados foram: (*i*) cerca de metade da turma apresentava dificuldades ao resolver os exercícios por não conseguirem enxergar um caminho de solução; (*ii*) apesar de os alunos terem familiaridade com geometria, apresentavam dificuldades em resolver problemas ilustrados neste domínio; (*iii*) os alunos provindos de cursos técnicos em informática, possuíam dificuldades em expressar-se em Haskell; e (*iv*) alunos que nunca haviam estudado programação apresentavam uma facilidade maior na resolução dos exercícios e representação em Haskell.

A conclusão que chegamos após essa primeira análise foi que os alunos de computação, ao ingressarem no curso superior, ainda não se apropriaram da habilidade de ver um problema segundo diferentes ângulos. Isto se expressa, principalmente, nas informações *i* e *ii*. Outra conclusão é quanto a romper preconceitos. A dificuldade inicial, identificada em *iii*, é um processo natural de readaptação do sujeito a novas condições de aprendizagem, o que não é, necessariamente, um problema. Já em *iv* conseguimos uma confirmação de que estávamos obtendo resultados significativos.

Promovemos um debate, utilizando o método cooperativo da controvérsia acadêmica, cujo tema era "Programação Funcional". Como o método utilizado requer uma preparação prévia dos participantes, seguida de uma inversão de papéis, foi possível aprofundarmos bastante o tema até conseguirmos identificar mecanismos de apoio à aprendizagem de programação que pudessem envolver, ao mesmo tempo, alunos com dificuldades e alunos com fluência.

Uma das questões recorrentes no debate foi o feedback do interpretador da linguagem que estavam utilizando. Segundo os alunos, o interpretador deveria ser capaz de apontar não só o erro de sintaxe, como também erros conceituais. Mas este tipo de feedback nenhum interpretador ou compilador é desenvolvido para fornecer.

Outra questão bastante debatida foi a da cooperação. Os alunos se ressentiam da falta de oportunidades de trocas com os colegas e os próprios professores em outro momento que não o das aulas. Eles relataram que quando apresentavam dificuldades na resolução de um exercício recorriam freqüentemente a um colega, mas depois não conseguiam caminhar sozinhos, pois, muitas vezes, essa ajuda era feita no laboratório e não havia um registro desse conhecimento para que pudesse ser reaproveitado posteriormente.

Ao final da disciplina realizamos um confronto entre os instrumentos de coleta de dados, similar ao realizado na metade da disciplina. Concluímos que houve realmente um aprendizado, apesar das dificuldades mencionadas. Este aprendizado, no entanto, poderia ter sido maior se tivéssemos utilizado algum mecanismo que pudesse facilitar o compartilhamento do conhecimento e promovesse a cooperação.

4. O Ambiente "Aprende"

A análise do experimento realizado, descrita na seção anterior, trouxe algumas reflexões acerca de maneiras de facilitar a aprendizagem de programação. Dentre essas maneiras, está clara a necessidade de um ambiente de apoio à programação que possibilite a cooperação entre os diferentes autores do processo de aprendizagem. Portanto, vislumbramos um desdobramento para o ambiente SAAP, chamado Aprende.

O Aprende tem este nome para enfatizar a atividade de aprender a programar, embora o feedback ainda seja uma das principais contribuições do ambiente. A diferença é que a cooperação é o que dá forma ao ambiente, proporcionando uma maior confiabilidade nos artefatos gerados.

Antes de começarmos a descrever o Aprende, imaginemos uma turma de estudantes iniciantes em programação ávida para interagir, aprender a solucionar problemas e representar as soluções em uma máquina funcional. Em toda turma existem aqueles estudantes que estão mais bem equipados para resolver problemas. Em geral, esses estudantes gostam de ajudar seus colegas a resolver os problemas e realmente ajudam. Por outro lado, há problemas que são mais facilmente resolvidos por um grupo de estudantes e outros tipos de problemas por outro grupo de estudantes. É comum vermos os estudantes trocarem conhecimentos, mas o problema está no registro desse conhecimento. Como não há meios que possibilitem o registro dessa troca em ambientes de apoio à aprendizagem de programação, os estudantes acabam não aproveitando muito o resultado da cooperação dos colegas.

Para possibilitarmos a atuação de estudantes como facilitadores da aprendizagem o Aprende possui dois atores: o Estudante e o Time de Revisores. O Estudante é qualquer pessoa que deseja aprender programação, utilizando programação funcional; e o Time de Revisores é composto por qualquer pessoa (estudante, professor, pesquisador, etc.) que possui alguma contribuição para fornecer ao ambiente, seja em forma de resposta a um questionamento ou seja na forma de uma nova solução para um problema, ou ainda, a inserção de uma nova classe de problemas.

A Figura 1 representa o funcionamento do ambiente Aprende. Além dos dois atores (agentes reais) do processo de aprendizagem, três agentes povoam o ambiente. O primeiro é o chamado HUGinho. Ele possui este nome por ser o responsável de interagir com o interpretador HUGS e informar ao estudante caso seu programa apresente inadequações referentes às boas práticas de programação, erros de sintaxe (que o impedem de testá-lo) ou erros de semântica (o

teste foi mal sucedido). Neste último caso, o HUGinho informa ao estudante qual foi o teste em que seu programa não passou.

O segundo agente é o Facilitador, que possui este nome porque é responsável por relacionar programas a classes e problemas armazenados em bases de conhecimento. Seu papel foi definido de acordo com definições fornecidas pela epistemologia genética de Piaget [INHELDER e CELLÉRIER, 1996]. O terceiro agente é o Mediador, também inspirado em teorias de aprendizagem. Como a avaliação de cada solução ou classe proposta é distribuída para o corpo de revisores, o Mediador atua como intermediário neste processo.

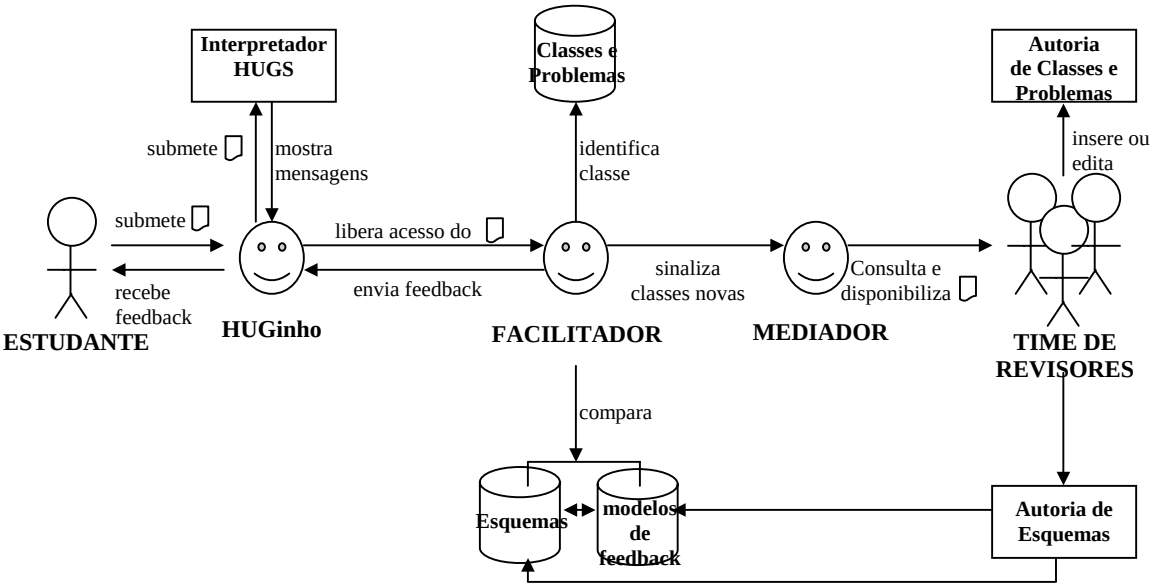

Figura 1 - O funcionamento do ambiente Aprende

Em termos de funcionamento, o Aprende possui uma única interface com o Estudante e outra com o Time de Revisores. O processo de aprendizagem se inicia com uma submissão de programa pelo Estudante. A partir daí, os agentes do ambiente assumem o controle das operações. Primeiramente, o HUGinho submete o programa do estudante ao interpretador Hugs e confere o resultado. Se houver erro de sintaxe o interpretador exibe uma mensagem de erro, a qual o HUGinho repassa ao estudante. Caso o interpretador aceite o programa, se houver, o HUGinho submete um conjunto de testes pré-definidos e se houver erro de semântica, ele fornece o feedback ao estudante.

Passada esta fase inicial, o Facilitador procura identificar a classe de problema mais aplicável ao programa, ao mesmo tempo em que procurar identificar se o problema que o originou já está na base de conhecimento. Para isso, ele compara o programa do estudante com os esquemas de programa representados. Se o programa for relacionado a um esquema, o próprio Facilitador recupera o feedback adequado àquela representação, enviando o feedback ao HUGinho, que o repassa para o Estudante. Porém, se o Facilitador não conseguir identificar o problema, ele o sinaliza para o Mediador que uma possível classe nova de problemas está sendo detectada. O Mediador, então, escolhe três membros do Time de Revisores e solicita que avaliem o programa e sugiram um novo esquema, se necessário, ou ajustem alguma classe existente. Esta etapa de revisão segue o princípio da revisão por pares em congressos científicos e pode ser mais bem visualizada na Figura 2.

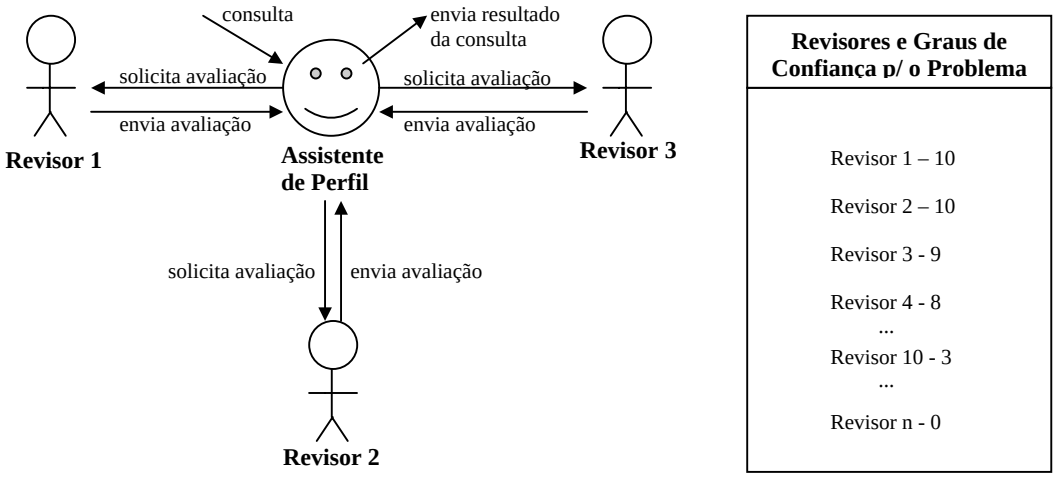

Figura 2 - Revisão de Propostas de Esquemas e Inclusões de Novos Revisores

Qualquer membro do Time de Revisores pode, sempre que desejar, utilizar diretamente a ferramenta de autoria de esquemas, incluindo novos esquemas relacionados a modelos de feedback. Neste caso, assim como no caso de solicitação do Mediador, a proposta passa pela avaliação do Time de Revisores. Nesse ponto, é importante salientar que o mesmo estudante que está submetendo seu programa pode, em outra situação, participar do ambiente como um revisor. Basta, para isso, que seja cadastrado como tal e que sua participação seja aprovada pelo Time de Revisores.

A participação no Time de Revisores pode ser solicitada pelo próprio candidato a revisor ou por um outro revisor. A recomendação é que esta escolha obedeça ao critério da taxonomia das habilidades cognitivas de Bloom [BLOOM et al, 1971], aceitando os candidatos que estivem nos últimos níveis de desenvolvimento cognitivo, avaliação e síntese.

Em uma requisição ao Time de Revisores, a solicitação de revisão é realizada de acordo com o perfil de cada participante. Adotamos, primeiramente os critérios fornecidos pelos próprios revisores, como interesses e afinidades com certos tipos de problemas. Posteriormente, baseado no currículo, o Assistente de Perfil, atribui um grau específico, considerando o tempo de experiência com programação funcional, a titulação e se participou ou participa de programa de monitoria – no caso de estudantes.

5. Considerações Finais

Procuramos relatar neste artigo todo o processo de levantamento de requisitos para a construção de um ambiente cooperativo de apoio à aprendizagem de programação porque entendemos que é de extrema importância que tanto professores quanto desenvolvedores de software educacionais atendam plenamente às necessidades dos estudantes. Para que essas necessidades sejam aferidas, é imprescindível que se elabore e aplique um experimento, ainda que seja prospectivo.

Uma vez identificadas as necessidades dos estudantes utilizamos elementos de um protótipo resultante de pesquisa anterior e incluímos suporte a novas funcionalidades. A reformulação do trabalho anterior (SAAP) deu origem a uma arquitetura onde se enfatiza o processo cooperativo para a construção do conhecimento coletivo.

A próxima etapa do projeto é a implementação destas novas funcionalidades e integração com as ferramentas desenvolvidas para o SAAP. Após essa etapa, pretendemos

utilizar o ambiente Aprende nas turmas de Engenharia e Ciência da Computação da UFAM que ingressarem em 2005.

Referências

Bloom, B.S. et al (1971). *Taxonomy of educational objectives: the classification of educational goals – handbook 1: cognitive domain*. Editora David McKay Co., New York. 1971.

Castro, T. H. C., Castro Jr, A. N., Menezes, C. S., Boeres, M. C. S., Rauber, M. C. P. V. (2002a) *Utilizando Programação Funcional em Disciplinas Introdutórias de Computação*. In: XXII Congresso da Sociedade Brasileira de Computação, volume 4 - X Workshop sobre Educação em Computação, Florianópolis, 2002.

Castro, T. H. C., Castro-Jr, A. N., Menezes, C. S., e Cury, D. (2002b). *Arquitetura SAAP - Sistema de Apoio à Aprendizagem de Programação*. Nos Anais de: XXII Congresso da Sociedade Brasileira de Computação, volume 5 - VIII Workshop de Informática na Escola, Florianópolis, 2002.

Goleman, D., Kaufman, P. e Ray, M. (2001). *O Espírito Criativo*. Editora Pensamento-Cultrix Ltda. 4a. Edição. São Paulo, 2001.

Inhelder, B. e Cellérier, G. *O Desenrolar das Descobertas da Criança – um estudo sobre as microgêneses cognitivas*. Editora Artmed. Porto Alegre, 1996.

Jones, M. P, Reid, A., the Yale Haskell Group, e the OGI School of Science & Engineering at OHSU. *The Hugs 98 User Manual*. 1994-2002. Disponível em http://cvs.haskell.org/Hugs/pages/hugsman/index.html. Visitada em Julho de 2004.

Menezes, C. S., Castro Jr, A. N., Castro, T. H. C., C. S., Boeres, M. C. S. e Rauber, M. C. P. V. *Introdução à Programação: uma Abordagem Funcional*. Apostila publicada em: http://www.dcc.fua.br/~pf. Visitada em Julho de 2004.

Thompson, S. *Haskell: The Craft of Functional Programming*. Segunda Edição. Editora Addison-Wesley. 1999.

Polya, G. *How to Solve It – a New Aspect of Mathematical Method*. Editora Princeton Science Library. Segunda Edição. New Jersey, 1988.