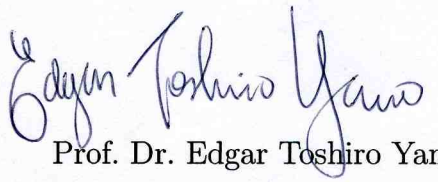
Tese apresentada à Divisão de Pós-graduação do Instituto Tecnológico de Aeronáutica como parte dos requisitos para obtenção do título de Mestre em Ciência no Curso de Engenharia Eletrônica e Computação, Área de Informática.

Daniel dos Santos Pêgas

COLETA E ANÁLISE DE MÉTRICAS NO PROCESSO DE APRENDIZAGEM DE LINGUAGENS DE PROGRAMAÇÃO

Tese aprovada em sua versão final pelos abaixo assinados:



Prof. Dr. Edgar Toshiro Yano
Orientador

Prof. Dr. Homero Santiago Maciel
Chefe da Divisão de Pós-graduação

Campo Montenegro
São José dos Campos, SP - Brasil
2005

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão Biblioteca Central do ITA/CTA

Pêgas, Daniel dos Santos
Coleta e Análise de métricas no processo de aprendizagem de linguagens de programação / Daniel dos Santos Pêgas.
São José dos Campos, 2005.
109f.

Tese de Mestrado – Curso de Engenharia Eletrônica e Computação – Área de Informática, 2005. Orientador: Prof. Dr. Edgar Toshiro Yano. .

1. Linguagens de programação. 2. Aprendizagem. 3. Modelagem (processos). 4. Métricas (software). 5. Métodos educacionais. 6. Ensino. 7. Educação. 8. Engenharia de software. I. Centro Técnico Aeroespacial. Instituto Tecnológico de Aeronáutica. Divisão de Ciência da Computação. II. Título.

REFERÊNCIA BIBLIOGRÁFICA

PÊGAS, Daniel dos Santos. **Coleta e Análise de métricas no processo de aprendizagem de linguagens de programação**. 2005. 109f. Tese de Mestrado – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Daniel dos Santos Pêgas
TITULO DO TRABALHO: Coleta e Análise de métricas no processo de aprendizagem de linguagens de programação.
TIPO DO TRABALHO/ANO: Tese / 2005

Daniel dos Santos Pêgas
Rua Alaska, 241 - Jd Flórida
CEP 12.321-700 – Jacareí-SP

COLETA E ANÁLISE DE MÉTRICAS NO PROCESSO DE APRENDIZAGEM DE LINGUAGENS DE PROGRAMAÇÃO

Daniel dos Santos Pêgas

Composição da Banca Examinadora:

Prof. Dr.	José Maria Parente de Oliveira	Presidente	-	ITA
Prof. Dr.	Edgar Toshiro Yano	Orientador	-	ITA
Prof. Dr.	Alexandre Ibrahim Direne	Membro Externo	-	UFPR
Prof. Dr.	Clóvis Torres Fernandes	Membro	-	ITA
Prof. Dr.	Celso Massaki Hirata	Membro	-	ITA

ITA

Agradecimentos

Tudo que consegui alcançar agradeço a Deus e a minha família.

Agradecimento especial ao Prof. Orientador, Dr. Edgar Toshiro Yano.

Agradeço, ainda, a todos que cruzaram minha vida e contribuíram para meu crescimento pessoal e profissional.

Resumo

Uma tendência no processo de aprendizagem em geral é o professor cada vez mais exigir do aluno a capacidade de exploração e criação de novos conhecimentos e não, simplesmente, memorização do conteúdo de uma disciplina. Entretanto, essa nova abordagem requer novos instrumentos para acompanhamento e avaliação do processo de aprendizagem.

Na abordagem tradicional de ensino de linguagem de programação, a avaliação se concentra no resultado, ou seja, provas e códigos prontos referentes a atividades propostas. Esta dissertação estende-se esta avaliação a todo o processo de aprendizagem, e para suportar isto, utiliza-se um modelo de acompanhamento e análise do processo de aprendizagem de linguagens de programação.

Um estudo de caso foi elaborado e quatro cursos de introdução à linguagem de programação foram ministrados a fim de se obter dados que pudessem dizer que o uso do modelo de acompanhamento e análise do processo de aprendizagem é uma boa alternativa tanto para o ensino como para a avaliação da aprendizagem de linguagens de programação.

Palavras–Chave: Linguagens de programação, Modelagem Cognitiva Aplicada à Educação, Observação eletrônica

Abstract

A trend in the learning process in general is the professor each time more to demand of the pupil the exploration capacity and creation of new knowledge and not, simply, memorization of the content of one disciplines. However, this new approach requires new instruments for accompaniment and evaluation of the learning process.

In the traditional approach of learning of programming language, the evaluation concentrates in the result, or either, tests and codes of activities proposals. In this research we extends this evaluation for all the process of learning, and to support this, we use a model of accompaniment and analysis of the process of learning of programming languages.

A study case was elaborated and four courses of introduction the programming language had been given to supply data that could say that the use a model of accopaniment is a good alternative for the learning of programming languages.

Keywords: Programming Languages, Cognitive Modeling Applied to Education, Eletronic Observation

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

SqlServer	Microsoft Sql Server 2000
Ms	Microsoft
Vs	Microsoft Visual Studio .Net 2003
VB	Visual Basic
IIS	Microsoft Internet Information Services
IExplorer	Microsot Internet Explorer
ABP	Aprendizagem Baseada em Problema
CD	Compact Disk
ASP	Active Server Pages

Lista de Símbolos

S	Definição da modelagem do aprendiz
Q	Grupo finito de atividades
P	Grupo finito de interações
V	Interações realizadas pelo aprendiz
F	Função de modelagem do aprendiz
Pe	Interações esperadas do especialista

Capítulo 1

Introdução

Neste capítulo, são apresentadas inicialmente um pouco das dificuldades encontradas no processo de aprendizagem de linguagens de programação, desde a preparação da aula, passando pelo acompanhamento do processo de aprendizagem chegando, por fim, à avaliação do aprendiz. Em seguida, são apresentados a motivação e o objetivo para esta dissertação e, por último, a estrutura desta.

1.1 O Problema

(PROULX, 2000) observou que um grave e freqüente problema vivenciado por aprendizes é a primeira experiência no aprendizado de programação, que se transforma em uma grande barreira para vários aprendizes.

Os motivos para essa experiência frustrante são vários: a preocupação excessiva com detalhes de sintaxe da linguagem sendo usada; a falta de uma visão daquilo que se quer solucionar, de idealizar soluções adequadas, de mapear essas soluções em passos seqüenciais e de abstrair o funcionamento dos mecanismos escolhidos; o estabelecimento de um raciocínio lógico visando a resolução de problemas, com base em um modelo incremental, em relação à complexidade e à estratégia de refinamentos sucessivos.

Estes problemas, pelo modelo tradicional de ensino de linguagens de programação, são dificilmente detectados ou quando são, normalmente após exames e provas, já não há tempo hábil para uma ação reparatória.

Atualmente, os professores, tendem a exigir, cada vez mais do aprendiz, a capacidade de exploração e criação de novos conhecimentos e não, simplesmente, memorização do conteúdo de uma disciplina. Com isso, podemos perceber que o professor tem uma tarefa extremamente complexa: exigir do aprendiz o que ele tem dificuldade, pois todos os fatores apresentados como dificuldades para o aprendiz devem ser considerados pelo professor quando estiver formulando o material que será apresentado em sala de aula, principalmente as atividades que serão propostas para exercitar os conceitos e técnicas.

Este novo processo exige do professor um acompanhamento mais próximo do aprendiz e não, simplesmente corrigir trabalhos e provas, mas isto não é tão simples, pois o desenvolvimento de um único projeto é geralmente demorado requerendo várias horas ou dias de trabalho, logo o professor não consegue fazer um acompanhamento mais próximo do trabalho realizado pelo aprendiz.

A avaliação deste trabalho é complexa, pois agora é necessário avaliar também a aprendizagem, e por causa disso torna-se bastante imprecisa. Muitas vezes, o professor não sabe com precisão avaliar se os objetivos do projeto foram atingidos, se as técnicas recomendadas foram utilizadas, quantas horas o aprendiz realmente gastou para implementar o projeto ou quais as dificuldades encontradas. A avaliação, agora, tem que ser de caracter formativo.

Outro ponto chave é a avaliação comparativa de desempenho dos diferentes aprendizes, que é muito subjetiva, pois cada projeto desenvolvido apresenta peculiaridades distintas, ainda que trate de um mesmo tema.

1.2 Motivação e Objetivo

As disciplinas introdutórias de programação têm um dos maiores índices de reprovação em todas as instituições de ensino brasileiras, o que torna ponto de reflexão por parte dos professores preocupados com a melhoria da qualidade no processo, ratificando a necessidade de alterações didáticas e metodológicas de apresentação (JúNIOR, 2002).

Visto que um aprendiz que tenha participado de um curso introdutório de linguagens de programação que propicie uma solução para a maioria das dificuldades mencionadas, estará, teoricamente, mais preparado para cursos avançados bem como para a vida pro-

fissional caso siga este ramo da informática, e que ainda não é encontrado na literatura acadêmica um modelo estabelecido de como acompanhar e avaliar o processo de aprendizagem de linguagens de programação, este trabalho tem como objetivo:

- Definir um modelo de acompanhamento e avaliação do processo de aprendizagem de linguagens de programação visando identificar as principais dificuldades dos aprendizes e, conseqüentemente, diminuir os índices de reprovação em disciplinas de programação.

1.3 Organização do trabalho

Esta dissertação está organizada em oito capítulos, precedidos desta introdução.

Uma introdução de como é o ensino e a avaliação de linguagens de programação nos dias de hoje é mostrada no capítulo 2.

No capítulo 3 é mostrado o modelo de acompanhamento do processo de aprendizagem de linguagens de programação, incluindo como é modelado o conhecimento do aprendiz a fim de se obter dados sobre seu processo de aprendizagem.

No capítulo 4 é mostrado onde as métricas se encaixam no processo de acompanhamento e avaliação da aprendizagem de linguagens de programação e expostas as métricas inicialmente propostas, o procedimento adotado para encontrá-las e os requisitos do sistema implementado.

A arquitetura do sistema de coleta e análise de métricas, as ferramentas utilizadas e suas limitações são mostradas no capítulo 5. É exposto também onde cada módulo do sistema implementado se encaixa no modelo pedagógico utilizado.

A melhor maneira de verificar se a modelo elaborado, se as métricas definidas e se o sistema irão auxiliar o professor no entendimento do processo de aprendizagem de programação, é a realização de um estudo de caso, onde sua estruturação é mostrada no capítulo 6, onde foi implementado: um curso de introdução a linguagem C#, os critérios adotados para a avaliação do professor e o modelo para a classificação do entendimento. É apresentado também neste capítulo trabalhos correlatos encontrados no mundo acadêmico.

No capítulo 7 são mostrados resultados práticos obtidos com experiências no processo

de aprendizagem da linguagem de programação C#. É mostrada a evolução de quatro turmas onde o curso foi ministrado e os dados que o sistema coletou e para viabilizar a comparação com a impressão do professor, visto que nesta pesquisa o professor acompanha todo o desenvolvimento de uma atividade. São expostos também os resultados julgados mais interessantes.

Finalmente, no capítulo 8, são apresentadas as conclusões, os trabalhos publicados e os trabalhos futuros.

Capítulo 2

O ensino de linguagens de programação

Segundo (BOULAY; O´SHEA, 1981) há três problemas fundamentais no ensino de programação:

1. Quais a dificuldades encontradas pelos aprendizes quando aprendem a programar?
2. Quais as características que uma linguagem de programação deve ter para que esta se torne a primeira a ser ensinada?
3. Qual a natureza da habilidade a ser ensinada e qual o melhor modo de ensinar?

Para responder a primeira pergunta, é necessário analisar as fases necessárias para realização de uma atividade de programação. A segunda pergunta é um pouco mais complexa. Em (BOULAY; O´SHEA, 1981) e (MARTIN, 1996) é possível observar diferenças de opiniões sobre qual linguagem deve ser a primeira a ser apresentada, mas é fato que o ambiente de programação deve prover ajuda ao aprendiz, sendo tão importante quanto a própria linguagem. A terceira pergunta leva a um estudo da evolução da educação.

O ideal quando se vai realizar uma atividade de programação é dividí-la em três fases:

1. **Planejamento**: É quando o aprendiz compreende o problema e elabora uma solução, através de algoritmos e fluxogramas. Esta solução já é voltada para a implementação, ou seja, o aprendiz nesta fase já esta passando o problema real para

uma forma computável. É de conhecimento de todos, que na quase totalidade, os aprendizes pulam esta fase, seja por falta de incentivo ou por acharem que não é necessária (MARTIN, 1996).

2. **Codificação**: Esta fase envolve o conhecimento da sintática e da semântica individual de cada comando (BOULAY; SOTHCOOT, 1987), da lógica de programação - habilidade de agrupar adequadamente os comandos a fim de resultar num algoritmo (BINDER; DIRENE, 1999). A sintaxe refere-se à escrita correta do comando, a semântica à correta aplicação deste, e a lógica ao mapeamento do problema em um programa. A habilidade semântica é normalmente considerada a mais importante e difícil de adquirir, enquanto que para a habilidade sintática é assumido que será adquirida através da familiaridade com a linguagem, das mensagens do verificador sintático e de livros texto. Na prática a maioria dos novatos possui dificuldade em corrigir erros sintáticos.

3. **Testes**: Nesta fase o aprendiz irá localizar e remover os erros da codificação. Quanto maior a experiência do programador, maior é a facilidade de remoção destes erros. Estes erros podem ser divididos em dois tipos: sintáticos e semânticos. O compilador facilmente detecta erros sintáticos, porém, (MARTIN, 1996) observou que mensagens mal formuladas pelo compilador levaram alguns alunos a modificar a lógica do programa, que estava correta. Os erros semânticos são os mais difíceis de serem corrigidos, porque levam em consideração a lógica e compreensão do problema.

Antes de responder qual a melhor modo de ensinar programação, é necessário observar a evolução na educação, que vem vivendo uma época de grandes transformações. É possível observar em (CORTELAZZO; OLIVEIRA, 1996) que os modelos usados já não atendem mais às necessidades atuais.

Atualmente professores tendem a exigir cada vez mais do aprendiz a capacidade de exploração e criação de novos conhecimentos e não, simplesmente, memorização do conteúdo de uma disciplina. (LITTO, 1996) identifica as principais características dos paradigmas da Educação:

- **O paradigma antigo**: Nesta concepção, o aprendiz ou receptor não possui conhecimentos prévios e cabe ao professor, tem um papel ativo, transmitir informações

a este. A aquisição de conhecimento é testada periodicamente através de provas e exames. A memorização da informação é o ponto fundamental deste paradigma.

- **O paradigma novo**: O aumento do volume de informação disponível, o aumento da complexidade nos setores da vida profissional, são alguns fatores que contribuíram para que o paradigma antigo se tornasse ultrapassado. A avaliação agora é feita constante e serenamente na carreira do aprendiz. A ênfase é colocada não na memorização de fatos, mas na capacidade do aprendiz pensar e se expressar claramente, solucionar problemas e tomar decisões adequadamente.

  O professor tem a liberdade para ser mais um facilitador, um parceiro na procura da informação e da verdade, aumentando a participação ativa do aprendiz no entendimento do conteúdo que está sendo transmitido. Sendo assim a motivação para a aprendizagem surge no aprendiz, ao invés de ser algo imposto pelo professor ou pelos pais. E o mais importante, há o reconhecimento de que a aprendizagem permanente daqui em diante será uma tarefa constante na vida profissional, cabendo à escola capacitar o aprendiz para aprender qualquer assunto que lhe interesse.

2.1 O ensino de programação

Atualmente, o ensino de programação segue uma mistura dos dois paradigmas: o professor transmite o conteúdo através de aulas tradicionais seguindo o paradigma antigo. O paradigma novo pode ser observado, pois a atividade de programar é extremamente criativa e a parte que deve ser memorizada é mínima se comparada com outras disciplinas.

As aulas restringem-se à sala de aula, o laboratório fica disponível ao aprendiz, mas sem que haja a presença de um professor. Cabe ao aprendiz, fora da sala de aula, vencer as várias dificuldades sozinho, como ambientar-se ao compilador.

O professor em geral tem pouco contato com a produção dos aprendizes, pois o contato restringe-se a participação do aprendiz na sala de aula. As dúvidas mais comuns neste começo muitas vezes ocorrem no laboratório, onde o aprendiz tem a oportunidade de passar os conhecimentos vistos na sala para a prática.

2.2 A avaliação

Segundo (PERRENOUD, 1999) a avaliação da aprendizagem, no novo paradigma, é um processo mediador na construção do currículo e se encontra intimamente relacionada à gestão da aprendizagem dos aprendizes. Na avaliação da aprendizagem, o professor não deve permitir que os resultados das provas periódicas, geralmente de caráter classificatório, sejam supervalorizados em detrimento de suas observações diárias, de caráter diagnóstico. A avaliação descreve que conhecimentos, atitudes ou aptidões que os aprendizes adquiriram, ou seja, que objetivos do ensino já atingiram num determinado ponto de percurso e que dificuldades estão a revelar.

Esta informação é necessária ao professor para procurar meios e estratégias que possam ajudar os aprendizes a resolver essas dificuldades e é necessária aos aprendizes para tentarem ultrapassá-las com a ajuda do professor e com o próprio esforço. Por isso, a avaliação tem uma intenção formativa.

A avaliação formativa que, conforme (HAYDT, 1995), permite constatar se os aprendizes estão, de fato, atingindo os objetivos pretendidos, verificando a compatibilidade entre tais objetivos e os resultados efetivamente alcançados durante o desenvolvimento das atividades propostas.

Esta avaliação representa o principal meio através do qual o aprendiz passa a conhecer seus erros e acertos, assim, tem um maior estímulo para um estudo sistemático dos conteúdos.

Outro aspecto é o da orientação fornecida por este tipo de avaliação, tanto ao estudo do aprendiz como ao trabalho do professor, principalmente através de mecanismos de feedback. Estes mecanismos permitem que o professor detecte e identifique deficiências na forma de ensinar, possibilitando reformulações no seu trabalho didático, visando aperfeiçoa-lo.

Capítulo 3

Modelo Pedagógico para o Ensino de Linguagens de Programação

Segundo o dicionário, a palavra método significa caminho ou processo racional para atingir um dado fim. Agir com um dado método supõe uma prévia análise dos objetivos que se pretendem atingir, as situações a enfrentar, assim como dos recursos e o tempo disponíveis, e por último das várias alternativas possíveis. Trata-se pois, de uma ação planejada, baseada num quadro de procedimentos sistematizados e previamente conhecidos.

Em pedagogia, entende-se por modelos os diferentes modos de proporcionar uma dada aprendizagem e que foram sendo individualizados pelos pedagogos ou pela investigação científica. O modelo não diz respeito aos vários saberes que são transmitidos, mas sim, ao modo como se realiza a sua transmissão. Podemos definir um modelo pedagógico como: uma forma específica de organização dos conhecimentos, levando em conta os objetivos do programa de formação, as características dos aprendizes e os recursos disponíveis.

Na escolha de um modelo pedagógico, o professor deverá considerar três fatores essenciais:

- As características dos aprendizes;
- As características do saber;
- O condicionamento e os recursos inerentes à situação de formação.

A escolha do modelo pedagógico pode influenciar os resultados finais. Não nos podemos esquecer que num grupo de aprendizes existe uma enorme diversidade de estilos e de ritmos de aprendizagem e, através da escolha e da aplicação correta dos métodos, se faz a gestão destas diferenças, portanto a escolha implica o sucesso ou o insucesso de alguns aprendizes.

Como apresentado, as disciplinas introdutórias de programação têm um dos maiores índices de reprovação em todas as instituições de ensino brasileiras, por isso a utilização de novos modelos pedagógicos se torna fundamental na tentativa de reverter este quadro.

Para o ensino de programação o modelo pedagógico tem que atender as fases de desenvolvimento de uma atividade (planejamento, codificação e teste), o novo paradigma da educação e oferecer suporte a avaliação formativa. Para atender estas necessidades esta pesquisa propõe um modelo pedagógico de acompanhamento e análise do processo de aprendizagem de linguagens de programação.

3.1 O modelo pedagógico de acompanhamento e análise do processo de aprendizagem de linguagens de programação

O novo paradigma propõe que a avaliação seja formativa, por isso um acompanhando mais próximo do aprendiz se torna fundamental. Observar e entender o que este faz durante a resolução de uma atividade é um ponto chave para esta avaliação. Para que isto se torne possível o modelo pedagógico de acompanhamento e análise do processo de aprendizagem de linguagens de programação coleta informações dos aprendizes enquanto este realiza uma atividade direto no ambiente de desenvolvimento.

A coleta de informações diretamente no ambiente de desenvolvimento enquanto o aprendiz realiza uma atividade é possível através de um sistema implementado e descrito no capítulo 4.

Para um melhor entendimento das etapas que o aprendiz realiza quando desenvolvendo uma atividade de programação, o modelo de acompanhamento e análise do processo de aprendizagem de linguagens de programação foi dividido em fases, onde em cada uma foram especificadas as estratégias pedagógicas a serem aplicadas. A seguir são apresentadas

estas fases.

3.1.1 Fase de Preparação do Aprendiz

Nessa fase é feita a apresentação em sala de aula, de modo tradicional, do conteúdo que será envolvido na atividade a ser desenvolvida.

3.1.2 Fase de Apresentação do Problema

O professor apresentará um problema (desconhecido até então) ao aprendiz, obedecendo à seqüência de referência especificada, identificando assim quais unidades pedagógicas estarão relacionadas ao problema em questão. O problema será apresentado quando o professor julgar que todos os aprendizes forem capazes de iniciar o problema.

Os problemas são apresentados de acordo com a unidade pedagógica em que o aprendiz se encontra. Alguns problemas podem envolver conceitos de mais de uma sub-unidade.

A diferença do problema proposto neste modelo de acompanhamento em relação ao modelo tradicional é que aqui os problemas são pouco estruturados, não pretendendo que haja uma única solução e quando, ao longo do processo de resolução do problema, se reúnem novas informações, a percepção do problema , conseqüentemente, suas soluções alteram-se.

Pretende-se que o problema sejam semelhantes a problemas do mundo real e conseqüentemente não sejam tão simples, ou seja, possam ser divididos em subproblemas.

Com isto espera-se que o esforço gasto pelo aprendiz para resolução destes problemas apóiem a aquisição da habilidade de programar e não sejam apenas testes ou exames.

3.1.3 Fase de Assimilação

Esta fase equivale à fase planejamento da realização de uma atividade de programação. É a fase de discussão do problema, geração de hipóteses, algoritmos, fluxogramas, identificação de requisitos, etc., relevantes ao contexto, a partir de estudo auto-dirigido (pesquisa na Web, em livros, troca de idéias com o professor, acesso ao Help, etc.). Em

suma, nessa fase o aprendiz faz o levantamento dos requisitos necessários à solução do problema.

O sistema implementado acompanha apenas o acesso ao help para futura avaliação paralela. O help em linguagens de programação é uma ferramenta de extrema importância, pois além de conter exemplos, contém toda a teoria, mesmo a parte que não foi apresentada pelo professor.

É de conhecimento público que o aprendiz na maioria das vezes pula esta fase quando se trata de desenvolver um programa, isto é um fator relevante que fez com que a pesquisa enfatizasse mais a fase de resolução do problema.

3.1.4 Fase de Resolução do Problema

Teoricamente com base na fundamentação obtida na fase anterior, o aprendiz tentará resolver o problema, cabe aqui salientar, que, em atividades de programação o aprendiz na maioria das vezes começa diretamente nesta fase, pulando a fase anterior.

Caso não consiga encontrar uma solução, o professor poderá eliminar dúvidas, orientar o aprendiz, apresentar conceitos transversais relevantes etc. Após a orientação do professor, o aprendiz poderá permanecer nessa fase ou retomar a fase de assimilação para reavaliação dos resultados encontrados até o momento, ou, se necessário, efetuar mais pesquisas. Em seguida, o aprendiz tentará resolver o problema.

Se o problema for resolvido, este irá para a fase de validação e avaliação dos resultados.

Em uma atividade de programação este ciclo entre fases é constante, mas o aprendiz na maioria das vezes não documenta isto, preferindo alterar diretamente o código gerado, por isto esta é uma fase de grande importância para o entendimento da aprendizagem de linguagens de programação.

O sistema implementado nesta pesquisa monitora todos os passos realizados pelo aprendiz nesta fase.

Esta fase equivale as fases de codificação e teste da realização de uma atividade de linguagem de programação.

3.1.5 Fase de Validação e Avaliação Final

Nesta fase serão usados valores obtidos na fase anterior para validar a solução do problema apresentada pelo aprendiz. À medida que os resultados são apresentados e validados, é feita uma avaliação quantitativa e qualitativa dos mesmos pelo professor.

Caso a avaliação não seja satisfatória o professor pode pedir adotar medidas para solucionar este problema.

Como esta avaliação leva em conta todo o processo realizado pelo aprendiz, provê um feedback tanto para o professor como para o aprendiz e permite, através das informações coletadas, identificar dificuldades (isto é mostrado no capítulo 6) é possível afirmar que esta avaliação é formativa.

Para que estas informações coletadas possam auxiliar o professor a entender e avaliar o processo de aprendizagem é necessário um modelo que informe o que aprendiz esta fazendo ou pensando em determinado momento e conseqüentemente o seu progresso. Este modelo é o modelo do aprendiz.

3.2 Modelo do Aprendiz

A modelagem do comportamento de uma pessoa apresenta um nível de dificuldade considerável. Isso ocorre devido ao fato das reações dos seres humanos serem completamente imprevisíveis e não seguirem um padrão específico. Por exemplo, alguns indivíduos tendem a reagir de maneira diferente, mesmo submetido às mesmas situações. Daí, um dos motivos pelos quais modelar o comportamento humano seja um verdadeiro desafio.

O caso específico de modelagem de um aprendiz segue a mesma dificuldade. O grau de dificuldade pode ser bem complexo dependendo do nível do aprendiz, do conteúdo a ser estudado e da metodologia aplicada ao processo de ensino/aprendizagem.

Em geral, se pensa em um modelo do aprendiz com uma descrição envolvendo o que o aprendiz sabe sobre um domínio específico (WENGER, 1987). É muito simples considerar que o conhecimento do aprendiz é estático como uma lista de fatos contida em sua mente. Pode-se concluir melhor, que o modelo deve conter se pensarmos em termos de o que o aprendiz está fazendo. Assim, teríamos o seu comportamento diante das atividades e o

que ele pode articular quando justifica uma hipótese (seu provável comportamento).

Um modelo cognitivo do aprendiz caracteriza o domínio do conhecimento em termos das propostas e aplicações dadas por ele para a assimilação de um fato, solução de um problema, etc. Assim, podemos dizer que o conhecimento do aprendiz é centrado em um subconjunto em que estão sendo considerados métodos e estratégias particulares. Desse modo, podemos concluir que deve ser modelada uma pessoa pensando, pois o modelo será usado para explicar o que o aprendiz está tentando fazer.

Em termos gerais, a modelagem do aprendiz é feita de tal forma que, nesse módulo, sejam armazenadas informações sobre o seu progresso. Tais informações podem posteriormente ser usadas para medidas de desempenho e avaliação. Por exemplo, se é necessário saber se o aprendiz está tendo progresso em relação ao domínio estudado.

3.3 A modelagem do aprendiz

Na modelagem de um aprendiz são muitas as variáveis que devem ser consideradas. Enquanto alguns aprendizes assimilam um determinado assunto muito facilmente, outros, de mesmo nível, têm dificuldades.

Existem na literatura modelos que definem o comportamento dinâmico do aprendiz, como, por exemplo, os modelos overlay e bug (WENGER, 1987).

No modelo de overlay, o conhecimento do aprendiz é considerado um subconjunto do conhecimento do professor. A partir do confronto desses dois níveis de conhecimento a aprendizagem consiste em adquirir progressivamente um subconjunto cada vez mais completo. Este modelo tem uma possibilidade limitada de representação e é incapaz de tratar de erros ou noções incorretas.

A estrutura do modelo bug library compara o comportamento do aprendiz com um banco de dados de ”erros previsíveis”, que foi manualmente construído a partir da coleta dos erros mais comumente cometidos. Entretanto, como a coleta de dados é manual, implica em um grande esforço e na necessidade de elaborá-lo para as diferentes áreas de domínio.

Numa proposta denominada Dinamic Bug Library, (CRAW; SLEEMAN, 1990), a base

de dados é ampliada a partir de informações coletadas de comportamentos do aprendiz que não estejam previamente modeladas pela máquina.

Seguindo esse enfoque, foram elaborados modelos mistos, onde os bancos de dados continham informações tanto de erros como de acertos e modelos revisionistas, (BAFFES, 1994) que atuam comparando a teoria correta com uma série de exemplos, mas limitados a um conjunto de domínios específicos.

Ambos os modelos possuem grandes limitações. No overlay temos o fato de que como o conhecimento do aprendiz é subconjunto do conhecimento do especialista, no caso do problema permitir múltiplas soluções, haverá o risco do aprendiz resolvê-lo de maneira que sua solução não pertença ao domínio do especialista; sendo assim, embora sua decisão possa estar correta o sistema poderá considerá-la errada. O modelo bug, por sua vez, limita-se pela dificuldade de gerenciamento de sua biblioteca de erros. De acordo com a previsão de aprendizes que irão usar o sistema, o número de erros a ser cadastrado pode crescer muito, impedindo assim uma boa performance do sistema.

Este trabalho implementa o modelo overlay (WENGER, 1987), acompanhando a fase de resolução do problema pelo aprendiz. O professor a qualquer momento tem acesso aos dados das atividades que foram ou que estão sendo desenvolvidas pelos aprendizes, assim, suas pendências poderão ser reconhecidas e tratadas devidamente pelo professor que é um aliado fundamental nesta tarefa. Os resultados alcançados podem ser usados como fator determinante na continuidade das estratégias pedagógicas utilizadas ou, se necessário, na redefinição de novas estratégias pelo professor.

O modelo do aprendiz é responsável por informar ao professor sobre o comportamento do aprendiz. As informações repassadas por este modelo representam a situação atual do aprendiz. Tais informações indicam o que o aprendiz está tentando fazer naquela etapa do estudo, levando em conta o objetivo a ser alcançado, que é a conclusão de uma atividade proposta.

Visto que uma atividade pode ter várias soluções, isto é fato em linguagens de programação, a avaliação do processo é feita pelo sistema em conjunto com o professor, sendo este último o responsável pela decisão final.

O professor poderá comparar os dados coletados, durante a fase de resolução do problema pelos aprendizes, com os resultados esperados pelo especialista no domínio (que

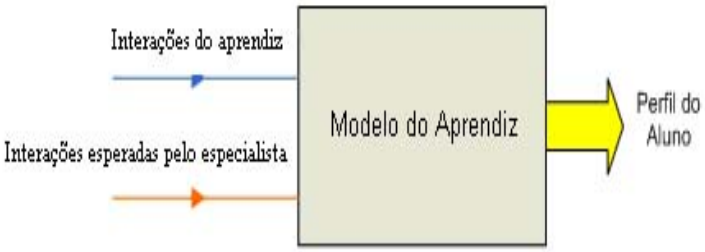


Figura 3.1: O modelo do aprendiz

pode ser o próprio professor ou mesmo algum outro aprendiz).

3.4 Funcionamento do Modelo do Aprendiz

O modelo implementado possui duas entradas e uma saída. As entradas correspondem às interações do aprendiz, e as interações realizadas pelo especialista. A saída indica o perfil do aprendiz analisado com base em regras estabelecidas. A figura 3.1 representa em forma de caixa preta o modelo do aprendiz indicando suas entradas e saídas.

Nesta pesquisa, interações são ações que o aprendiz realiza durante a fase de resolução do problema diretamente no ambiente de desenvolvimento, por exemplo, compilações, uso de debug, acesso ao help, etc.

Para a implementação da relação, entre o especialista nos domínios da atividade e o domínio do conhecimento do aprendiz, as partições do domínio são expressas em termos de classes de equivalência. Através da comparação entre as partições, o professor talvez acerte o que o aprendiz está tentando fazer, na pior das hipóteses o professor terá conhecimento das principais dúvidas enfrentadas pelo aprendiz.

A definição da modelagem do aprendiz, denominada S. Temos:

$$S = (Q, P, V, F)$$

Onde:

- Q = é não vazio, grupo finito de atividades;
- P = é não vazio, grupo finito de subproblemas, mas no modelo acompanhamento é

entendido como interações esperadas;

- V é dado por:

$$V = \bigcup_{p \in P} Vp$$

é não vazio, e seta valores para os subproblemas e Vp será chamado de domínio do subproblema p e

- F = é a função de modelagem do aprendiz

$$F = QxP \rightarrow Vf(q,p) \in Vp$$

Para todo

$$q \in Q \quad e \quad p \in P$$

A função acima é chamada de conhecimento do aprendiz de q em S.

Uma vantagem do modelo de acompanhamento é a facilidade no uso de diferentes abordagens pedagógicas.

Estendendo o modelo do aprendiz, adicionado uma nova coluna Pe, que representará as interações esperadas do especialista às atividades. Pe terá valores m1, m2, m3... onde mi representa o método de resolução do problema pelo especialista para responder Qi.

$$S = (Q, P, V, F), Fq(p)$$

Onde:

$$q \in Q \quad e \quad p \in Ps = \{P - Pe\}$$

Tem-se Ps como subconjunto de conhecimento do aprendiz que ainda falta para ser igual ao conjunto de conhecimento do especialista.

Exemplificando:

S=(Q,P,V,F),Fq(p)

onde Q é a atividade;

P = P1, P2 - Possíveis interações;

V = Verdadeiro, Falso - Interações realizadas pelo aprendiz e

Fq=R - Resposta do especialista

No modelo de acompanhamento a resposta do especialista corresponde a todas as possíveis interações realizadas por ele.

3.5 Como utilizar o modelo pedagógico de acompanhamento e análise do processo de aprendizagem de linguagens de programação

O professor irá apresentar os conceitos gerais, dos tópicos contidos na unidade, de maneira tradicional em sala de aula. Tópicos que já são de conhecimento dos aprendizes, como por exemplo, laços de repetição que são vistos na disciplina de algoritmos, podem, ou não serem apresentados ou apenas ter a sintaxe mostrada.

A partir do conteúdo apresentado, o professor propõe um problema. Neste momento há uma discussão entre o professor e os aprendizes sobre o problema, ou seja, uma explicação dos requisitos que devem ser atendidos. Quando estes requisitos forem entendidos pelos aprendizes, estes irão começar o desenvolvimento.

A maioria dos aprendizes não elabora fluxogramas e algoritmos e preferem ir direto para a codificação, ou seja, fase de resolução do problema, onde o sistema implementado atua, por isso a fase de assimilação não é enfatizada.

Durante a fase de resolução do problema, o professor se transforma em um facilitador, por exemplo, o aprendiz precisa de determinada função, raiz quadrada por exemplo, mas não sabe a sintaxe e nem como utilizar. Neste caso o facilitador pode dizer a sintaxe e cabe ao aprendiz descobrir como utilizá-la.

A fase de resolução do problema, pode ser realizada com a presença ou não do facilitador. Meios de comunicação como emails e chats podem ser utilizados.

Com os dados obtidos pelo sistema implementado o professor irá fazer a avaliação do processo de aprendizagem e após novas unidades podem ser apresentadas.

3.6 Trabalhos correlatos

Há uma classe de sistemas chamados diagnosticadores (Bug Finders) ou Diagnóstico Automático de Programas que têm como objetivo encontrar e classificar erros nos programas feitos pelos aprendizes, visando ter um feedback tanto para o aluno como para o professor afim de se obter um melhor aprendizado por parte do aprendiz.

Os sistemas de diagnóstico podem ser encaixados em três categorias (BOULAY; SOTHCOOT, 1987):

- **Solução de referência**: Sistemas que utilizam uma solução de referência para confrontar com a solução do aprendiz. Um exemplo de trabalho que segue esta filosofia é o LAURA (ADAM; LAURENT, 1980). O LAURA utiliza um processo para correlacionar padrões entre a solução de referência e a solução do aprendiz mesmo que estes não sejam idênticos. A grande deficiência do LAURA é ser capaz apenas de diagnosticar erros pequenos que não levam em conta o contexto principal de solução do problema abordado.

  Ao comparar o LAURA com o modelo de acompanhamento desta pesquisa, pode-se perceber duas diferenças: enquanto o LAURA informa o aprendiz sobre seus acertos, o modelo de acompanhamento informa ao professor o que o aprendiz esta tentando fazer além de deixar para o aprendiz descobrir quais são seus erros e acertos. Uma solução bem diferente da solução de referência é acusada como incorreta no LAURA enquanto no modelo de acompanhamento o professor tem a capacidade de verificar que a solução está correta.

  Ainda nesta linha de pesquisa há o trabalho de (DIRENE; GUEDES; SANTOS, 2003) onde através de uma linguagem de especificação é possível a detecção de erros ou acertos no programa do aprendiz, possivelmente melhorando a qualidade do feedback

correspondente à descrição de problemas (ou virtudes) na solução do aprendiz.

Esta abordagem utiliza um linguagem de especificação própria enquanto o modelo de acompanhamento pode ser utilizado em qualquer linguagem de programação disponível.

- **Análise de Especificação**: Sistemas que avaliam o programa do aluno segundo a especificação do problema. Alguns exemplos de sistemas que utilizam análise de especificação são MYCROFT (GOLDSTEIN, 1975), AURAC (HASEMER, 1983), PUDSY (LUKEY, 1980) e PROUST (JOHNSON, 1986). A grande vantagem dessa abordagem é a capacidade de relacionar fragmentos de código aos objetivos presentes na especificação.

  Como em linguagens de programação há soluções distintas para o mesmo problema, na análise de especificação isto se torna muito complexo, as vezes não é possível identificar uma solução correta, enquanto no modelo de acompanhamento não se restringe a isto, pois se leva em conta o que o aprendiz faz para chegar a solução.

- **Diálogo de Depuração**: Sistemas dessa categoria entram em um diálogo com o aprendiz, tentando orientá-lo a encontrar o erro. Um exemplo desse tipo de sistema é o desenvolvido por (SHAPIRO, 1983).

  No modelo de acompanhamento não há dialogo com o aprendiz, este tem que descobrir o erro sozinho ou no máximo pedindo auxilio ao facilitador na fase de resolução de problemas.

Ao pesquisar sobre coletar dados durante o desenvolvimento da atividade pelo aprendiz tem-se alguns trabalhos correlatos a esta pesquisa listados a seguir:

O departamento de computação da Open University, possui em andamento um projeto de pesquisa denominado AESOP, (PAINE; THOMAS, 1999), que consiste em coletar dados de um ambiente de aprendizagem de orientação a objeto para:

- Informar ao educador qual foi o melhor efeito da aula;
- Prover um aparato para identificar problemas que estudantes podem experimentar enquanto aprendem e

- Prover evidências que melhorem o projeto do ambiente de programação.

Inicialmente o projeto contou com a coleta de informação de 30 alunos, em 2000 este número estava em torno de 300 estudantes.

O sistema funciona através de um ambiente de aprendizado que utiliza SmallTalk e que funciona da seguinte maneira: o aprendiz recebe um CD contendo trabalhos de aprendizagem. Cada trabalho possui livros de aprendizagem que contém atividades práticas relacionadas.

O aprendiz desenvolve a atividade e entrega um CD com as atividades desenvolvidas. Através deste CD o professor pode verificar quais atividades o aprendiz fez e quais ele deixou de fazer.

Em conjunto com este trabalho, também na Open University, há um outro enfoque que trabalha em conjunto com o projeto AESOP que trata da gravação e análise do trabalho feito para entender as atividades práticas (THOMAS; PAINE, 2002). Como é um projeto em paralelo contou com os mesmos alunos do projeto AESOP divididos em 8 grupos de atividades. Puderam ser constatadas diferenças significativas entre a percepção do autor do curso e como os alunos se comportam realmente, principalmente no que diz respeito ao tempo de resolução das atividades.

A grande diferença entre este modelo proposto pela Open University e o modelo de acompanhamento é que no primeiro o professor só tem a idéia das dúvidas ou dificuldades do aprendiz ao final da atividade quando o CD é entregue, enquanto no modelo de acompanhamento o professor tem estas informações à medida que o aprendiz resolve a atividade.

A Monash University está realizando um estudo para entender melhor o esforço do estudante na tarefa de desenvolver programas. Para isso estão desenvolvendo um plug-in para o Visual Studio .NET 2003 Academic para capturar conteúdo dos projetos, progresso de compilação e estatísticas de execução. Este projeto ainda está em fase de definição dos requisitos necessários, portanto a comparação com o modelo de acompanhamento é ideal apenas quando aquele estiver pronto.

A University of Hull também está realizando um estudo para entender os hábitos dos estudantes de programação. Para isto estão desenvolvendo um plug-in para o Visual Stu-

dio.Net 2003 Academic para capturar quantas vezes o aluno compila um projeto para entender como o aluno trabalha nele e por quanto tempo, (GREY, 2004). O projeto se encontra parado esperando o lançamento da nova versão do vs que irá agregar funcionalidades que o sistema irá utilizar.

É possível perceber que o modelo de acompanhamento busca informações além de estatísticas de tempo e compilação visando identificar dificuldades encontradas pelos aprendizes.

Capítulo 4

Métricas de aprendizagem de linguagens de programação

Para avaliar o processo que o aprendiz utilizou para a resolução do problema é preciso saber como ele se comportou, ou seja, o modelo do aprendiz que é responsável por informar ao professor sobre o comportamento do aprendiz.

O modelo do aprendiz funciona pelo princípio da comparação, ou seja, compara as interações realizadas pelo aprendiz durante a resolução do problema com as interações que um especialista realizou quando resolveu o mesmo problema. Estas interações passarão a ser chamadas de métricas.

Na fase de validação e avaliação final do modelo pedagógico de acompanhamento de aprendizagem, estas métricas serão os parâmetros de entrada que o professor utilizará para, auxiliado pelo sistema implementado, atribuir um conceito para o processo de aprendizagem.

4.1 Processo usado na elaboração das métricas utilizadas

As métricas iniciais de aprendizagem de programação foram elaboradas juntamente com o processo para determinar os requisitos de coleta do sistema. Estes foram elaborados da seguinte maneira:

1. Identificação preliminar de métricas;

2. Pesquisa com professores de cursos de programação;

3. Análise dos resultados da pesquisa com os professores.

A identificação preliminar de métricas foi executada através da elaboração de uma lista de métricas obtida através de experiências dos autores com o ensino de linguagens de programação e ambientes de desenvolvimento de software. A partir dessa lista de métricas foi elaborado um questionário distribuído para professores de diferentes cursos de programação. Foram envolvidos professores de três instituições de ensino: uma das instituições é de ensino médio profissionalizante, outra é uma instituição com curso de graduação e a terceira apresenta cursos de graduação e de pós-graduação, totalizando vinte professores participantes da pesquisa.

Na tabela 4.1, ao final deste capítulo, temos as questões apresentadas aos entrevistados, todas foram categorizadas como muito útil, útil, moderado, pouco útil ou inútil.

Além destas questões havia uma área para comentários, críticas e sugestões. Ao final deste capítulo, na tabela 4.2, temos as respostas já totalizadas e em porcentagens.

Na tabela 4.2 foram selecionadas as respostas cuja soma das porcentagens de respostas úteis e muito úteis foram superior ou igual a cinqüenta por cento, assim tem-se a idéia do que é realmente útil coletar através da experiência dos entrevistados.

Com isso foram identificados os seguintes requisitos para acompanhar a aprendizagem:

- Relacionar todos os erros acusados, cada vez que o aluno compila o projeto;
- Mostrar a relação entre o tempo que o aluno passa depurando o projeto (debug) com o tempo total gasto no desenvolvimento;
- Mostrar o momento em que o aluno começa a usar o debug;
- Mostrar se o aluno passou por todo o código com o debug;
- Mostrar o que o aluno monitora durante o debug (variáveis, expressões e threads);
- Mostrar estatísticas de erros mais comuns de um aluno e/ou turma;

- Mostrar estatísticas de tempos de um aluno e/ou turma;
- Mostrar o que o aluno pesquisou no help;
- Mostrar os trechos de código que foram aproveitados de atividades anteriores "Mostrar se o aluno utiliza "wizards";
- Mostrar se os padrões recomendados foram utilizados;
- Mostrar a relação entre linhas de códigos e comentários;
- Mostrar a evolução do arquivo fonte.

Muitos entrevistados ressaltaram a importância de saber a qualidade dos comentários, se as estruturas de decisão e de repetição estão sendo usadas corretamente, o uso correto de passagem de parâmetros e especialmente a importância de comparar os resultados das análises obtidas com analises de fluxogramas, algoritmos e diagramas que os alunos venham a desenvolver em conjunto. Esta comparação levará, segundo os entrevistados, a uma visão mais detalhada de como o aluno aprende linguagens de programação, mas infelizmente, como já mencionado, nosso sistema analisa apenas o código-fonte.

4.2 Métricas utilizadas no sistema implementado

A partir das informações obtidas na seção anterior, foi colocado na tabela 4.3, ao final deste capítulo, as métricas de aprendizagem que o sistema implementado trabalha.

A partir destas informações foram definidos os requisitos para o sistema implementado:

- Armazenar horário e data de inicio e fim do uso do debug;
- Armazenar linha inicial e final por onde passa o debug;
- Armazenar erros acusados pelo compilador;
- Armazenar horário e data de inicio e fim de uma compilação;
- Armazenar horário e data, bem como os próprios eventos gerados pelo compilador;
- Armazenar variáveis, expressões e threads que são monitoradas pelo debug;

- Armazenar linha de inicio e fim de cada comentário e classificá-los quanto à sua relevância;
- Armazenar número de classes, variáveis, funções e referências do projeto de acordo como tempo;
- Armazenar data e hora que o aluno abre o projeto;
- Armazenar número IP da máquina que o aluno esta utilizando;
- Armazenar data, hora e expressão utilizada pelo help;
- Armazenar linha de inicio e fim de um padrão recomendado identificado;
- Cadastrar e identificar o aluno através de login e senha;
- Armazenar teclas relevantes acionadas pelo aluno;
- Armazenar data e hora de inicio e fim da atividade;
- Disponibilizar atividade para o aluno;
- Analisar informações e
- Mostrar resultado das análises.

4.3 Métrica de evolução

Analisando a evolução do código fonte, principalmente o número de linhas de código por compilação, temos uma métrica de evolução que merece uma atenção especial.

Quando analisado o gráfico de evolução do arquivo fonte durante o desenvolvimento do projeto pelo aprendiz, pode-se dividí-lo em duas partes conforme a figura 4.1.

Onde:

- **P (Platô)**: indica que o desenvolvimento (tamanho do código fonte) está parado, as possíveis explicações para isto são:
    1. O aluno está depurando o código, ou seja, tentando resolver algum erro;

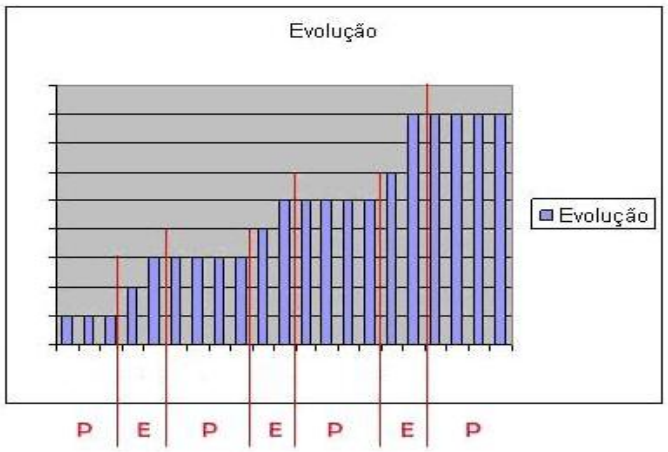


Figura 4.1: Gráfico de evolução do código fonte

2. O aluno está encontrando alguma dificuldade em determinado trecho de código, ou seja, no tópico da matéria correspondente;
3. O aluno não sabe como prosseguir;

- **E (Evolução)**: indica que o aluno está codificando;

O número de platôs e evoluções pode ser considerado como o número de subproblemas que o aluno encontrou durante o desenvolvimento da atividade, já o tamanho do platô pode indicar qual a dificuldade encontrada pelo aprendiz em determinado subproblema, ou o quanto o aluno testou determinado trecho do código. O tamanho da evolução pode indicar o quão grande é o subproblema encontrado pelo aluno.

Algumas variações do gráfico podem ser encontradas conforme figuras 4.2 e 4.3.

A figura 4.2 pode indicar um especialista, já que o tamanho do platô é praticamente nulo, ou seja, o aprendiz não encontrou dificuldades. Mas também indica que o código foi pouco testado, pois em cada compilação seu tamanho aumentou.

A figura 4.3 pode indicar um desenvolvimento de forma ”ideal”. O problema foi dividido em subproblemas e aparentemente testado antes de continuar a codificação. O ultimo platô pode indicar um teste geral do código realizado, envolvendo todos os subproblemas. Os tamanhos ideais de cada platô e evolução irão variar de acordo com cada subproblema (tamanho, grau de dificuldade e complexidade).

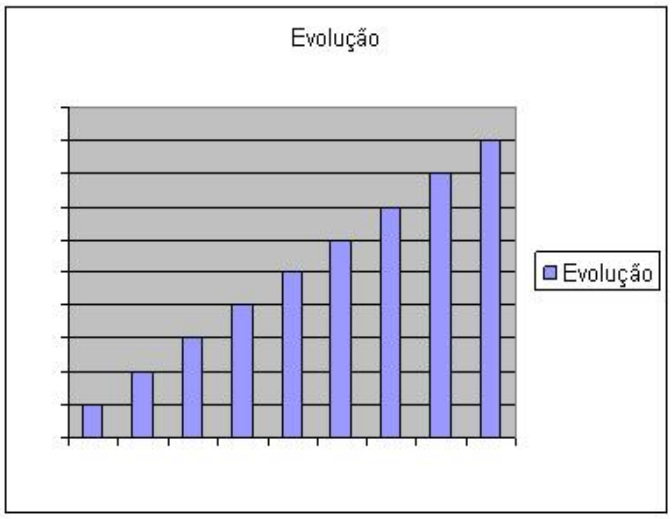


Figura 4.2: Primeira variação do gráfico de evolução do código fonte

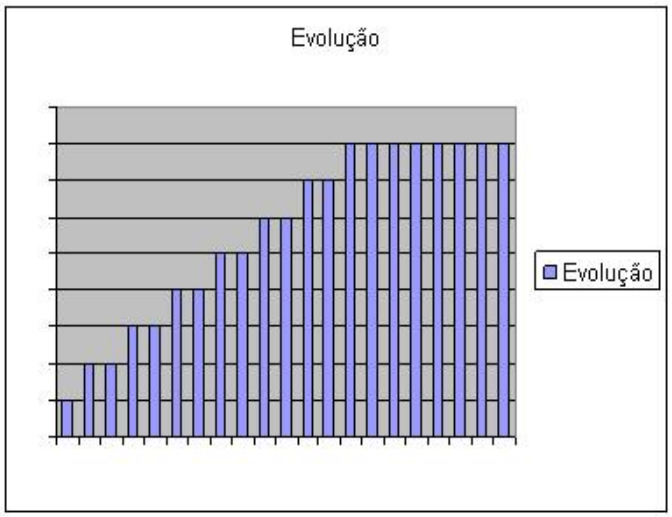


Figura 4.3: Segunda variação do gráfico de evolução do código fonte

4.4 Utilização das métricas no sistema implementado de coleta

Basicamente as métricas possuem dois papéis fundamentais no sistema implementado:

1. Dizer ao sistema o que ele deve monitorar e armazenar durante a fase de resolução do problema pelo aprendiz e
2. Servir de base para as comparações a fim de possibilitar a atribuição de um conceito ao processo de aprendizagem.

Com isto é possível concluir que as métricas são o ponto chave do sistema implementado, pois elas estão presentes em todos os módulos do sistema.

Tabela 4.1: Questionário distribuído aos professores

1. Saber quantas vezes o aluno compilou o projeto até chegar à versão final é
2. Saber se o aluno mexe nas configurações do compilador é
3. Saber o histórico de erros, ou seja, ter a lista de todos os erros ocorridos a cada compilação que o aluno executa é
4. Saber quanto tempo a aluno passa depurando (debug) o projeto em relação ao tempo total de desenvolvimento é
5. Saber o momento que o aluno começa a usar o debug, por exemplo, para saber qual erro após uma compilação o aluno esta tentando solucionar é
6. Saber se com o debug o aluno passou por todo o código é
7. Saber quais os breakpoints que o aluno utiliza é
8. Saber quantas vezes o aluno passa com o debug por determinado trecho do código é
9. Saber quantas variáveis e/ou expressões, estado da pilha, threads que o aluno monitora durante o debug é
10. Possuir estatísticas de erros mais comuns de alunos e/ou turmas é
11. Possuir estatísticas de tempos de alunos e/ou turmas é
12. Saber quantas vezes o aluno usa o help durante o desenvolvimento é
13. Saber o que o aluno pesquisou no help é
14. Saber o horário e o local (laboratório, em casa, etc...) que o aluno desenvolveu o projeto é
15. Saber qual parte do programa é reaproveitado de programas anteriores é
16. Saber se durante o desenvolvimento o aluno ”cola”trechos de código é
17. Saber se o aluno utiliza ”wizards”é
18. Saber se os padrões recomendados foram utilizados é
19. Saber a relação entre linhas de código e comentários é
20. Saber a evolução do arquivo fonte (tamanho, número de classes, etc...) é
21. Poder comparar os programas entregue pelos alunos para saber se há partes comuns”é
22. Saber se o aluno customiza o ambiente de desenvolvimento é
23. Possuir relatórios sobre todo o trabalho realizado pelo aluno no ambiente de desenvolvimento de software para futuras comparações é
24. Possuir relatórios de uma turma sobre o desenvolvimento de um projeto para futuras comparações é

Tabela 4.2: Respostas obtidas, totalizadas e em porcentagem

Pergunta	Muito Útil	Útil	Moderado	Pouco Útil	Inútil
1.	0	30	40	30	0
2.	0	40	30	20	10
3.	40	20	20	20	0
4.	40	50	10	0	0
5.	30	20	30	20	0
6.	20	30	30	20	0
7.	10	30	0	50	10
8.	0	40	30	20	10
9.	30	20	30	20	0
10.	70	20	10	0	0
11.	30	50	20	0	0
12.	10	20	60	10	0
13.	20	40	30	10	0
14.	20	0	40	30	10
15.	20	30	40	10	0
16.	20	20	10	40	10
17.	10	40	20	30	0
18.	33	57	11	0	0
19.	30	30	30	10	0
20.	10	50	30	0	10
21.	40	40	0	20	0
22.	0	40	40	20	0
23.	60	10	20	0	10
24.	60	10	20	10	0

Tabela 4.3: Métricas de aprendizagem de programação

Métricas de tempo
Tempo total do desenvolvimento da atividade
Tempo total de depuração da atividade
Métricas de compilação
Número de compilações
Erros por compilação
Horário de cada compilação
Métricas de debug
Horário de inicio e fim de cada utilização do debug
Trechos de código por onde o estudante passou com o debug
Determinação dos breakpoint utilizados
Métricas de evolução
Número de variáveis por compilação
Número de classes por compilação
Número de funções por compilação
Número de linhas de código por compilação
Métricas de pesquisa
Palavras pesquisadas no help
Métricas de reusabilidade
Utilização de copy e paste
Métricas de local
Local da realização da atividade

Capítulo 5

Arquitetura do Sistema Implementado

Para vialibilizar a coleta de informações na fase de resolução de problemas, foi necessária a implementação de um sistema, Data Collection Process, e este está dividido em três módulos conforme figura 5.1.

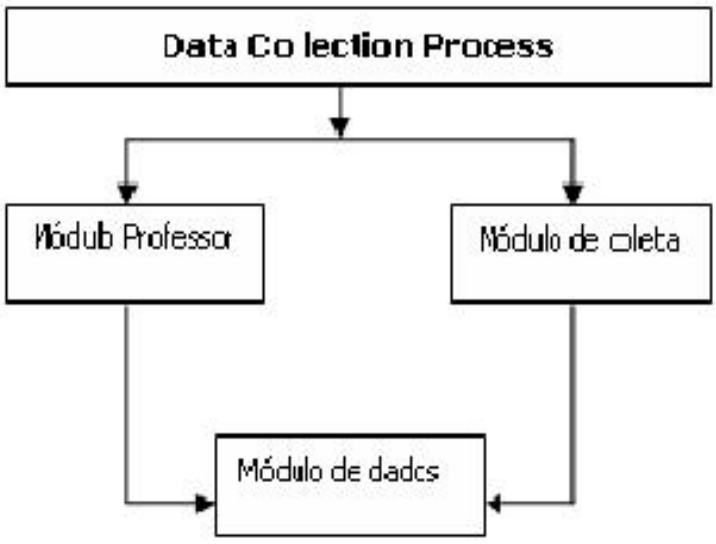


Figura 5.1: Arquitetura do sistema implementado

A comunicação entre os módulos de coleta e o módulo de dados é feita via Web Services, desta forma o aprendiz tem a flexibilidade de poder desenvolver a atividade em casa, de qualquer computador conectado à internet, ou de um laboratório da própria instituição de ensino, via rede local.

O professor tem a flexibilidade de acessar os dados do sistema de qualquer computador conectado a internet, pois a apresentação é feita via web application, ou seja, utilizando um web browser, como o IExplorer.

A implementação dos módulos foi feita através do uso das seguintes ferramentas: Vs, Sqlserver e servidor web IIS. Entretanto, a arquitetura foi concebida de modo a permitir a criação de uma implementação com ferramentas de código aberto.

5.1 Módulo de Coleta

O módulo de coleta consiste em um sistema instalado no ambiente de trabalho do aprendiz, que, de forma transparente, coleta e transmite informações, previamente definidas, sobre as atividades do aprendiz ao módulo de dados. Estas informações são transformadas em métricas, como, por exemplo, número de compilações.

A única funcionalidade que é apresentada ao aprendiz é uma tela de identificação (login), conforme figura 5.2 para que o sistema identifique o aprendiz cujas informações estão sendo coletadas. O módulo de coleta é instalado como uma extensão do tipo add-in em um ambiente de desenvolvimento como o vs.

O módulo de coleta tem duas funções principais:

1. Transformar eventos do ambiente de desenvolvimento em métricas que possam ser armazenadas em um servidor de banco de dados;

2. Analisar o código gerado pelo aprendiz, visando buscar, entre outras métricas, quantidade de variáveis, classes e funções utilizadas a cada compilação realizada.

Este módulo monitora a fase de resolução do problema do modelo pedagógico, tanto para auxiliar o professor como fornecer parâmetros para fase de validação e avaliação final.

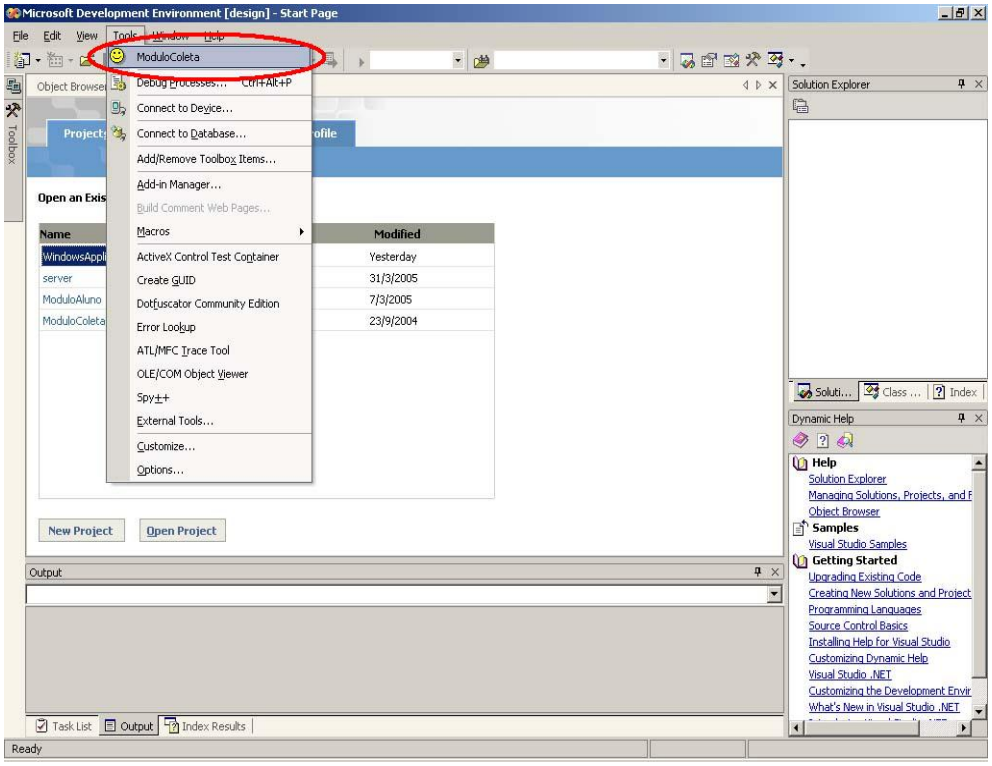


Figura 5.2: Funcionalidade acrescentada no ambiente de desenvolvimento

5.2 Módulo Professor

O módulo Professor está estruturado conforme a figura 5.3.

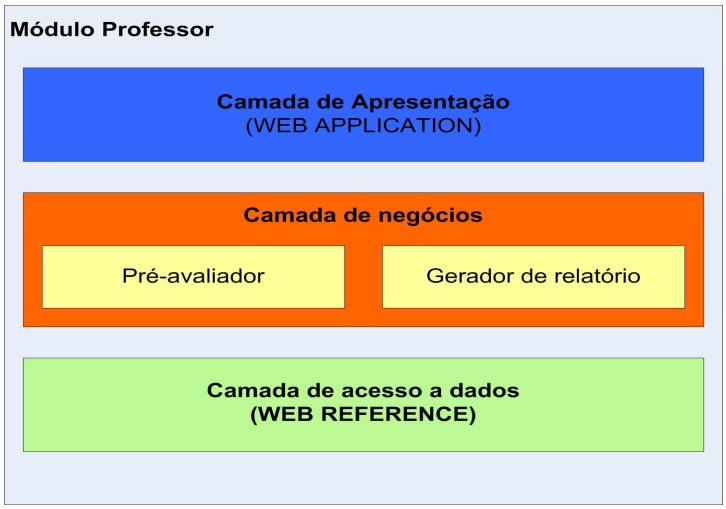


Figura 5.3: Arquitetura do módulo Professor

A camada de acesso a dados, também chamada de web reference, consiste em uma referência ao local onde os dados estão armazenados. A camada de negócio solicita a informação à camada de acesso a dados e esta busca no servidor de banco de dados

referenciado.

A camada de negócio é divida em duas funcionalidades:

- Pré-avaliador: consiste em um subsistema que informa ao professor o nível de aproveitamento do aprendiz. O subsistema compara algumas métricas obtidas na resolução do problema pelo especialista com as métricas coletadas do aprendiz. A figura 5.4 mostra um exemplo de como as informações são apresentadas ao professor.

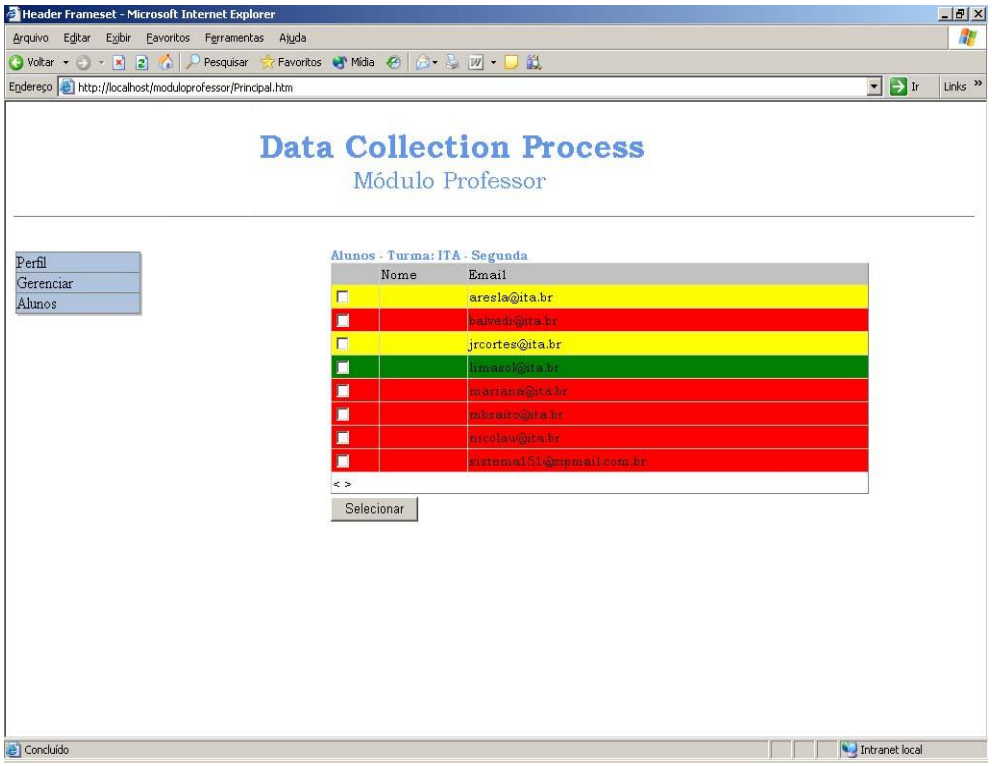


Figura 5.4: Tela de pré-avaliação gerada para o professor

Desta maneira o professor pode ir diretamente aos aprendizes que estão com maior dificuldades no processo de aprendizagem, pois estes estão categorizados através de cores: vermelho para baixo aproveitamento, amarelo para aproveitamento razoável e verde para um bom aproveitamento.

Esta funcionalidade trabalha em duas etapas do modelo pedagógico. Quando o aprendiz ainda esta resolvendo o problema apresenta métricas que podem ser utilizadas pelo facilitador. Quando o problema foi entregue, esta funcionalidade serve como uma pré-avaliação, ou avaliação do sistema do processo de aprendizagem, ou seja, fase de validação e avaliação final do modelo pedagógico.

- Gerador de relatório: consiste em um subsistema que a partir das métricas recebidas da camada de acesso a dados gera relatórios, gráficos e tabelas para que o professor possa entender o processo de aprendizagem do aprendiz. A figura 5.5 mostra um exemplo de gráfico que é apresentado ao professor.

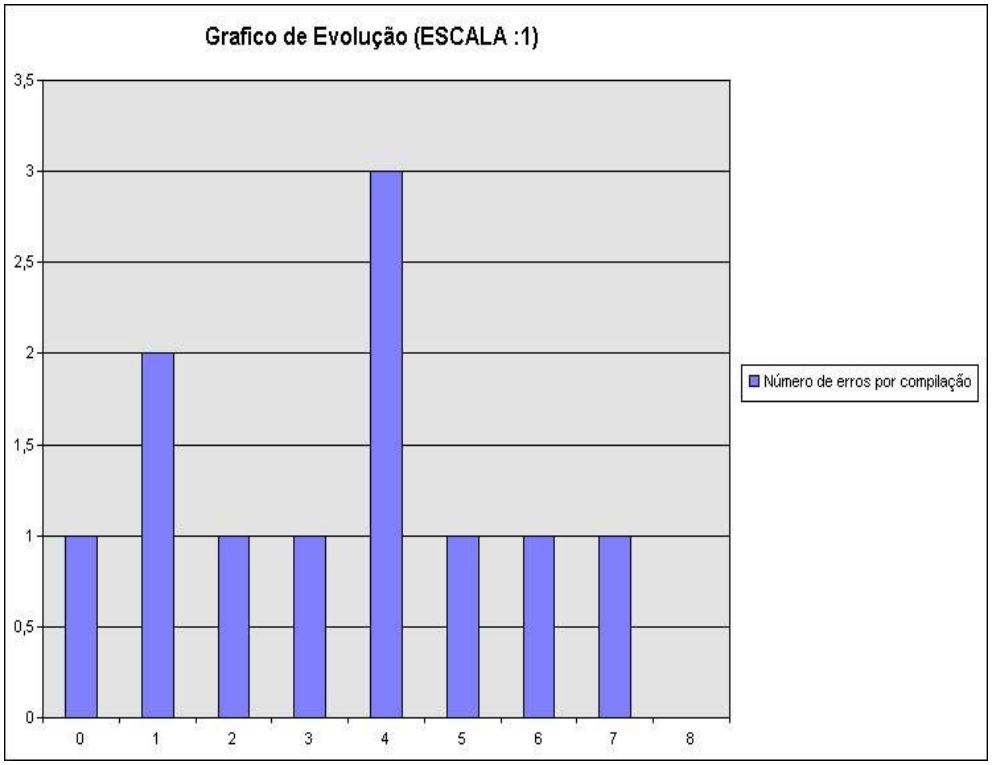


Figura 5.5: Gráfico gerado para o professor

Com este recurso o professor pode visualizar qualquer informação sobre o processo de aprendizagem e assim redefinir estratégias para melhorar o curso de programação.

Esta funcionalidade também esta presente em duas fases do modelo pedagógico. Ela auxilia o facilitador na fase de resolução do problema e auxilia o professor na fase de validação e avaliação final.

5.3 Módulo de Dados

O núcleo do sistema implementado é o módulo de dados, onde ocorre o armazenamento de todas as métricas coletadas durante o desenvolvimento da atividade pelo aprendiz. O primeiro passo para o desenvolvimento deste módulo foi o trabalho de modelagem dimensional, como visto na figura 5.6.

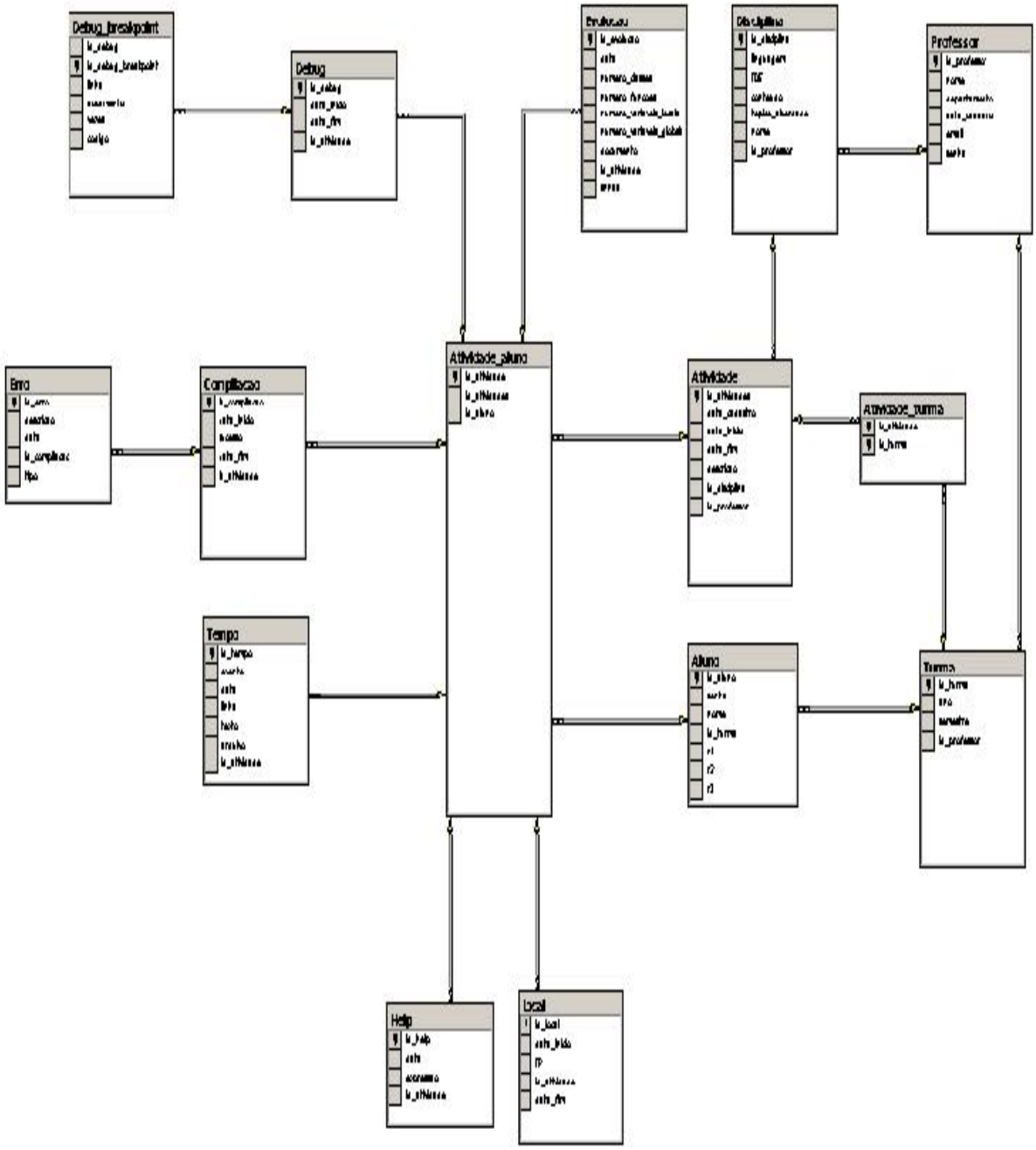


Figura 5.6: Modelo dimensional implementado

Com este modelo foi possível suprir todas as necessidades de armazenamento de métricas para o nosso sistema.

O tipo de modelo dimensional escolhido foi o estrela (ELMASRI, 2003) para atender os requisitos para a construção de um DataWarehouse (KIMBALL, 2002). O modelo propõe como tabela de fatos a atividade do aprendiz, já que é a partir dela que as métricas serão obtidas.

A idéia do uso de um DataWarehouse é combinar o resultado da fase de mineração de dados (KANTARDZIC, 2003) com a facilidade de geração de relatórios de ferramentas de Bussiness Intelligence (BI) para:

- Ajudar o professor a entender as métricas coletadas pelo sistema, já que uma das funcionalidades de um DataWarehouse é a facilidade de navegação e agregação de informações.
- Descobrir novas métricas de aprendizagem de linguagens de programação, pois na fase de mineração de dados há a possibilidade de se encontrar padrões que não foram previamente analisados.

Nesta pesquisa houve apenas um primeiro contato com o DataWarehouse onde se pode constatar que não havia dados suficientes para a aplicação desta tecnologia. O ideal será utilizar estes recursos após alguns cursos regulares de linguagens de programação, formando assim um base de dados consistente tanto em relação ao tempo (cursos com mesma duração) quanto em relação à quantidade de alunos (quanto mais informações maior a probabilidade de encontrar novas métricas). Por isto esta pesquisa se detém na análise de métricas previamente conhecidas.

5.4 Limitações do sistema implementado

A maior limitação do sistema é que a análise de documentos se restringe ao código fonte dos programas, ou seja, documentos como fluxogramas e algoritmos que os aprendizes criam não são analisados. Isto poderia ser um fator que inviabilizasse esta pesquisa, mas é de conhecimento que, a quase totalidade dos aprendizes, parte direto para a codificação

no ambiente de desenvolvimento, pulando uma fase no processo de desenvolvimento de uma atividade ou programa.

A outra limitação do sistema, como já foi mencionada, é a análise apenas de métricas previamente conhecidas, mas o sistema provê todo o suporte para agregação da tecnologia de DataWarehouse, permitindo assim em uma outra fase a descoberta e análise de novas métricas.

Capítulo 6

Estruturação de um estudo de caso

A melhor maneira de verificar se a metodologia desenvolvida, se as métricas definidas e se o sistema irão auxiliar o professor no entendimento do processo de aprendizagem de programação, é a realização de um estudo de caso.

Para realização do estudo de caso, foi elaborado um curso de introdução ao C#, para ser ministrado como um segundo curso de programação, ou seja, um curso livre.

6.1 O curso de introdução a C#

O curso de introdução à linguagem de programação C# foi escolhido para esta pesquisa porque é uma linguagem de programação nova e que está tendo uma boa aceitação no mercado, com isso ficaria mais fácil motivar aprendizes a fazer o curso e conseqüentemente participar da pesquisa.

Para o curso foi elaborada uma apostila com as seguintes unidades pedagógicas:

- Unidade 1 - Visão Geral da Plataforma .NET;
- Unidade 2 - Introdução ao C#;
- Unidade 3 - Variáveis;
- Unidade 4 - Métodos e Parâmetros;
- Unidade 5 - Classes e Objetos e

- Unidade 6 - Programação Orientada a Objeto.

O curso possui também três atividades práticas, que são a base para a coleta de métricas no processo de aprendizagem de programação.

O público ideal para o curso são aprendizes sem ou com pouco conhecimento em orientação a objeto, assim é possível ter uma idéia de como eles se comportam num primeiro contato com linguagens de programação orientada a objeto.

6.2 Atividades propostas para o curso de introdução ao C#

O curso de introdução a C# foi utilizado como base para a coleta de métricas possui três atividades.

A seguir são apresentadas estas atividades já com a modelagem do aprendiz.

6.2.1 Primeira atividade

Domínio:

- Variáveis
- Conversões entre variáveis;
- Métodos e parâmetros
- Ambientes de nomes (namespace)

Questão: Calcular o valor aproximado de p.
p = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 .....)
Onde p corresponde ao cálculo de pi.
O usuário deve entrar com o número de casas decimais que deseja e o erro aceitável é 10$\hat{}$(-7).

Esta atividade possui três subproblemas:

1. Um contador de inteiros ímpares no dividendo da série;

2. A variação da operação ora soma ora subtração;

3. A questão do erro.

O subproblema do erro merece uma atenção especial:

Considerando o erro e como sendo:

$$e = \pi - p$$

Observa-se que e se comporta conforme figura 6.1.

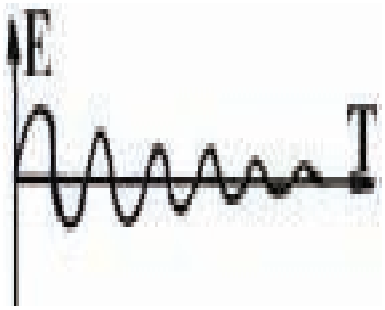


Figura 6.1: Comportamento do erro

Isto fará com que o aprendiz utilize apenas o módulo de e para verificar o final da série.

Como este subproblema não é explicito, o aprendiz só irá detectá-lo de duas maneiras:

1. Por tentativa e erro;

2. Utilizando o debug;

Este fato possibilitará ver como o aprendiz tentará solucionar este subproblema e ainda verificar se ele sabe utilizar os recursos do ambiente de desenvolvimento.

6.2.1.1 Tabela de modelagem

Para esta atividade consideraremos as seguintes interações para tentarmos entender a aprendizagem do aprendiz:

- **P1**: O uso do debug durante o desenvolvimento permitirá saber se o aprendiz adota o método "tentativa e erro"ou se tenta corrigir algum erro através da execução passo a passo, entendendo o que está acontecendo;
- **P2**: Saber o número de compilações efetuadas,
- **P3**: Saber os erros de cada compilação e erro mais freqüente;
- **P4**: Saber o tempo gasto para o desenvolvimento da atividade;

Assim tem-se a tabela da modelagem da função do conhecimento do aprendiz conforme tabela 6.1:

Interação	Número esperado
P1	Número de utilizações do debug pelo especialista com um erro de 20%
P2	Número de compilações realizadas pelo especialista com um erro de 20%
P3	Erros cometidos pelo especialista quando desenvolveu a atividade
P4	Tempo gasto pelo especialista com um erro de 20%

Tabela 6.1: Tabela de modelagem do conhecimento para a Atividade 1

6.2.2 Segunda atividade

Domínio:

- Variáveis
- Métodos e parâmetros
- Tipo de acesso
- Arrays
- Ambientes de nomes (namespace)

Questão: Implementar.
string[] palavras
void inserirPalavra(string palavra)
string[] retornarPalavras()
string[] retornarVerbos()
Considerar um verbo como uma palavra que termina com a letra r.

Para esta descrição, temos as seguintes considerações:
1. Um botão adicionar palavras;
2. Um textBox que onde se pode entrar palavras;
3. Um botão retornar verbos;
4. Um textBox onde serão apresentados os verbos;
5. Um botão retornar palavras;
6. Um textBox onde serão apresentadas as palavras;

Esta atividade possui cinco subproblemas:

1. Definição do tamanho do array;
2. Inserir um palavra em array;
3. Retornar um array;
4. Retornar verbos;
5. Converter um array em uma string;

O subproblema do tamanho do array possui duas soluções:

1. Limitar o tamanho do array;
2. Usar arrays dinâmicos.

6.2.2.1 Tabela de modelagem

Para esta atividade tem-se as seguintes interações para acompanhar a aprendizagem:

- **P1**: Não foi definido um tamanho para o array e array dinâmico não foi apresentado em sala, logo o aprendiz terá que supor um valor máximo e o código deve ter um tratamento básico de erro. Se o aprendiz optar por array dinâmico, podemos verificar se ele é capaz de entender um novo conceito, sozinho pelo help, e aplicá-lo;;

- **P2**: Através da coleta de dados durante o desenvolvimento podemos observar se o aprendiz irá desenvolver de forma ”evolutiva”, ou seja, dividir em subproblemas e resolver um a um;

- **P3**: Através da coleta de dados durante o desenvolvimento podemos observar se o aprendiz irá reaproveitar o código, por exemplo, copiar e colar o código do método mais simples e depois alterá-lo de maneira a atender a especificação do outro método ou se ele irá implementar os dois de maneira separada;

- **P4**: Saber o número de compilações efetuadas;

- **P5**: Saber os erros de cada compilação e erro mais freqüente;

- **P6**: Saber o tempo gasto para o desenvolvimento da atividade;

- **P7**: O uso do debug durante o desenvolvimento permitirá saber se aprendiz adota o método ”tentativa e erro”ou se tenta corrigir algum erro através da execução passo a passo, entendendo o que esta acontecendo.

Assim tem-se a tabela da modelagem da função do conhecimento do aprendiz conforme tabela 6.2:

Interação	Número esperado
P1	Possível acesso ao help por palavras chaves como array e dinamic
P2	Evolução de parâmetros, como número de funções, classes e variáveis formam um gráfico crescente
P3	Possível utilização do recurso copy e paste
P4	Número de compilações realizadas pelo especialista com um erro de 20%
P5	Erros cometidos pelo especialista quando desenvolveu a atividade
P6	Tempo gasto pelo especialista com um erro de 20%
P7	Número de utilizações do debug pelo especialista com um erro de 20%

Tabela 6.2: Tabela de modelagem do conhecimento para a Atividade 2

6.2.3 Terceira atividade

Domínio:

- Variáveis
- Métodos e parâmetros
- Tipo de acesso
- Propriedades
- Construtores
- Herança
- Classes e métodos abstratos
- Métodos virtuais
- Ambientes de nomes (namespace)

Questão: Implementar a hierarquia de classes da figura 6.2.

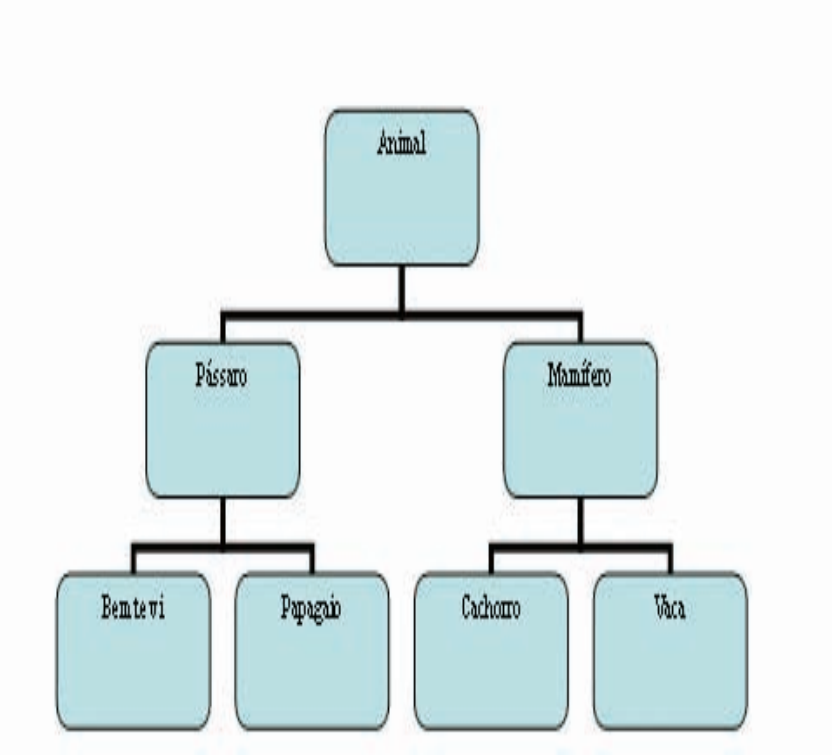


Figura 6.2: Hierarquia de classes da Atividade 3

Descrição das classes:

Animal:

string nome

void imprime() //imprime o nome do animal

string getNome()

void talk() //nada

Pássaro:

boolean voa // pode voar?

void setVoa(boolean v)

boolean getVoa()

void talk() // ”piu-piu”

Papagaio:

string vocabulario // frase que sabe falar

void talk() // imprime vocabulário

void setVoc(string s) // altera vocabulário

string getVoc() // retorna vocabulário

Bem te vi: void talk() // ”bem te vi”

Mamífero:

int patas // total de patas

int getPatas() // retorna total de patas

Cachorro:

boolean lateAlto // late alto ou baixo

void setLateAlto() // late alto

void setLateBaixo() // late baixo

void talk() // ”AU AU”ou ”au au”

Vaca:

boolean temLeite

void setLeite(boolean leite)

boolean getLeite()

void talk() // ”Muuu”

Esta atividade não possui subproblemas, pois se trata de uma simples implementação de um diagrama de classes, mas mesmo assim enquadra-se como problema do modelo de acompanhamento pois retrata um problema do mundo real.

6.2.3.1 Tabela de modelagem

Para esta atividade considera-se as seguintes interações para acompanhar a aprendizagem:

- **P1**: Através da coleta de dados durante o desenvolvimento podemos observar se o aprendiz irá desenvolver de forma ”evolutiva”, ou seja, dividir em subproblemas e resolver um a um;
- **P2**: Através da coleta de dados durante o desenvolvimento podemos observar se o aprendiz irá reaproveitar trechos e código semelhantes (copiar, colar e alterar);
- **P3**: Saber o número de compilações efetuadas;
- **P4**: Saber os erros de cada compilação e erro mais freqüente;
- **P5**: Saber o tempo gasto para o desenvolvimento da atividade;
- **P6**: O uso do debug durante o desenvolvimento permitirá saber se aprendiz adota o método ”tentativa e erro”ou se tenta corrigir algum erro através da execução passo a passo, entendendo o que esta acontecendo.

Assim tem-se a tabela da modelagem da função do conhecimento do aprendiz coforme tabela 6.2:

Interação	Número esperado
P1	Evolução de parâmetros, como número de funções, classes e variáveis formam um gráfico crescente
P2	Possível utilização do recurso copy e paste
P3	Número de compilações realizadas pelo especialista com um erro de 20%
P4	Erros cometidos pelo especialista quando desenvolveu a atividade
P5	Tempo gasto pelo especialista com um erro de 20%
P6	Número de utilizações do debug pelo especialista com um erro de 20%

Tabela 6.3: Tabela de modelagem do conhecimento para a Atividade 3

6.3 Método para a identificação de falsos conceitos em subproblemas

As atividades propostas e as respectivas tabelas de modelagem é uma tentativa de prever o que o aprendiz irá fazer durante o processo de resolução do problema. Mas, por exemplo, saber que ele utilizou o debug pode não ser suficiente para uma avaliação, é de grande importância saber se o debug foi utilizado de forma correta e em um trecho de código que normalmente necessite deste recurso. Para isso o professor terá que analisar mais profundamente esta interação, e o sistema que está implementado nesta pesquisa tem a capacidade de propiciar isto.

Outro caso, é o aprendiz não utilizar o debug e chegar a uma solução correta, logo caberá ao professor analisar as interações coletadas pelo sistema afim de ter uma avaliação correta sobre a aprendizagem do aprendiz.

O sistema coleta não somente as interações esperadas, mas todas as interações que o aprendiz realiza no ambiente de desenvolvimento e estas estão disponíveis para o professor poder realizar avaliações de forma mais precisa e correta.

Como o acompanhamento do aprendiz em programação com detecção de conceitos falsos ser algo novo, a pesquisa trabalha com pequenas quantidades de dados e com apenas dois tipos de resposta (verdadeiro, falso), a fim de validar os dados mais rapidamente.

Com isto tem-se a identificação de conceitos falsos (ALBANESE, 2000) conforme figura 6.3.

Mas no modelo do aprendiz, seu conhecimento é um subconjunto do conhecimento do especialista, logo foi definido um par de operações, para pontuar um grupo de interações, chamadas de: Operações de Baixa aproximação e Operações de Alta aproximação. Assim o método para a identificação foi revisado, e ilustrado na figura 6.4.

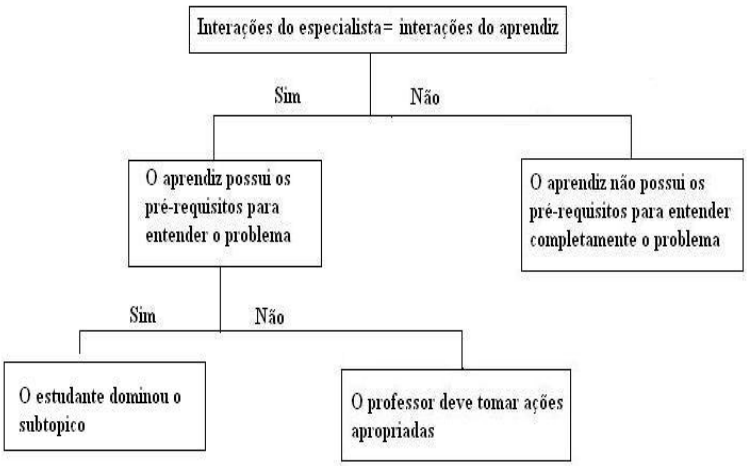


Figura 6.3: Identificação de conceitos falsos

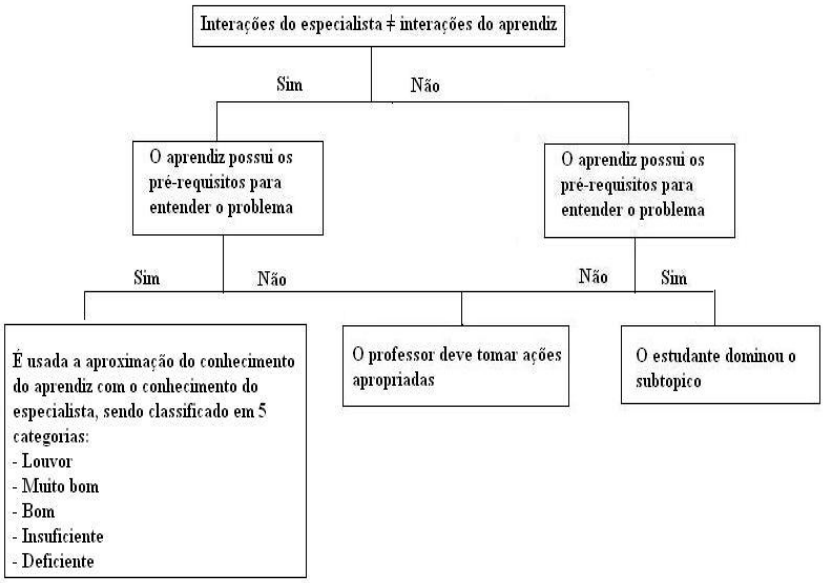


Figura 6.4: Identificação de conceitos falsos revisado

6.4 Classificação do entendimento

Usando a modelagem do sistema S=Q,P,V,F e X $\subseteq$ Q, para qualquer questão q $\in$ Q, podem-se dizer que:

Um domínio X é completamente entendido caso todas as interações do aprendiz (P) sejam iguais ao do especialista (Pe) para um conjunto de atividades.

A partir deste, cinco classes de entendimento para o problema X em S são determinadas.

- **Louvor** - Quando as interações realizadas pelo aluno são entre 90 - 99% das interações previstas pelo especialista;
- **Muito Bom** - Quando as interações realizadas pelo aluno são entre 70 a 90% das interações previstas pelo especialista. Neste contexto também se encaixam os aprendizes que conseguem resolver um problema de grande complexidade, envolvendo, por exemplo, o conteúdo de toda uma unidade, mas sem passar pelos conteúdos mais simples complementares;
- **Bom** - Quando as interações realizadas pelo aluno são entre 50 a 70% das interações previstas pelo especialista;
- **Insuficiente** - Quando as interações realizadas pelo aluno são entre 30 a 50% das interações previstas pelo especialista e
- **Deficiente** - Quando as interações do aprendiz são inferiores a 30% das interações do especialista para um conjunto de atividades; ou quando ainda não temos informações de resolução de problemas do aprendiz, ou seja, o aprendiz ainda se encontra na fase de assimilação do problema.

6.5 Critérios para avaliação do professor

Em muitos casos pode ocorrer do professor utilizar critérios diferentes na correção de uma atividade, dificultando futuras comparações entre aprendizes, grupos e/ou turmas. Para minimizar este fato, é necessário adotar alguns critérios para padronizar a correção das atividades em linguagem de programação. A partir destes critérios, que podem ou não ter pesos diferentes, uma tabela é gerada, onde o professor simplesmente irá checar os ítens como verdadeiro ou falso e a partir deste a nota do aprendiz será calculada.

A tabela esta dividida em duas partes:

1. Avaliação do resultado: onde será avaliado o código pronto entregue pelo aprendiz e
2. Avaliação do processo de aprendizagem: onde as métricas coletados durante o processo de aprendizagem serão avaliadas.

6.5.1 Avaliação do resultado

A avaliação do resultado é feita através dos seguinte ítens:

1. **Qualidade dos comentários** - não basta o aprendiz encher o programa de comentários, estes têm que ser relevantes ao trecho que correspondem:
   - Explicam a lógica de um trecho de código e não somente uma tradução do comando;
   - Em cabeçalhos de funções podem contem explicações sobre os parâmetros de entrada e saída e
   - Podem dizer como realizar um teste de validação;
2. **Organização do código** - o código tem que estar bem estruturado:
   - O código deve estar bem alinhado, por exemplo, não pode estar todo à esquerda;
   - Cada função deve executar uma única tarefa;
   - Utilização de ”nomes fortes”para variáveis, funções e classes;
   - Loops são complicados de analisar, neste trecho o código deve ser ”limpo”e
   - Expressões booleanas devem ser claras e parentizadas;
3. **Compilação** - o código deve ser compilado sem gerar nenhum erro ou warning;

Para reforçar a padronização da nota, a mesma escala para falsos conceitos é utilizada, logo os ítens principais (Qualidade dos comentários, Organização do código e Compilação) terão conceitos que irão de deficiente a louvor.

Cada subitem dos ítens principais recebem nota 1 ou 0 (verdadeiro ou falso) e tem esse valor multiplicado por um peso que o professor define por grau de importância do subitem. O somatório dos pesos dos subítens não pode ultrapassar o valor cinco, que corresponde ao conceito louvor. Assim a nota do item principal é o somatório das notas dos subítens multiplicadas pelo peso correspondente. A nota principal também tem um peso em relação à nota final.

Ao longo da pesquisa, foi analisado o software FxCop que é um analisador de código para programas feitos utilizando a tecnologia .NET. O intuito da utilização deste software era automatizar esta parte da avaliação, mas o principal fator que impediu sua utilização foi que ele analisa códigos apenas em inglês. Nomes de variáveis e funções teriam que estar em inglês para poderem ser validados.

Um analisador de comentários poderá ser desenvolvido, por exemplo, ele verifica a ocorrência de palavras-chave, mas para isso é necessária uma boa base de dados extraída de comentários feitos por aprendizes.

Em relação à compilação sem erros e warnings, o sistema pode pegar o resultado da ultima compilação realizada, desde que o professor indique que o programa atende os requisitos, pois um programa que não atende pode compilar sem erros e warnings também.

A figura 6.5 mostra como será feita a avaliação do resultado de um ítem principal.

	Nota aplicada	Peso	Nota Final
Qualidade dos comentários		Peso	NF
Explicação da lógica de um trecho de código	0 ou 1	p1	n1
Cabeçalhos de funções contém explicações dos parâmetros	0 ou 1	p2	n2
Descrição de caso de teste	0 ou 1	p3	n3
Organização do código			
Alinhamento do código			
Cada função executa uma única tarefa			
Utilização de "nomes fortes" para variáveis, funções e classes			
Loops com código limpo			
Expressões booleanas claras e parentizadas			
Compilação			
Compilação sem gerar nenhum erro ou warning			

Figura 6.5: Avaliação do resultado

Onde p1+p2+p3 = 5, n1 = p1* (0 ou 1), nn = pn * (0 ou 1).

A nota final (NF) do item será Peso * (n1+n2+...nn)

6.5.2 Avaliação do processo de aprendizagem

A avaliação do processo de aprendizagem segue a mesma lógica da avaliação do resultado, sendo que os ítens avaliados serão as possíveis interações do aprendiz em relação ao especialista.

A figura 6.6 mostra como será feita esta avaliação.

			Nota Aplicada	Peso	Nota Final
Interações do aprendiz					
P1			0 ou 5	pe1	n1
P2			0 ou 5	pe2	n2
P3			0 ou 5	pe3	n3
Pn			0 ou 5	pen	nn

Figura 6.6: Avaliação do processo de aprendizagem

Onde P1,P2,...Pn podem assumir valores 0 e 5, já que não possuem subitens. Em apenas um caso nesta pesquisa P pode assumir valores entre 0 e 5 que é o caso da comparação dos erros cometidos pelo aprendiz em relação ao especialista. Por exemplo, em uma atividade em que especialista não cometeu nenhum erro e o aprendiz cometeu erros de sintaxe e erros de conversão de variáveis. O valor de P neste caso será 3, ou seja, 5 menos a quantidade de grupos de erros que o aprendiz cometeu que difere dos erros cometidos pelo especialista durante o desenvolvimento.

6.5.3 Nota final da atividade

A nota final da atividade será dada pela média do somatório das notas finais dos ítens principais da avaliação do resultado e das notas finais da avaliação do processo de aprendizagem dividido pelo somatório dos pesos de cada ítem avaliado..

Para transformar a nota atribuída no conceito definido anteriormente é utilizada a seguinte escala de conversão:

- Nota superior de 4.5 corresponde ao conceito Louvor;

- Nota entre 3.5 e 4.4 corresponde ao conceito Muito Bom;
- Nota entre 2.5 e 3.4 corresponde ao conceito Bom;
- Nota entre 1.5 e 2.4 corresponde ao conceito Insuficiente;
- Nota inferior a 1.5 corresponde ao conceito Deficiente;

Capítulo 7

Resultados obtidos

Entender o comportamento do aprendiz durante a aprendizagem de linguagem de programação através da coleta de métricas diretamente no ambiente de desenvolvimento é algo complexo. É preciso então, transformar estas métricas na realidade encontrada pelo professor (dúvidas, dificuldades, etc...), com isso a coleta de dados se torna fundamental para a pesquisa, que no seu decorrer teve o curso de introdução ao C# ministrado quatro vezes para:

1. Identificar aprendizes que tiveram dúvidas e dificuldades e verificar se, através dos relatórios e da avaliação do modelo pedagógico, o sistema confirma esta situação, já que o professor estará presente durante a fase de resolução do problema nos primeiros cursos ministrados;

2. Identificar aprendizes que tiveram um comportamento esperado e verificar se, através dos relatórios e da avaliação do modelo pedagógico, o sistema confirma esta situação, já que o professor estará presente durante a fase de resolução do problema nos primeiros cursos ministrados;

3. Demonstrar que as informações coletadas irão auxiliar o professor no processo de avaliação do aprendizado;

Para cada um dos três exercícios propostos, foram sugeridas interações que o aprendiz possivelmente irá realizar na fase de resolução do problema. Através da comparação, do número de realizações ou não dessas interações, com as interações realizadas pelo especi-

alista quando este resolveu atividade é suposto um possível comportamento de aprendizagem.

Muitas interações, como por exemplo, quantidade de compilações, quantidade de uso do debug, tempo para resolução do problema, fornece métricas quantitativas e através destas a comparação com a métrica do especialista se torna viável, supondo um erro inicial de 20%. Como exemplo, caso o especialista resolva a atividade em 50 minutos, o sistema considerará que a aprendizagem foi bem sucedida; caso o aprendiz realizasse a atividade entre 40 e 60 minutos ou o especialista realizou 10 compilações, então o aprendiz teria que realizar entre 8 e 12 compilações.

Para se obter uma idéia inicial de uma margem de erro satisfatória, já que o ser humano reage de maneira diferente nas mesmas situações, inicialmente é proposto um erro aceitável de 20%, que pode ser ajustado com a análise de informações coletadas em casos reais de aprendizagem.

Para cada atividade um subconjunto das métricas quantitativas foi determinado, assim os valores poderão determinar um comportamento esperado do aprendiz.

Analisando apenas uma métrica quantitativa isoladamente, neste caso, três situações que devem ser consideradas:

1. O aprendiz que realizou a atividade numa quantidade menor de interações do que a esperada: nesta situação se encaixa três tipos de aprendizes:
   - O que já sabia o conteúdo da disciplina e resolveu facilmente;
   - O que tem uma facilidade ”acima da média”para a absorção do conteúdo apresentado;
   - O que copia a atividade de outro aprendiz.
2. O aprendiz que realizou a atividade com um número de interações esperado:
   - Este aprendiz está tendo um comportamento conforme esperado e previamente identificado como padrão de aprendizagem;
   - O que copia a atividade de outro aprendiz.
3. O aprendiz que realizou a atividade com um número de interações superior ao esperado: nesta situação há três tipos de aprendizes:

- O que não conseguiu realizar a atividade;
- O que teve uma maior dificuldade em um determinado tópico, gastando mais tempo para resolver o subproblema correspondente, mas que conseguiu resolver o problema e teve um aprendizado positivo;
- O que copia a atividade de outro aprendiz.

O aprendiz que copia a atividade se encaixa nas três situações, portanto é algo não muito fácil de ser identificado.

Observando a análise de uma métrica é suficiente apenas para o caso de obtermos o valor esperado, porque nas situações diferentes desta há casos em que a aprendizagem foi bem sucedida e o sistema não interpretaria deste jeito. Neste caso identificar uma outra métrica, com um valor esperado pré-determinado, que associada a esta que está sendo analisada poderá identificar a aprendizagem bem sucedida. Provavelmente em alguns casos esta associação seria bem sucedida, em outros seria necessária à adição de mais métricas.

Mas nesta situação, ocorre o mesmo caso do valor da métrica associada não estar no intervalo esperado, por isso a análise do professor é sempre fundamental.

O sistema, então, disponibiliza todas as métricas quantitativas coletadas durante a fase de resolução do problema para o professor. Provavelmente, em muitas situações, o professor irá encontrar a realidade que ele vê na sala de aula enquanto ministra a parte teórica. Ele pode perceber que determinado aprendiz está tendo dificuldade em certa unidade pedagógica e comprovar isso com os dados da fase de resolução do problema, seja pela análise de uma única métrica ou de varias, da maneira que julgar conveniente.

A experiência do professor com a utilização do sistema irá auxiliar na determinação de novas combinações de métricas e seus respectivos valores, a fim de, determinar um comportamento esperado e conseqüentemente facilitar a avaliação da aprendizagem.

Um outro tipo de métrica, a qualitativa, irá auxiliar o professor nestes casos em que os valores estão fora do esperado, por exemplo: o aprendiz utilizou apenas uma vez o debug enquanto o esperado era entre 2 e 4 vezes. Em vez de procurar outras métricas o professor poderá analisar a qualidade desta utilização, por exemplo, a utilização foi em um trecho de código que realmente necessitava de um debug? O aprendiz monitorou

valores de variáveis? Ou seja, o uso correto do debug mostra informações ao professor que a aprendizagem foi bem sucedida.

A avaliação da aprendizagem em linguagens de programação é extremamente útil, pois atualmente o professor avalia apenas o aprendizado, ou seja, se a atividade entregue funciona corretamente ou não. Muitas vezes o professor exige uma apresentação e durante esta faz perguntas sobre determinado trechos do código, para saber se realmente foi o aprendiz que fez a atividade. Alguns professores olham os códigos entregues procurando semelhanças, procurando os aprendizes que fazem a atividade para o colega. E após isso aplicam uma prova para consolidar o aprendizado. Nesta situação o professor não está preocupado se ele está adotando a melhor estratégia de ensino para a turma.

Muitos aprendizes não entendem durante a aula teórica e não manifestam isso ao professor, seja por qualquer motivo. Com a análise da resolução do problema, o professor tem condições de avaliar a estratégia adotada, podendo sugerir a necessidade de ação corretiva. Portanto ele também está avaliando o próprio desempenho.

Muitas vezes uma prova escrita não traduz a realidade dos conhecimentos do aprendiz, pois podem cair questões que o aprendiz previamente estudou, ou mesmo decorou a lógica e caso caísse uma variação ele não saberia resolver. E em uma prova escrita não há possibilidade do aprendiz utilizar uma série de recursos hoje disponíveis para auxiliar no desenvolvimento de código (debug, help, etc...), pois, visto que as linguagens atuais são complexas e extensas é praticamente impossível o aprendiz decorar todos os recursos oferecidos por elas.

7.1 O primeiro curso de introdução ao C#

O primeiro curso de introdução ao C# foi ministrado para 24 aprendizes de uma instituição de ensino técnico.

O perfil dos aprendizes, apresentado pelo coordenador do curso da instituição, era da seguinte maneira:

- 18 aprendizes que haviam tido um semestre de lógica de programação e estavam no final do curso de ASP;

- 6 aprendizes que haviam tido um semestre de lógica de programação, um semestre de ASP e estavam no final do curso de Visual Basic 6.0;
- Nenhum aprendiz havia tido contato com linguagens .NET da Ms.

Analisando o perfil dos aprendizes chegou-se a conclusão que a estrutura do curso elaborado, incluindo as atividades, estava adequada. Mas logo na primeira aula ministrada foi verificado que a realidade dos aprendizes era outra:

- Os 18 aprendizes que teoricamente já tinham tido um semestre de lógica de programação não tiveram um curso completo, por exemplo, não viram laços de repetição e estavam vendo durante o curso de ASP;
- Os aprendizes que estavam no final do curso de Visual Basic 6.0, não tiveram lógica de programação, pois a grade curricular da escola havia sido alterada há pouco tempo para sanar essa deficiência e no curso de VB viam apenas componentes visuais.

Com esta nova realidade, o curso elaborado não estava adequado à realidade dos aprendizes, pois o curso não abrange a teoria de lógica de programação, como laços de repetição e comandos condicionais. Mesmo assim, a primeira atividade foi proposta aos aprendizes.

Era esperado, portanto:

- A quase totalidade dos aprendizes com muita dificuldade para realização da atividade e conseqüentemente uma avaliação baixa pelo sistema;
- Identificar aprendizes que sobressaíssem ao grupo;
- Comparar os aprendizes com dificuldades e os que sobressaíssem pelo sistema com as impressões do professor.

7.1.1 A primeira atividade

O primeiro passo para avaliação foi definir os pesos para os ítens que seriam avaliados. Um quadro conforme figura 7.1 foi gerado para avaliar a primeira atividade realizada pelos aprendizes.

	Peso	Nota Padrão	Nota Aplicada		
Análise do código entregue					
Qualidade dos comentários	**1**	**5**			
Explicação da lógica de um trecho de código	2	1			
Cabeçálhos de funções contém explicações dos parametros	2	1			
Descrição de caso de teste	1	1			
Organização do código	**1**	**5**			
Alinhamento do código	1	1			
Cada função executa uma única tarefa	1	1			
Utilização de "nomes fortes" para variáveis, funções e classes	1	1			
Loops com código limpo	1	1			
Expressões booleanas claras e parentizadas	1	1			
Compilação	**2**	**5**			
Compilação sem gerar nenhum erro ou warning	1	5			
Análise do processo de aprendizagem				**Valor Esperado**	**Valor obtido**
Número de utilizações do debug pelo especialista com um erro de 20%	**2**	5		Entre 1 e 3	
Número de compilações realizadas pelo especialista com um erro de 20%	**1**	5		Entre 6 e 10	
Erros cometidos pelo especialista quando desenvolveu a atividade	**2**	5		Erros de sintaxe	
Tempo gasto pelo especialista com um erro de 20%	**1**	5		Entre 12 e 18	
NOTA FINAL					

Figura 7.1: Quadro para avaliação da primeira atividade

No quadro já consta os valores que o especialista obteve resolvendo a atividade, para comparar com os dados do aprendiz. Alguns ítens receberam peso dobrado, pois foi julgado estes como sendo de maior importância para a nota final:

- Entregar o programa funcionando sem nenhum erro e atendendo os requisitos propostos;
- Utilização do debug, pois assim o aprendiz está realmente tentando resolver um subproblema e utilizando os recursos disponíveis do ambiente;
- Os erros cometidos indicam a dificuldade em determinada unidade instrucional.

O tempo gasto e a quantidade de compilações não receberam um peso maior porque

era imaginado que em um primeiro contato estes valores seriam muito superiores ao do especialista.

Apenas 17% dos aprendizes entregaram a atividade de acordo com os requisitos, justamente os aprendizes que estavam no final do curso de VB. Os outros 83% apresentaram muitas dificuldades e a maioria delas coincidiu com as impressões do professor.

Analisando, primeiramente, alguns casos de aprendizes que apresentaram dificuldades.

Iniciando a análise do aprendiz chamado aqui de A1. Ele desenvolveu a atividade em 154 minutos, aproximadamente dez vezes mais que o tempo gasto pelo especialista, fato este já esperado. Segundo impressões do professor este aprendiz teve muita dificuldade no entendimento das unidades envolvidas nesta atividade. Pelo sistema há condições de confirmar isto:

1. Quantidade elevada de erros por compilação, como mostra a figura 7.2.

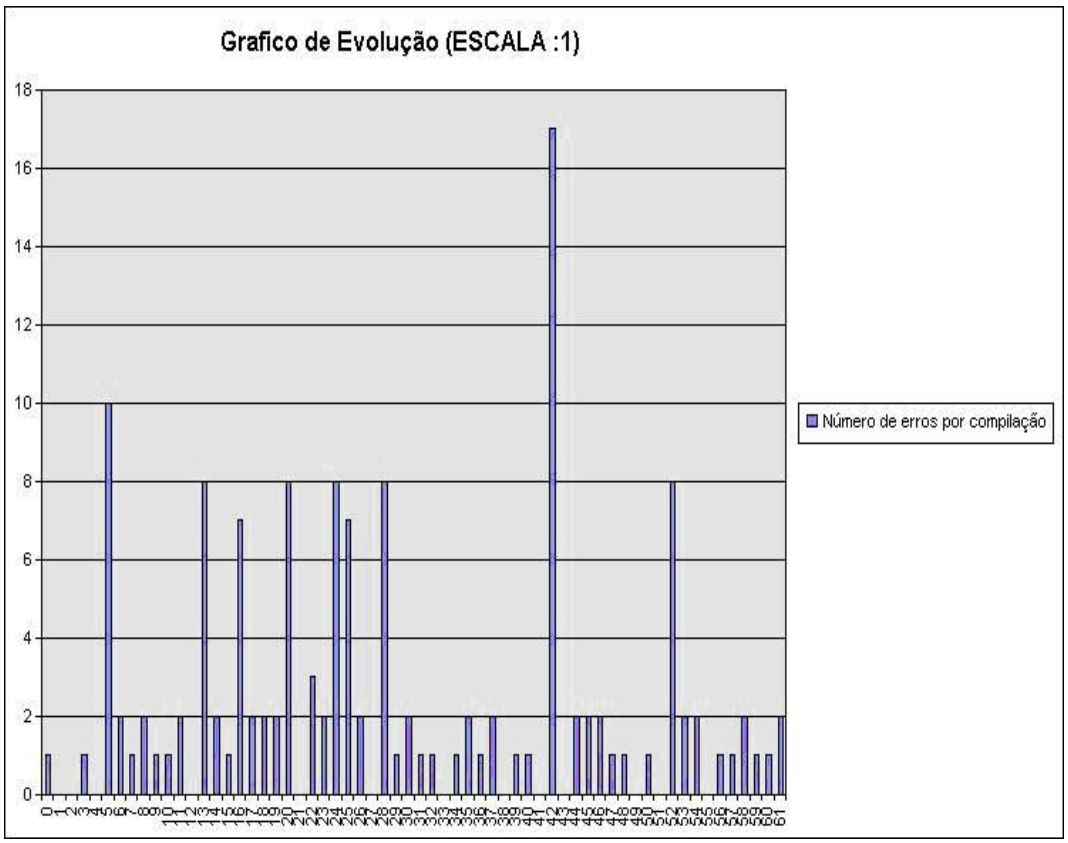


Figura 7.2: Erros por compilação do aprendiz A1

2. Extrema dificuldade em entender a unidade sobre variáveis e conversões, o que pode ser comprovado pelas ocorrências desses tipos de erros, conforma figura 7.3.

Erro	Número de ocorrências
Constant value '1' cannot be converted to a 'bool'	24
Cannot implicitly convert type 'double' to 'bool'	17
Constant value '-1' cannot be converted to a 'bool'	10
Cannot implicitly convert type 'int' to 'bool'	9
Operator '*' cannot be applied to operands of type 'int' and 'bool'	5
Operator '^' cannot be applied to operands of type 'bool' and 'int'	3
Operator '*' cannot be applied to operands of type 'double' and 'bool'	2
Operator '==' cannot be applied to operands of type 'bool' and 'int'	1
Operator '+' cannot be applied to operands of type 'bool' and 'double'	1
Operator '-' cannot be applied to operands of type 'bool' and 'double'	1
Operator '*' cannot be applied to operands of type 'string' and 'string'	1
Constant value '0' cannot be converted to a 'bool'	1

Figura 7.3: Ocorrência de erros no desenvolvimento da atividade por A1

3. Nenhum acesso ao help;

4. Nenhuma utilização do debug;

5. As figuras 7.4, 7.5 e 7.6 mostram a evolução do código fonte desenvolvido por A1.

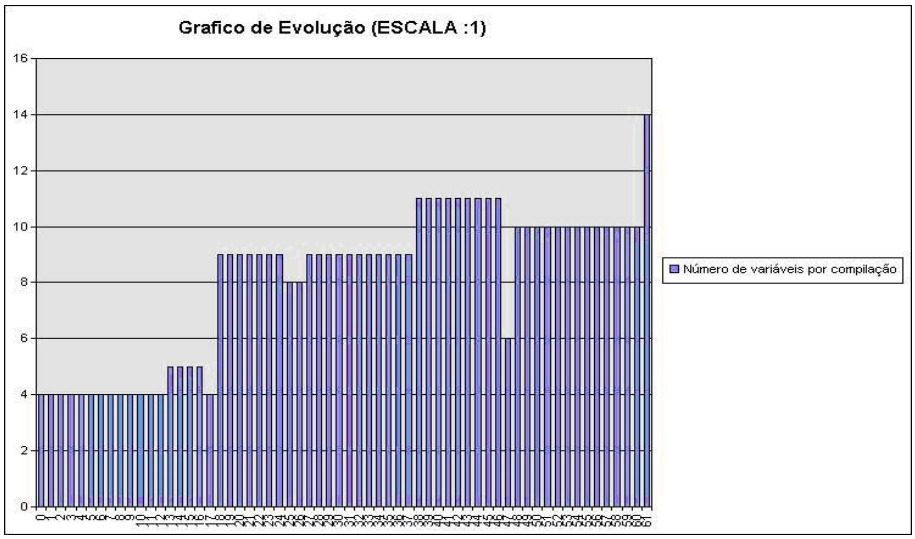


Figura 7.4: Número de variáveis por compilação de A1

Como não houve utilizações do debug, pode-se deduzir que este aprendiz estava usando o método ”tentativa e erro”. Pela grande variação do número de variáveis e linhas de código ele provavelmente pensou em algum algoritmo para resolução do

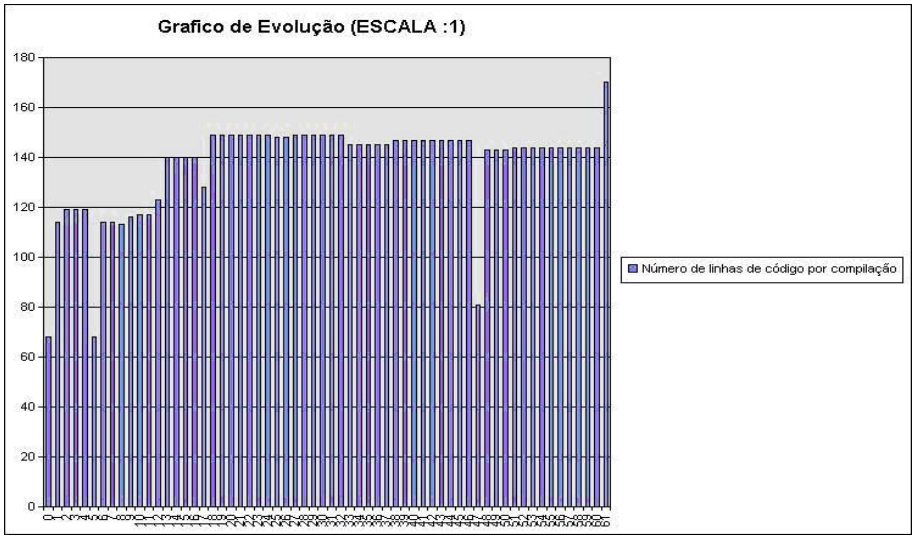


Figura 7.5: Número de linhas de código por compilação de A1

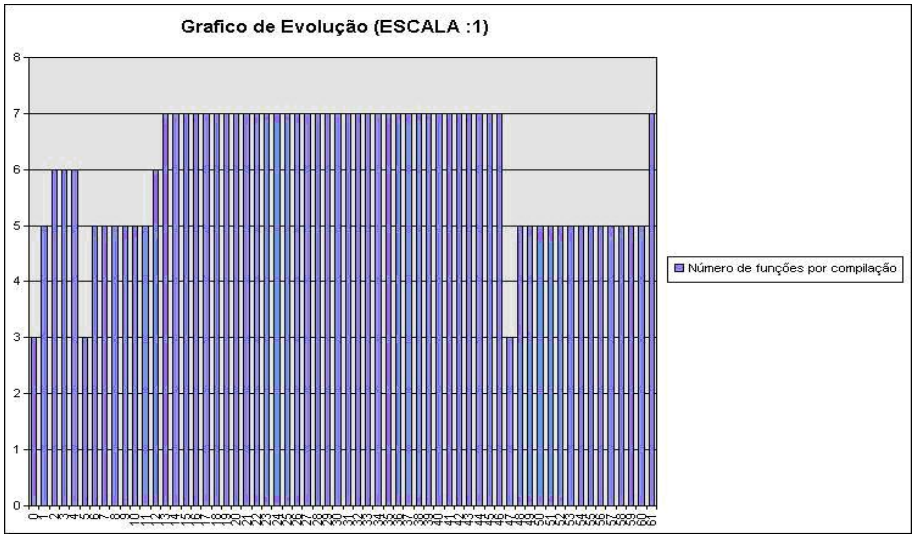


Figura 7.6: Número de funções por compilação de A1

problema e ao longo do desenvolvimento foi fazendo adaptações, mas não de uma forma pensada e sim intuitiva.

A variação do número de funções é explicada por dois motivos:

(a) Observando o código entregue foi constatada a presença de funções relativas a tratamentos de eventos, mas sem nada implementado;

(b) Como era o primeiro contato do aprendiz com o ambiente de desenvolvimento VS, e provavelmente, em todo momento ele realizava um duplo clique na área de design e isto gera automaticamente o código de tratamento de um evento (que é uma função), por isso o número de funções variou muito e esteve sempre

num número muito acima do esperado;

Esta variação do número de funções foi constatada na maioria dos aprendizes.

A aprendizagem de A1 teve o conceito deficiente dado pelo sistema. Segundo impressões do professor o conceito atribuído condiz com a realidade, pois o aprendiz não possuía os pré-requisitos necessários para a atividade.

O aprendiz A2 entregou a atividade funcionando, mas copiou a solução de um subproblema de um colega pois não conseguia resolvê-lo. Este aprendiz gastou 153 minutos para desenvolver a atividade.

Apesar de ter tido dificuldades, A2 teve mais facilidade que A1, como observado pelo número de erros por compilação, conforme figura 7.7.

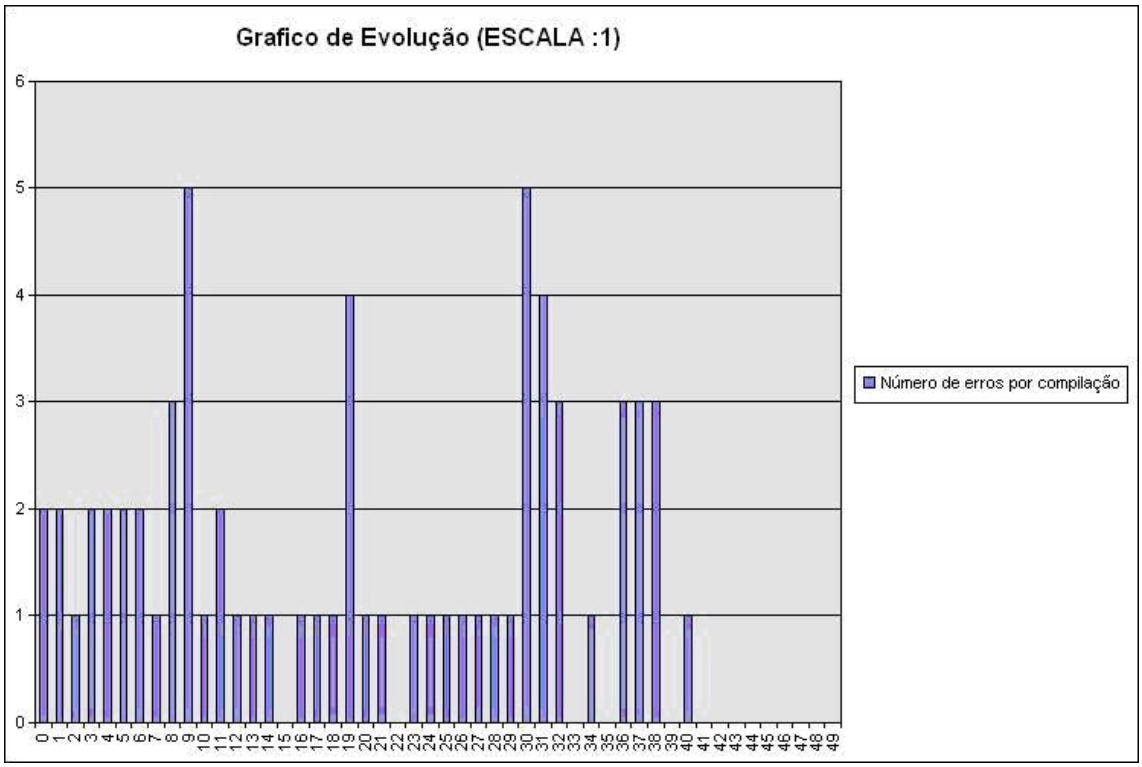


Figura 7.7: Erros por compilação do aprendiz A2

Há apenas dois picos no gráfico, mas com o detalhamento é possível verificar que eram erros de sintaxe, como por exemplo, falta de ponto-e-vírgula, fato já esperado por ser o primeiro contato com a linguagem de programação C#.

Os erros em sua maioria eram erros de sintaxe e erros de conversão, o que indica uma dificuldade nesta unidade pedagógica.

As últimas 9 compilações foram bem sucedidas, sem erros, indicando que o aprendiz estava testando o código, ou seja, aplicando alguns casos de testes que ele mesmo imaginou.

Embora a evolução do código não tenha ocorrido de maneira constante, é percebido que o aprendiz gerou um código equivalente a um algoritmo que ele imaginou e ao longo do desenvolvimento foi fazendo adaptações de forma intuitiva, conforme figura 7.8.

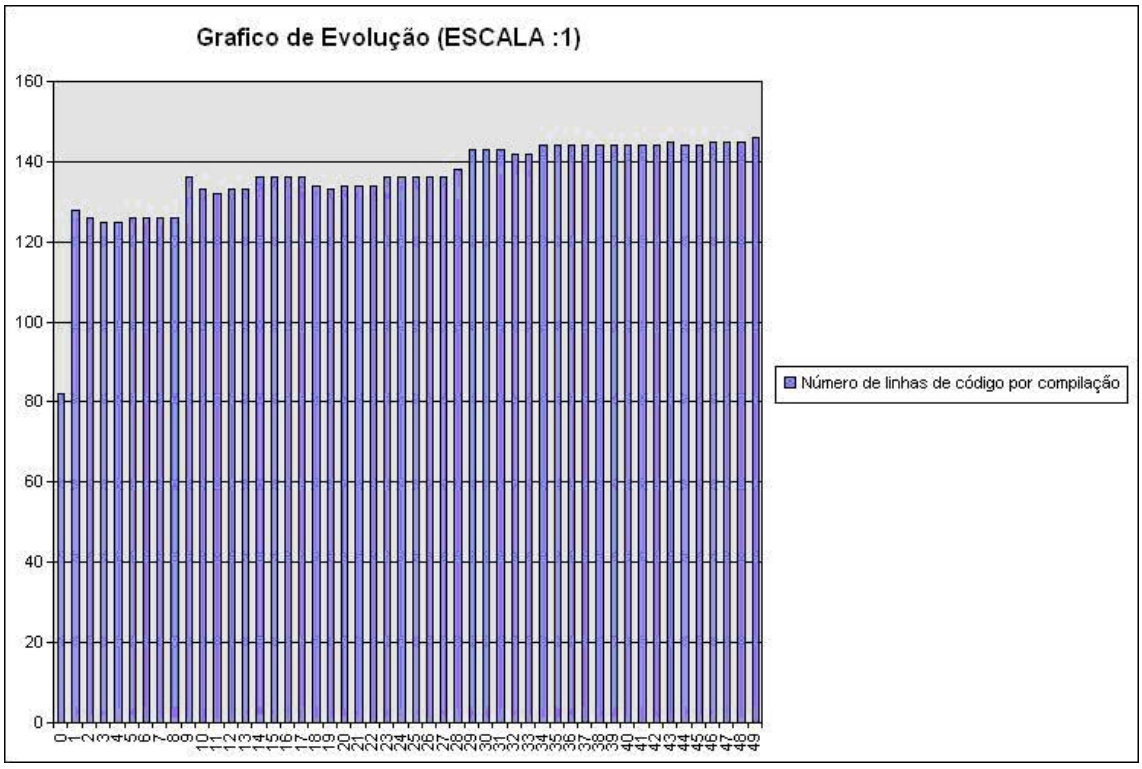


Figura 7.8: Evolução do código fonte do aprendiz A2

O aprendiz A2, não soube utilizar o debug apesar de utilizá-lo duas vezes durante o desenvolvimento da atividade. Nas duas vezes, o aprendiz marcou como breakpoint a linha onde era resolvida o subproblema do erro, justamente o que ele não conseguiu fazer. Mas durante um loop ele testou apenas uma interação ambas as vezes e não monitorou nenhuma variável.

Apesar de estar no mesmo nível de A1, ambos eram da mesma classe, claramente A2 teve menos dificuldades. Dos três subproblemas que a atividade continha, apenas uma ele não conseguiu realizar. Se analisada a relação entre a utilização do debug e compilações bem sucedidas em relação ao tempo, é visto que após a tentativa de uso de debug, passaram-se dez minutos e houve então as nove compilações bem sucedidas. Isto é um indicador que o aprendiz copiou a atividade pois ele não soube fazer o debug.

O sistema classificou a aprendizagem como insuficiente, mas por um décimo não foi classificado como bom. Isto reflete a realidade, pois apesar de ter entregue a atividade funcionando, o aprendiz não a fez sozinho e não soube utilizar os recursos disponíveis.

O restante dos aprendizes que tentaram, como A1 e A2, resolver a atividade não fugiram desses dois padrões, variando apenas métricas como tempo gasto e número de compilações, mas nunca chegando próximo ao especialista.

Analisando agora dois aprendizes, A3 e A4, que não fizeram a atividade e a entregaram de acordo com os requisitos, ou seja, copiaram de colegas.

A análise é iniciada por A3, que gastou 87 minutos para desenvolver a atividade. Claramente pela figura 7.9 é observado que algo está incoerente no desenvolvimento da atividade.

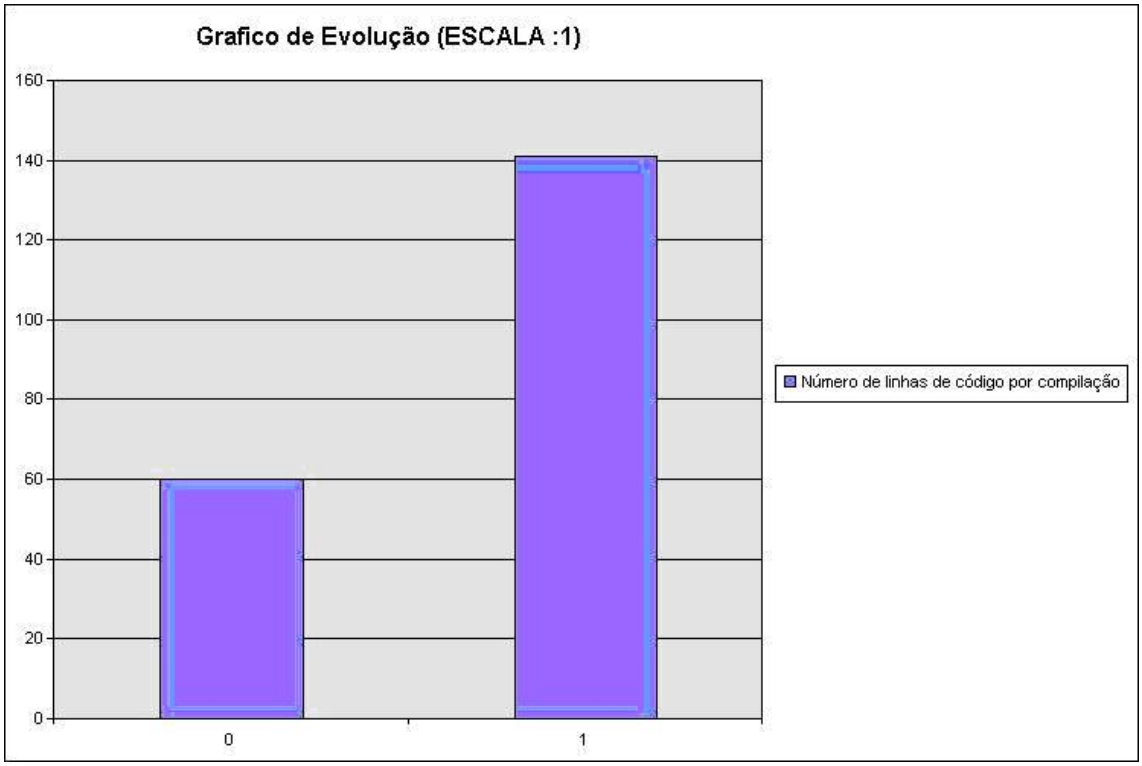


Figura 7.9: Evolução do código fonte do aprendiz A3

O aprendiz realizou apenas duas compilações, e pelo o horário delas, é visto que foram ao final do tempo gasto para realizar a atividade. Conclui-se então : ou o aprendiz tinha total domínio do conteúdo da atividade, que não é o caso, ou ele simplesmente esperou outro aprendiz resolver para depois copiar.

A aprendizagem foi classificada pelo sistema como deficiente.

Analisando os dados coletados de A4, que gastou 97 minutos para desenvolver a atividade, é possível perceber que ele também não a realizou.

A principio a figura 7.10 mostra um desenvolvimento esperado. Com 28 compilações, poucos erros concentrados mais no inicio, o que sugere uma primeira dificuldade que foi sendo entendida ao longo do tempo.

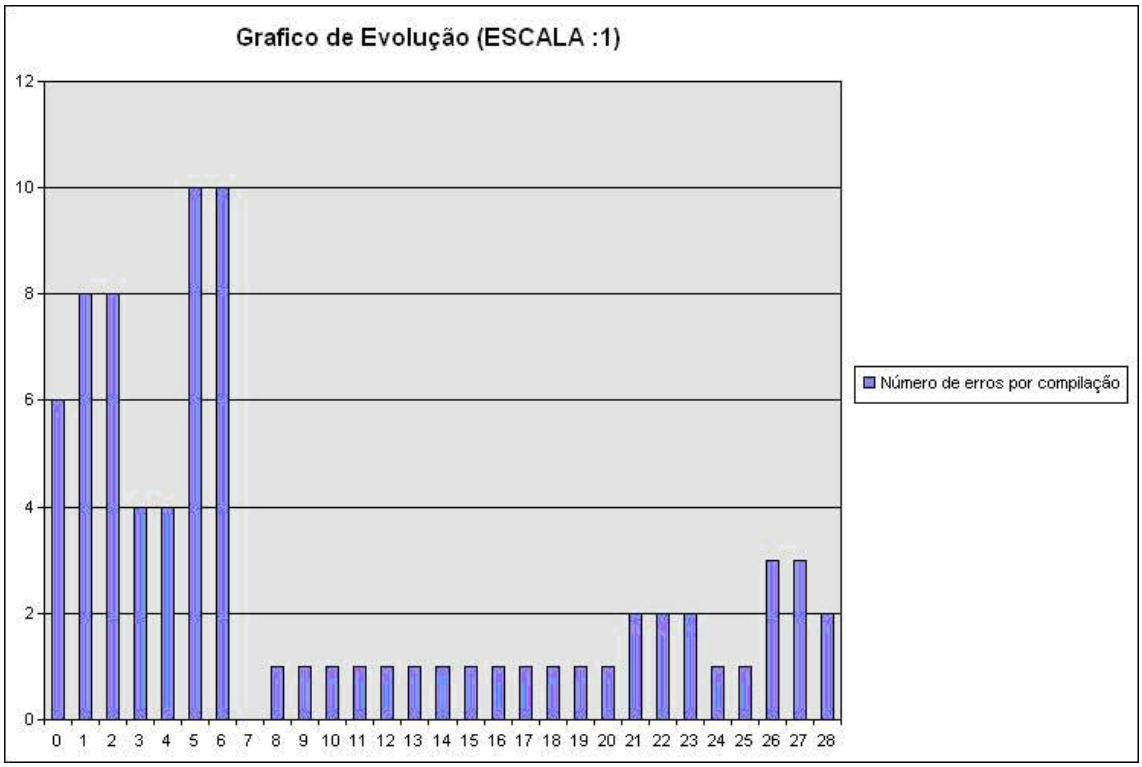


Figura 7.10: Erros de compilação do aprendiz A4

Olhando os erros ocorridos e a freqüência destes, é visto que em sua maioria são erros de variáveis e componentes não definidos.

Se apenas existissem erros de variáveis não definidas poderíamos dizer que o aprendiz está com uma dificuldade nesta unidade, mas o erro de componente não declarado indica algo. Se fosse um nome, como no exemplo, form1 e Form1 era aceitável, pois o C# difere maiúsculas de minúsculas. Mas no código, percebe-se que o aprendiz deu o nome do componente de form1 e o erro indica que "form_pi"não existe. E outro aprendiz utilizou este nome, caracterizando, portanto que A4 copiou o código e provavelmente não copiou o trecho onde havia a declaração das variáveis, por isso os erros.

Para reforçar esta hipótese, há a possibilidade de observar os intervalos de tempo entre as compilações, que ora variam entre 2 e 3 minutos ora variam entre 20 e 25 minutos.

A primeira compilação após um grande intervalo de tempo sempre apresenta erros de declaração de variável e componentes.

A aprendizagem foi classificada como deficiente pelo sistema.

Analisando agora um aprendiz, A5, que fez a atividade e a entregou atendendo os requisitos.

O aprendiz A5 gastou 87 minutos para desenvolver a atividade e entregou um programa que realizava mais coisas do que foi solicitado, isto já indica um prévio conhecimento em programação.

Apesar do número elevado de compilações, 66, como mostra a figura 7.11, isto é explicado pelo fato do aprendiz ter feito mais do que foi pedido.

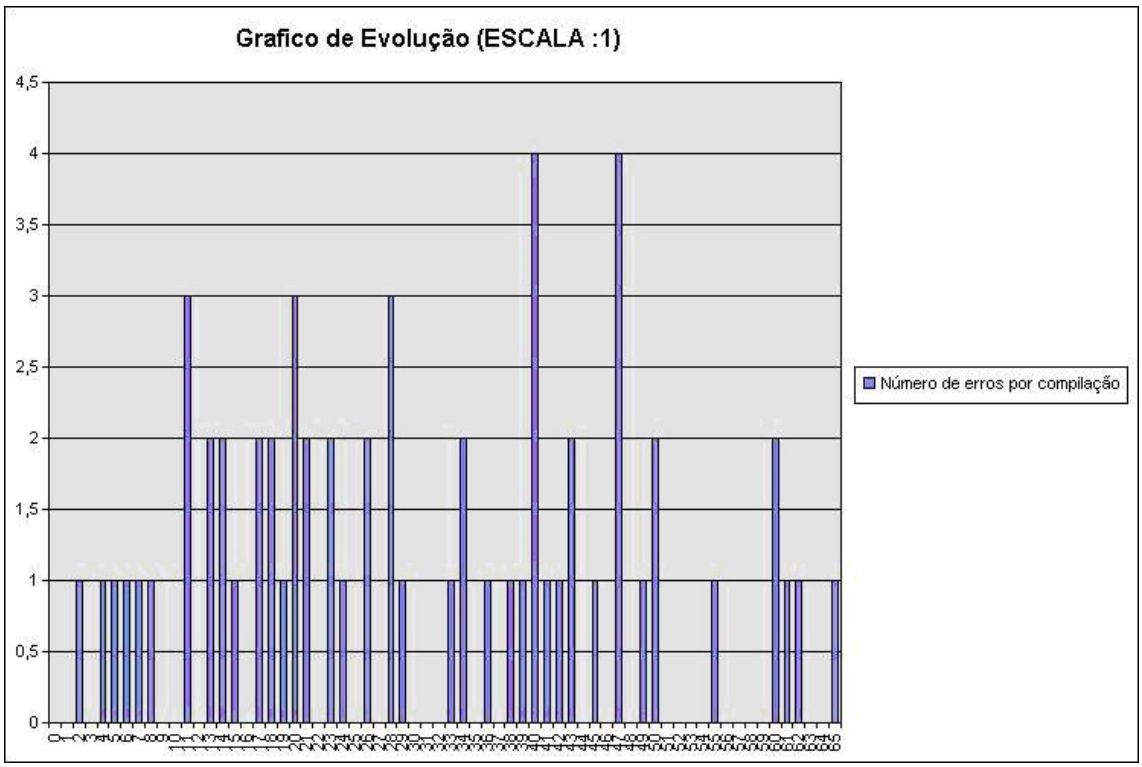


Figura 7.11: Erros de compilação do aprendiz A5

É observado um baixo número de erros por compilação e estes se concentraram em dois tipos:

1. Erros de sintaxe, que já era esperado;
2. Erros de variáveis e conversão do tipo booleanas, provavelmente o aprendiz não teve contato com variáveis do tipo booleana nos cursos de programação anteriores.

O aprendiz não utilizou o help, mas conseguiu, por exemplo, fazer uma mensagem de apresentação por ”tentativa e erro”que pode ser verificada com os erros de compilação.

O aprendiz utilizou variáveis globais e locais, o que não faz parte do curso, indicando o prévio conhecimento de programação. Também foi utilizado recursos disponíveis como o debug em trechos relevantes. Em um determinado momento ele usou o recurso quatorze vezes no mesmo trecho de código, indicando aqui uma certa dificuldade para resolver este subproblema.

A aprendizagem deste aprendiz foi classificada pelo sistema como insuficiente, mas isto é justificável, pois o aprendiz fez mais do que fora pedido e os parâmetros de comparação (especialista) não previam isto. Cabe aqui então, ao professor atribuir um conceito melhor.

7.1.1.1 A turma

Ao analisar as métricas coletadas, conclui-se, que na realidade apenas um aprendiz realizou a atividade e os outros ou copiaram ou não conseguiram fazer.

O tempo gasto para o desenvolvimento gasto em média pela turma foi de 113 minutos, um número bem superior ao gasto pelo especialista, mas isto já era esperado, pois era um primeiro contato com o ambiente do vs e da linguagem de programação C#. Nenhum aprendiz realizou comentários no código.

O gráfico de evolução do código fonte dos aprendizes foi conforme figura 7.12, ou seja, houve um desenvolvimento total inicial e depois apenas adaptações no algoritmo gerado.

A turma não possuía os pré-requisitos necessários para a atividade, logo a estratégia do curso também não estava adequada, por exemplo, o curso deveria abordar mais a parte de variáveis, laços de repetição, que foram ítens onde os aprendizes tiveram muita dificuldade, visto que não tiveram um curso adequado de lógica de programação.

Uma atividade mais simples, com menos subproblemas, talvez fosse a mais adequada, tanto pelo grau de dificuldade quanto para motivar os aprendizes, pois se já na primeira atividade não houve sucesso isso pode ter desmotivado alguns aprendizes.

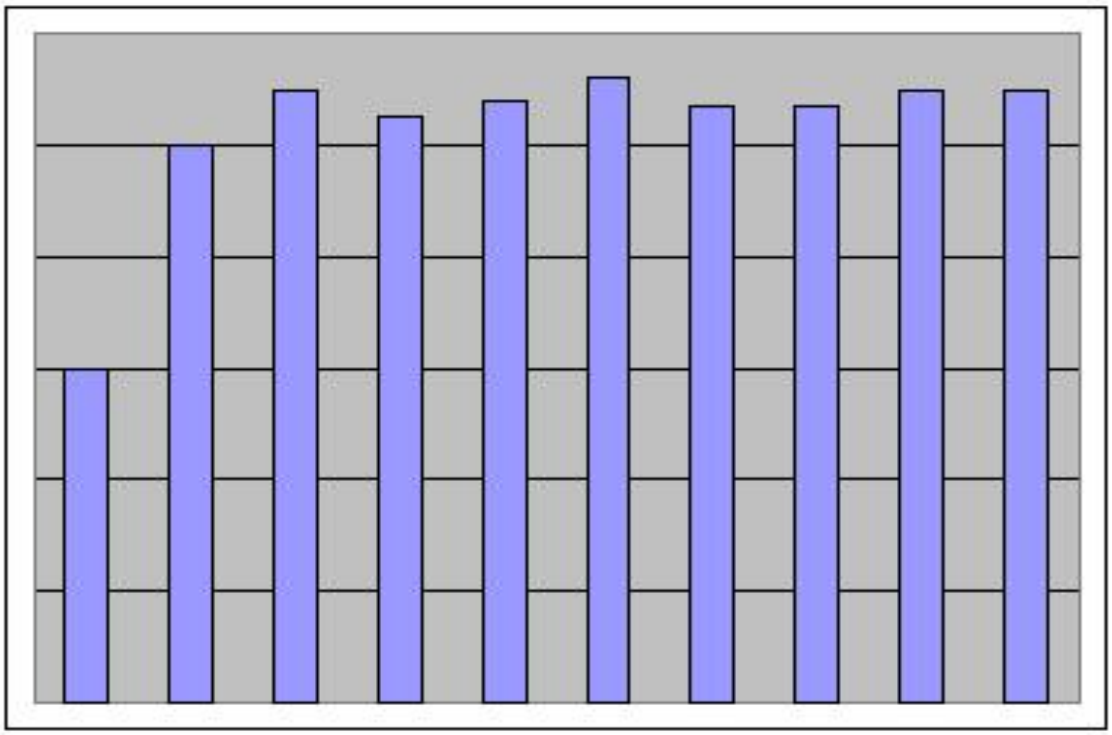


Figura 7.12: Gráfico da métrica de evolução da turma 1.

7.1.2 A segunda e a terceira atividade

Infelizmente neste curso houve um problema de gerenciamento de licenças e as métricas coletadas para a segunda e terceira atividades ficaram comprometidos, sendo assim, não foi possível ter conclusões do sistema, sobre a evolução do processo de aprendizagem destes aprendizes.

7.2 A segunda sessão de cursos de introdução ao C#

Para validar os dados coletados na primeira turma, um segundo curso de introdução ao C# foi proposto para aprendizes de graduação e pós-graduação que tivessem um perfil parecido com os aprendizes da primeira turma, ou seja, conhecimento de lógica de programação, pouco conhecimento em programação e nenhum contato com a linguagem C#.

Devido a grande procura, visto o tamanho do laboratório disponível para o curso, sete computadores, houve a criação de três novas turmas, sendo que cada turma teria uma aula por semana, um dia por semana. Com este formato se torna viável analisar os

resultados coletados e caso o professor julgasse necessário melhorar a estratégia do curso para a próxima turma e, já analisar os resultados para verificar se a nova estratégia foi adequada, caso não, esta poderia ser revista e adaptada para a terceira turma. Por este motivo estes três cursos foram agrupados em uma sessão.

Nesta sessão de cursos ministrados não houve os mesmos problemas do primeiro curso, ou seja, todos os aprendizes possuíam o perfil ideal e não houve nenhum problema com o sistema que inviabilizasse a coleta de métricas.

7.2.1 A primeira atividade

A fim de comparar os resultados obtidos com o do experimento anterior, foi utilizado o mesmo quadro de avaliação utilizado na primeira atividade.

Ao contrário da primeira turma, nos três cursos ministrados em seguida, houve um aproveitamento esperado:

- 10% dos aprendizes com conceito insuficiente;
- 70% com conceito bom e
- 20% dos aprendizes com conceito muito bom.

Isto já é um fator que indica que a estratégia adotada estava adequada.

Três aprendizes foram escolhidos para analise do processo de aprendizagem ao longo do curso, assim há a viabilidade de observar suas evoluções. Mas no final desta análise, também é analisada toda a turma e alguns casos que julgados interessantes.

Iniciando a análise com o aprendiz S1, que teve conceito bom nesta atividade, mas a entregou o código sem atender todos os requisitos, ou seja, não completou toda a atividade.

O código entregue estava muito próximo da solução, faltava apenas um pequeno ajuste no subproblema do erro. O aprendiz não terminou porque tinha outro compromisso agendado e não podia ficar. Vale aqui ressaltar que nesta fase o desenvolvimento da atividade foi supervisionado pelo professor.

O aprendiz S1 entregou um código limpo, bem organizado e estruturado, diferente dos aprendizes da primeira turma. Apesar do código não estar completo este não possuía

nenhum erro, apenas um warning quando se compilava. Utilizou o debug uma vez, justamente na parte onde não terminou, e cometeu poucos erros de inicialização de variáveis, comprovando a impressão do professor, já que o aprendiz fez algumas perguntas durante o desenvolvimento referente a este tópico.

Foram realizadas oito compilações, o que está de acordo com o número de compilações realizadas pelo especialista. O tempo total gasto para o desenvolvimento foi de 65 minutos.

A evolução do código fonte seguiu o mesmo padrão da turma anterior, conforme figura 7.13.

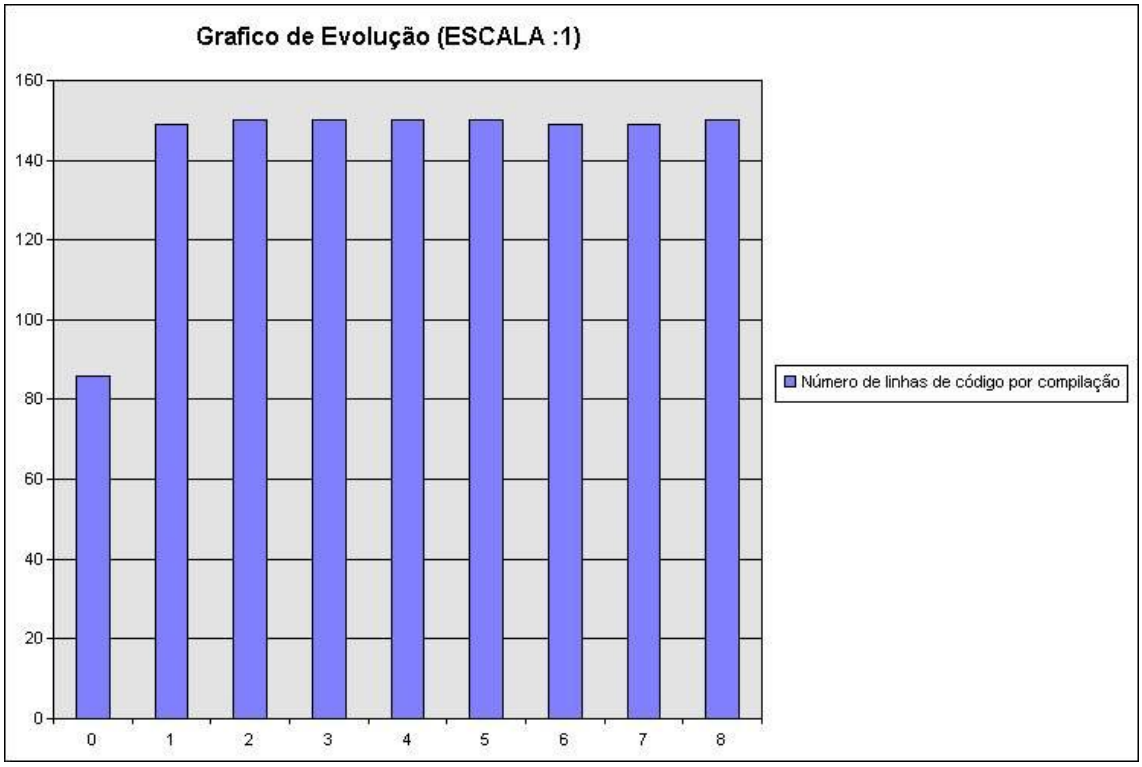


Figura 7.13: Gráfico da métrica de evolução do aprendiz S1.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, apesar do código entregue não estar completo. A tabela 7.1 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	65/15 = 4.33 (433%)
Quantidade de items avaliados com nota máxima em relação ao total possível	3/7 = 0.43 (43%)

Tabela 7.1: Parâmetros de aprendizagem do aprendiz S1

O aprendiz S2 teve conceito muito bom. Ele entregou o código bem organizado e

estruturado, mas as expressões booleanas podiam ser mais claras. O código entregue atendia todos os requisitos e não gerava nenhum erro nem warning na compilação.

O aprendiz S2 gastou 35 minutos para desenvolver a atividade, realizou 14 compilações, resultado de um número maior de utilizações do debug, três ao total em trechos relevantes do código.

Ao longo do desenvolvimento, S2 cometeu apenas erros de sintaxe e um erro de conversão de variáveis. Por ser apenas um erro gerado podemos supor que foi uma falta de atenção e não uma dificuldade.

A evolução do código fonte seguiu o padrão dos outros aprendizes, conforme figura 7.14.

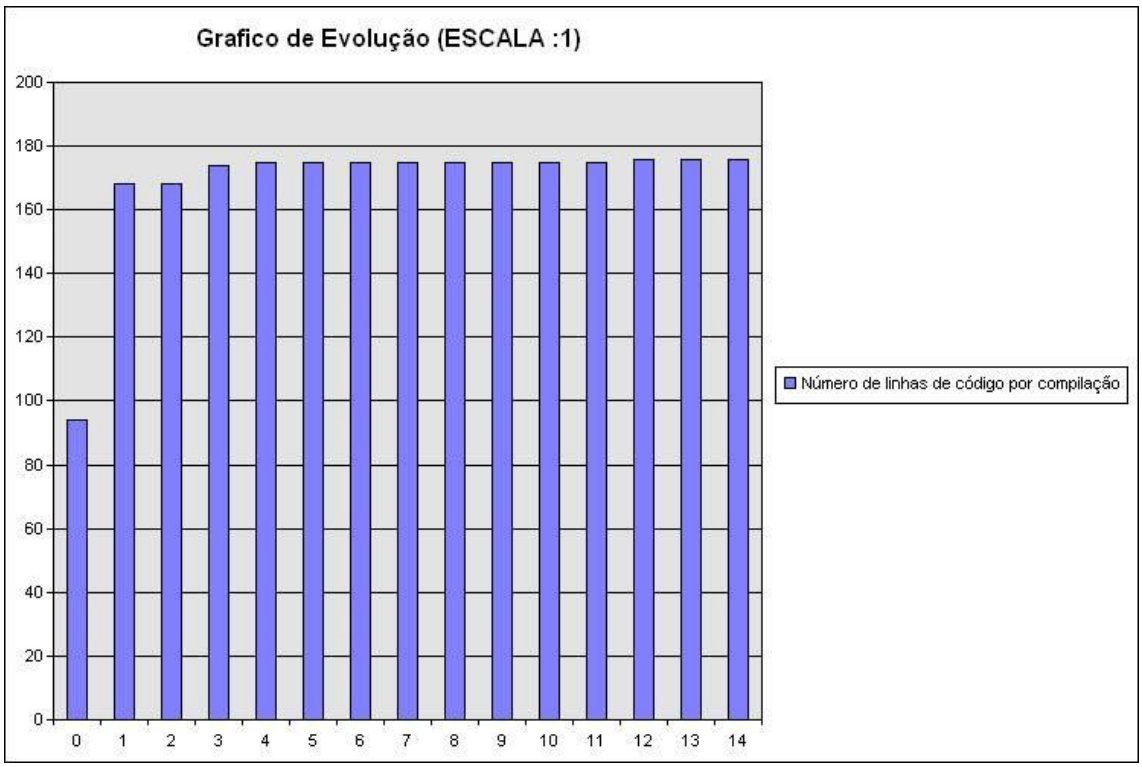


Figura 7.14: Gráfico da métrica de evolução do aprendiz S2.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, de que este aprendiz, apesar de não ter tido contato com orientação a objeto já possuía conhecimento em linguagens de programação.A tabela 7.2 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	35/15 = 2.33 (233%)
Quantidade de items avaliados com nota máxima em relação ao total possível	2/7 = 0.33 (33%)

Tabela 7.2: Parâmetros de aprendizagem do aprendiz S2

O aprendiz S3 teve conceito bom. Ele entregou o código limpo, bem organizado e estruturado. O código entregue atendia todos os requisitos e não gerava nenhum erro nem warning na compilação.

O aprendiz S3 gastou 112 minutos para desenvolver a atividade, realizou 41 compilações, um número bem superior aos outros aprendizes. Isto faz com que analisemos os erros de cada compilação, conforme figura 7.15.

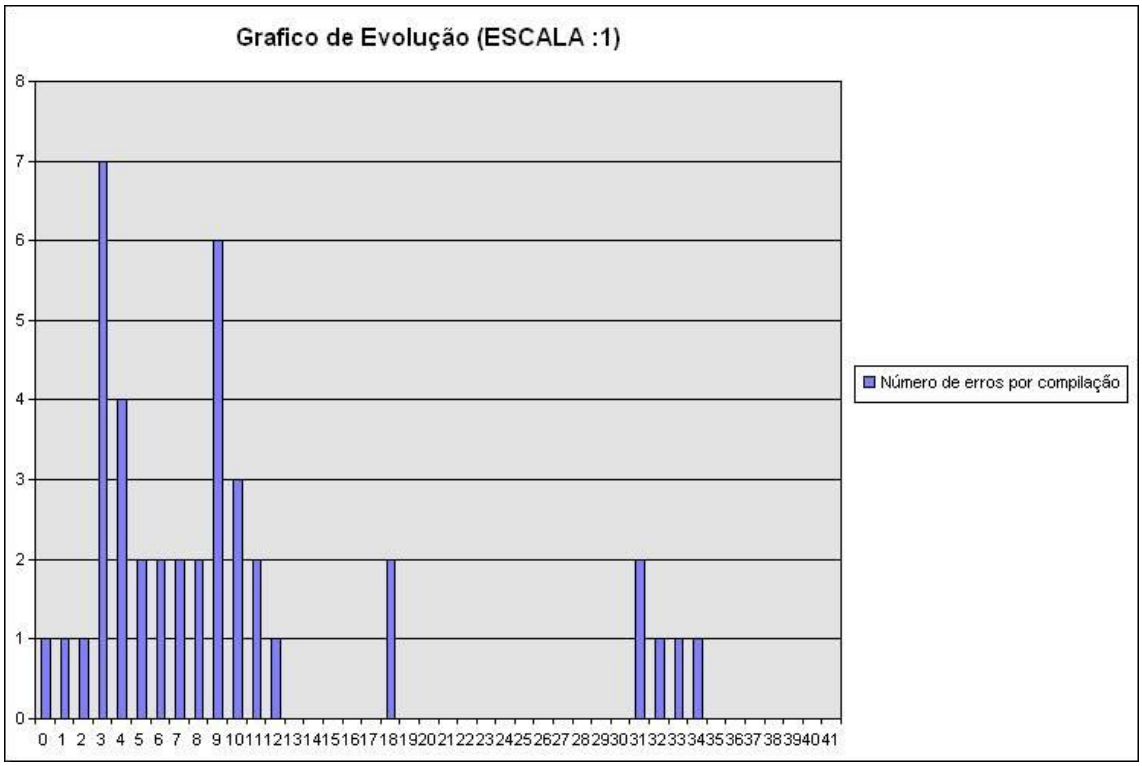


Figura 7.15: Gráfico de erros por compilação do aprendiz S2.

Uma concentração de erros é percebida no início do desenvolvimento, principalmente erros de sintaxe, o que é totalmente esperado por se tratar do primeiro contato com a linguagem. Mas houve alguns erros de conversão de variáveis também, indicando aqui uma dificuldade. A utilização do debug em uma linha de código onde havia um comando de conversão reforça esta dificuldade.

Outras duas utilizações do debug ocorreram, mas em trechos relevantes, como loops.

A evolução do código fonte seguiu o padrão dos outros aprendizes, conforme figura 7.16.

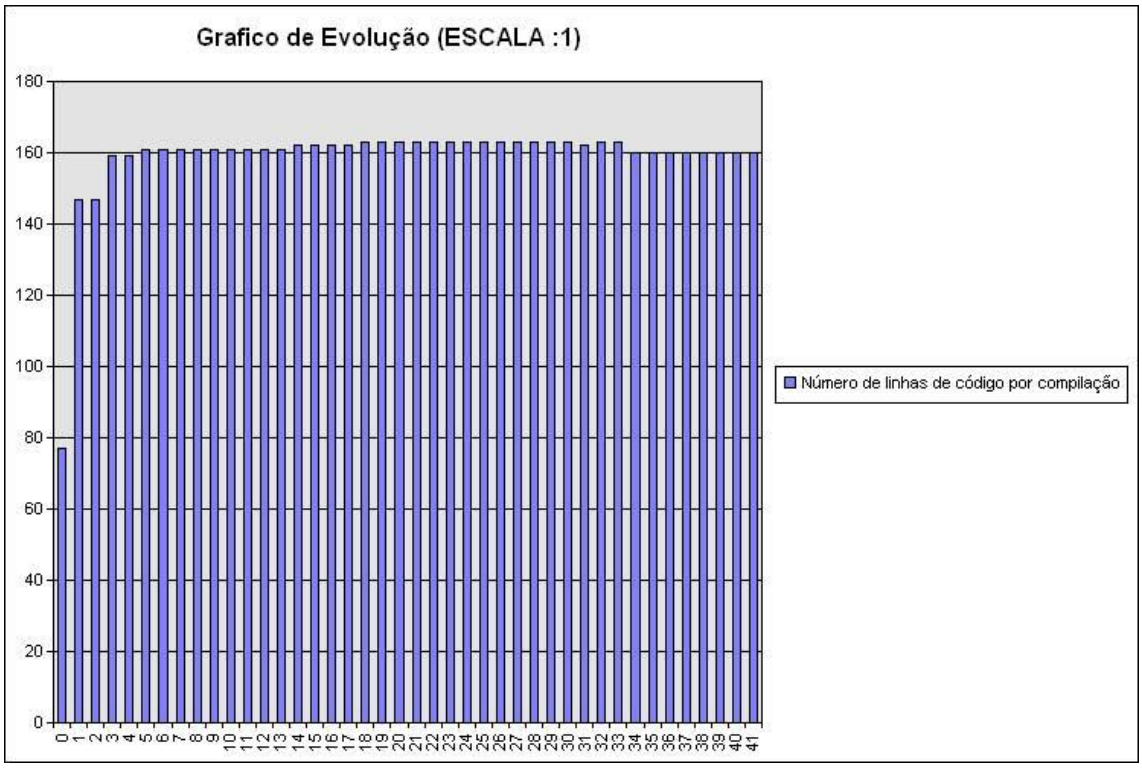


Figura 7.16: Gráfico da métrica de evolução do aprendiz S3.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, que o aprendiz não teve muitas dificuldades para resolução da atividade e teve um bom entendimento do conteúdo envolvido. A tabela 7.3 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	112/15 = 7.47 (747%)
Quantidade de items avaliados com nota máxima em relação ao total possível	2/7 = 0.33 (33%)

Tabela 7.3: Parâmetros de aprendizagem do aprendiz S3

Com estes exemplos, conclui-se que um aprendiz (S3) teve mais dificuldades que S1 e S2, mas ele teve uma boa aprendizagem, reforçada pela impressão do professor.

Nesta atividade todos os aprendizes seguiram o mesmo padrão do gráfico da métrica de evolução, nenhum aprendiz, como na primeira turma, fez comentários no código e o tempo médio gasto no desenvolvimento foi de 78 minutos.

7.2.2 A segunda atividade

Primeiramente para avaliação foi necessário definir os pesos para os ítens que seriam avaliados. Um quadro foi gerado conforme figura 7.17 que foi utilizado para avaliar a segunda atividade realizada pelos aprendizes.

	Peso	Nota Padrão	Nota Aplicada		
Análise do código entregue					
Qualidade dos comentários	**1**	**5**			
Explicação da lógica de um trecho de código	2	1			
Cabeçálhos de funções contém explicações dos parametros	2	1			
Descrição de caso de teste	1	1			
Organização do código	**1**	**5**			
Alinhamento do código	1	1			
Cada função executa uma única tarefa	1	1			
Utilização de "nomes fortes" para variáveis, funções e classes	1	1			
Loops com código limpo	1	1			
Expressões booleanas claras e parentizadas	1	1			
Compilação	**2**	**5**			
Compilação sem gerar nenhum erro ou warning	1	5			
Análise do processo de aprendizagem				**Valor Esperado**	**Valor obtido**
Número de utilizações do debug pelo especialista com um erro de 20%	**2**	**5**		Entre 1 e 4	
Número de compilações realizadas pelo especialista com um erro de 20%	**1**	**5**		Entre 14 e 22	
Erros cometidos pelo especialista quando desenvolveu a atividade	**2**	**5**		Erros de sintaxe; Comparação com valor null;	
Tempo gasto pelo especialista com um erro de 20%	**1**	**5**		Entre 25 a 39	
Evolução de parâmetros, como número de funções, classes e variáveis	**1**	**5**		Crescente	
Possível utilização do recurso copy e paste	**1**	**5**		Entre 2 e 4	

Figura 7.17: Quadro para avaliação da segunda atividade

No quadro já consta os valores que o especialista obteve resolvendo a atividade, para comparar com os dados do aprendiz. Os mesmos pesos dobrados dos ítens da primeira atividade são mantidos, pois estes ainda têm uma maior importância para o conceito.

Nesta atividade obtive-se um aproveitamento melhor do que a primeira:

- 85% dos aprendizes obtiveram conceito bom e

- 15% dos aprendizes obtiveram conceito muito bom.

Estes índices mostram uma evolução, o que já era esperado, pois os aprendizes com a primeira atividade tiveram um primeiro contato com o ambiente e com a linguagem. E como o esforço gasto é diretamente proporcional a capacidade de programar, estes índices tendem a melhorar cada vez mais.

Os mesmos três aprendizes da primeira atividade foram acompanhados e assim é possível observar suas evoluções.

O aprendiz S1 teve novamente o conceito bom, mas nesta atividade ele teve um pouco mais de dificuldade, chegou a discutir a atividade com o aprendiz do lado. Entregou a atividade conforme os requisitos, manteve o código limpo, organizado e bem estruturado, sem nenhum erro ou warning. Utilizou duas vezes o debug em trechos relevantes indicando saber usá-lo.

O aprendiz realizou pouquíssimas compilações ao longo dos 99 minutos que gastou para realizar a atividade, um total de quatro. Isto se deve ao fato do aprendiz primeiramente discutir a atividade com outro e somente após ter certeza do que estava fazendo é que compilava o programa. Isto demonstra uma certa insegurança na hora da codificação, apesar que quando questionado pelo professor sobre o conteúdo da atividade, teve convicção em suas respostas que estavam corretas.

O aprendiz novamente cometeu erros de inicialização de variáveis. Aqui podemos afirmar que este subtópico não foi totalmente compreendido, já que está presente nas duas primeiras atividades. Cabe ao professor, neste momento, passar um atividade de reforço ou explicar o subtópico novamente, o que ele julgar mais adequado.

O aprendiz não utilizou o recurso copy e paste e a evolução do código fonte continuou com o mesmo padrão da primeira atividade.

Pelo tempo gasto, número de compilações muito baixas e evolução do código fonte, pelo sistema foi capaz de descobrir o aprendiz com que S1 discutia e conseqüentemente fazia a atividade.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor.A tabela 7.4 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	99/32 = 3.09 (309%)
Quantidade de items avaliados com nota máxima em relação ao total possível	3/9 = 0.33 (33%)

Tabela 7.4: Parâmetros de aprendizagem do aprendiz S3

O aprendiz S2 manteve o conceito muito bom. Entregou um código mais limpo do que na primeira atividade, utilizou expressões booleanas mais claras. A atividade entregue atendia todos os requisitos propostos.

Realizou poucas compilações, um número inferior ao especialista, mas dentro da faixa de erro de 20%. Realizou debug em trechos relevantes. Utilizou o recurso copy e paste, copiando um trecho de código que era semelhante em duas funções. O gráfico de evolução do código fonte seguiu o mesmo padrão da primeira atividade.

Um fato interessante foi a ocorrência de erros, tanto nos subtópicos envolvidos na atividade anterior como nesta. O aprendiz cometeu poucos erros de conversão e declaração de variáveis e chamada de métodos, conforme figura 7.18.

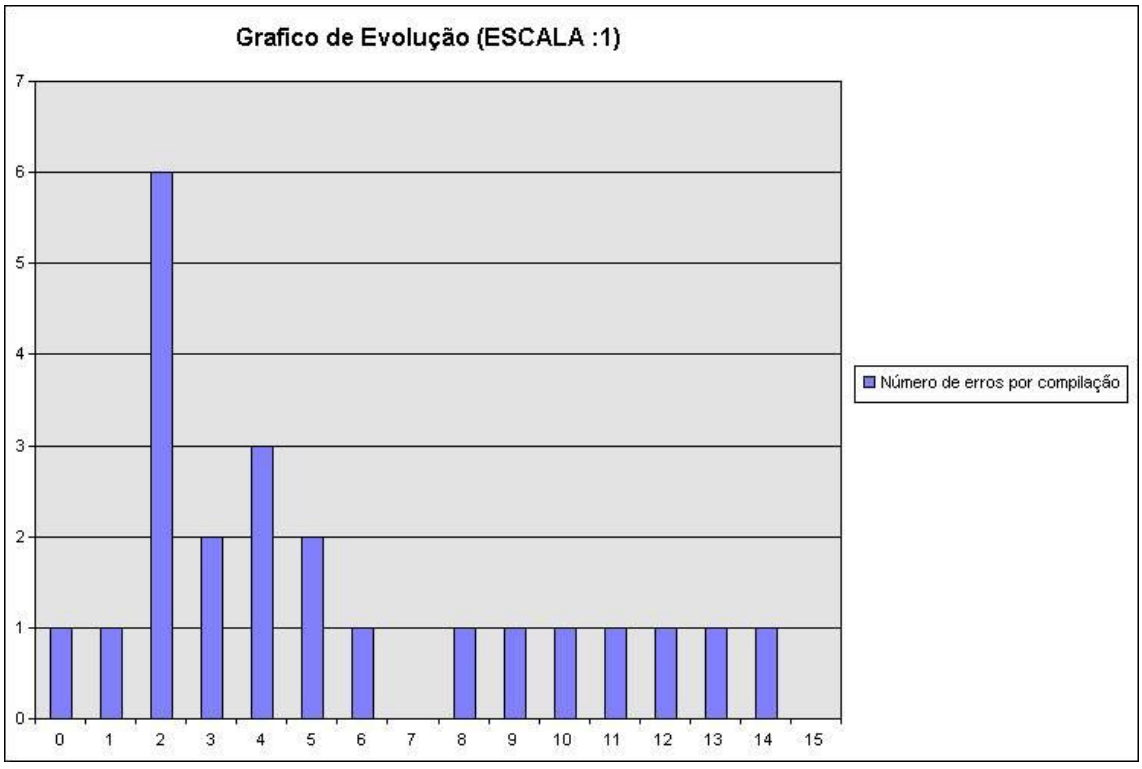


Figura 7.18: Erros por compilação na segunda atividade do aprendiz S2

No pico da figura 7.18 houve apenas erros de sintaxe. Em cada compilação há poucos erros, geralmente um. No detalhamento o erro não se repetia por duas compilações seguidas. Isto pode indicar uma falta de atenção do aprendiz e não uma dificuldade.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, já que este aprendiz tem uma facilidade maior de aprendizagem em relação ao resto da turma. A tabela 7.5 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	85/32 = 2.66 (266%)
Quantidade de items avaliados com nota máxima em relação ao total possível	6/9 = 0.66 (66%)

Tabela 7.5: Parâmetros de aprendizagem do aprendiz S3

O aprendiz S3 manteve o conceito bom, entregou o código limpo, organizado e estruturado conforme os requisitos e utilizou o recurso copy e paste em trechos semelhantes em funções distintas.

Novamente nesta atividade o aprendiz cometeu erros de conversão de variáveis e mais erros de declaração de variáveis, caindo na mesma situação do aprendiz S1, ou seja, seria necessário uma atividade extra ou nova explicação, o que o professor julgar mais adequado.

O gráfico de evolução de S3 não seguiu o padrão de nenhum outro aprendiz, nem mesmo o padrão por ele realizado na primeira atividade. Observa-se a evolução do código fonte desta atividade na figura 7.19.

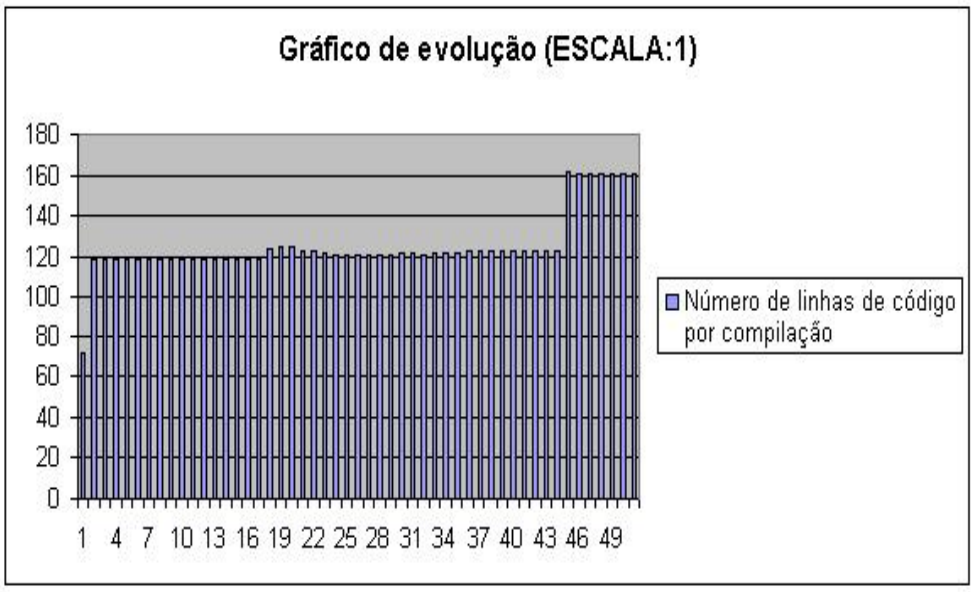


Figura 7.19: Evolução do código fonte na segunda atividade do aprendiz S3

O professor observou que o aprendiz ficava analisando o código, fazia uma alteração e compilava. Por isso o número elevado de compilações e apenas uma utilização de debug

em um trecho não relevante.

O gráfico da figura 7.19 comprova que o aprendiz dividou o problema em subproblemas e foi resolvendo um a um. O gráfico também indica uma certa dificuldade em resolver um subproblema, tanto que o tamanho do código permaneceu constante em boa parte do tempo.

Comparando o tempo gasto por este aprendiz com o restante, ele teve um tempo de desenvolvimento 15% inferior. Isto indica que para este caso dividir o problema em subproblemas menores, teoricamente, é uma boa solução.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, já que o aprendiz soube dividir os subproblemas da atividade e resolvê-los, uns com mais dificuldades outros com menos. A tabela 7.6 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	125/32 = 3.91 (391%)
Quantidade de items avaliados com nota máxima em relação ao total possível	5/9 = 0.56 (56%)

Tabela 7.6: Parâmetros de aprendizagem do aprendiz S3

7.2.2.1 A mudança de estratégia

Com base nas perguntas feitas durante a apresentação do conteúdo e durante o desenvolvimento da atividade pela primeira turma desta sessão, foi constatada a necessidade de dar mais ênfase a um determinado subtópico. Duas estratégias de apresentação foram elaboradas, uma apresentada para a segunda turma e outra para a terceira.

As estratégias diferiam quanto ao conteúdo de slides do powerpoint e exemplos.

Obtive-se duas respostas claras:

1. Número inferior de perguntas durante a apresentação e desenvolvimento do código referente ao subtópico e

2. Diminuição do tempo gasto para resolução da atividade, conforme tabela 7.7.

Turma	Tempo médio gasto (min)	Percentual de diminuição
Primeira	135	-
Segunda	92	32%
Terceira	100	25%

Tabela 7.7: Comparação de tempos gastos para resolução da segunda atividade

Por estes índices percebe-se que a segunda estratégia seria a mais adequada, teoricamente, para as outras turmas.

7.2.3 A terceira atividade

Para a avaliação da terceira atividade foi necessário definir os pesos para os ítens que seriam avaliados. Foi gerado, então, um quadro conforme figura 7.20 que foi utilizado para avaliar a terceira atividade realizada pelos aprendizes.

	Peso	Nota Padrão	Nota Aplicada		
Análise do código entregue					
Qualidade dos comentários	1	5			
Explicação da lógica de um trecho de código		2			
Cabeçalhos de funções contém explicações dos parametros		2			
Descrição de caso de teste		1			
Organização do código	1	5			
Alinhamento do código		1			
Cada função executa uma única tarefa		1			
Utilização de "nomes fortes" para variáveis, funções e classes		1			
Loops com código limpo		1			
Expressões booleanas claras e parentizadas		1			
Compilação	2	5			
Compilação sem gerar nenhum erro ou warning		5			
Análise do processo de aprendizagem				Valor Esperado	Valor obtido
Número de utilizações do debug pelo especialista com um erro de 20%	2	5		0	
Número de compilações realizadas pelo especialista com um erro de 20%	1	5		Entre 5 e 9	
Erros cometidos pelo especialista quando desenvolveu a atividade	2	5		Erros de sintaxe	
Tempo gasto pelo especialista com um erro de 20%	1	5		Entre 20 a 30	
Evolução de parâmetros, como número de funções, classes e variáveis	1	5		Crescente	
Possível utilização do recurso copy e paste	1	5		Entre 6 e 10	

Figura 7.20: Quadro para avaliação da terceira atividade

No quadro já constam os valores que o especialista obteve resolvendo a atividade, para

comparar com os dados do aprendiz. Continua-se com os mesmos pesos dobrados dos ítens da primeira atividade.

Com os resultados da análise das métricas coletadas desta atividade, já é possível observar uma evolução, esperada, dos aprendizes, já que todos tiveram conceito muito bom.

Um fato muito interessante é que nesta atividade, todos os aprendizes realizaram a atividade sozinhos, sem, em nenhum momento, fazer perguntas ao professor durante o desenvolvimento.

Todos entregaram códigos claros, organizados e estruturados, mesmo os que nas primeiras atividades não o fizeram assim, mas em nenhum momento, houve comentários no código.

A totalidade dos aprendizes desenvolveram todo o código primeiro e depois foram ajustando, realizando poucas compilações durante o desenvolvimento para teste de sintaxe e todos utilizaram o recurso copy e paste dentro do intervalo de erro do valor do especialista e em trechos semelhantes entre funções.

O aprendiz S1 realizou dez compilações e gastou 67 minutos para desenvolver a atividade, inferior à média da turma.

Além de alguns erros de sintaxe, o aprendiz cometeu poucos erros sobre classes e métodos abstratos, mas sozinho conseguiu arrumá-los. Isto pode indicar um pequeno equívoco na hora de passar da teoria para a prática e não uma dificuldade ou dúvida do aprendiz.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, já que sozinho o aprendiz realizou a atividade. A tabela 7.8 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	$67/25 = 2.68$ (268%)
Quantidade de items avaliados com nota máxima em relação ao total possível	$5/9 = 0.56$ (56%)

Tabela 7.8: Parâmetros de aprendizagem do aprendiz S1

O aprendiz S2 foi o que compilou a atividade mais vezes, 31 ao todo. Mas as com-

pilações se concentram no final do desenvolvimento e sem erros, isto indica, como visto pelo professor na sala, que o aprendiz esta testando o código. Apesar da atividade não ter nenhum caso de teste, o aprendiz fez alguns e testou.

Foram gastos 30 minutos pelo aprendiz para desenvolver a atividade, quase já dentro da faixa de erro do valor do especialista, e ele cometeu poucos erros de nível de acesso e erros de classes e métodos abstratos, mas como S1 ele os corrigiu sozinho.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, pois além de entregar a atividade conforme os requisitos ainda realizou casos de testes. A tabela 7.9 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	30/25 = 1.2 (120%)
Quantidade de items avaliados com nota máxima em relação ao total possível	5/9 = 0.56 (56%)

Tabela 7.9: Parâmetros de aprendizagem do aprendiz S2

O aprendiz S3 gastou 59 minutos para desenvolver a atividade, cometeu poucos erros de nível de acesso e de sobrecarga de métodos. Mas como os outros aprendizes, ele os corrigiu sozinho.

Este aprendiz foi o que menos compilou a atividade, cinco vezes, valor inferior ao do especialista. As compilações estavam bem espaçadas, em média 10 minutos entre elas. Isto indica, como observado, que o aprendiz só compilava o programa quando tinha certeza do que estava fazendo, apesar dos poucos erros cometidos.

O conceito atribuído pelo sistema comprova a realidade vista pelo professor, de um aprendizado muito bom. A tabela 7.10 mostra alguns parâmetros deste aprendiz que foram utilizados a fim de comparação para analisar a evolução da aprendizagem.

Tempo gasto em relação ao especialista	59/25 = 2.36 (236%)
Quantidade de items avaliados com nota máxima em relação ao total possível	6/9 = 0.66 (66%)

Tabela 7.10: Parâmetros de aprendizagem do aprendiz S3

A avaliação desta atividade, já está mais simples, pois os aprendizes já estão familiarizados com o ambiente e com a sintaxe da linguagem. Provavelmente se tivesse uma quarta atividade, as métricas já estariam muito próximas das do especialista, já que nesta

tivemos aprendizes que conseguiram igualar algumas.

7.2.4 Os três aprendizes

Apenas pelas tabelas apresentadas ao final de cada avaliação dos três aprendizes já é possível observar uma evolução. A tabela 7.11 junta estes valores para que se possa ter uma idéia melhor desta evolução.

	Primeira Atividade	Segunda Atividade	Terceira Atividade
Aprendiz S1			
Tempo gasto/Especialista	433%	309%	268%
Nota 5/total de conceitos	43%	33%	56%
Conceito	Bom	Bom	Muito Bom
Aprendiz S2			
Tempo gasto/Especialista	233%	266%	120%
Nota 5/total de conceitos	33%	66%	56%
Conceito	Muito Bom	Muito Bom	Muito Bom
Aprendiz S3			
Tempo gasto/Especialista	747%	391%	236%
Nota 5/total de conceitos	33%	56%	66%
Conceito	Bom	Bom	Muito Bom

Tabela 7.11: Parâmetros de aprendizagem dos aprendizes S1, S2 e S3

Pela evolução apresentada seria coerente atribuir aos aprendizes S1 e S3 o conceito final do curso muito bom e ao aprendiz S2 o conceito louvor.

Não se pode afirmar que estes aprendizes têm um excelente conhecimento da linguagem de programação C#, mesmo porque o curso não abrange todos os recursos disponíveis, como por exemplo acesso a banco de dados, por outro lado pode-se afirmar que estes aprendizes possuem uma base lhes que permitem aprender outros recursos, já que tudo em C# é baseado em objetos.

7.2.5 As turmas

Assim como foi observada a evolução dos três aprendizes, é possível observar a evolução das turmas. A tabela 7.12 mostra os mesmos parâmetros observados nos aprendizes só que, na média, para as três turmas.

	Primeira Atividade	Segunda Atividade	Terceira Atividade
Primeira turma Tempo gasto/ Especialista	100/15 = 6.67 (667%)	138/32 = 4.31 (431%)	85/25 = 3.4 (340%)
Nota 5/ total de conceitos	(2.4)/7 = 0.34 (34%)	(3.7)/9 = 0.41 (41%)	(5.5)/9 = 0.61 (61%)
Segunda turma Tempo gasto/ Especialista	82/15 = 5.47 (547%)	92/32 = 2.88 (288%)	59/25 = 2.36 (236%)
Nota 5/ total de conceitos	2/7 = 0.29 (29%)	(3.7)/9 = 0.41 (41%)	5/9 = 0.56 (56%)
Terceira turma Tempo gasto Especialista	92/15 = 6.13 (613%)	100/32 = 3.13 (313%)	65/25 = 2.6 (260%)
Nota 5/ total de conceitos	3/7 = 0.43 (43%)	5/9 = 0.56 (56%)	(5)/9 = 0.56 (56%)

Tabela 7.12: Parâmetros de aprendizagem das três turmas

Percebe-se que todas as turmas evoluíram, umas mais que outras. Isto pode ser justificado pela presença de aprendizes com maiores dificuldades em entender a matéria e/ou realizar a atividade. Mas no geral todos evoluíram.

Os aprendizes que não tiveram um bom conceito na primeira atividade, tiveram o mesmo conceito na terceira que os aprendizes que foram melhor na primeira.

7.3 Os resultados

Não podemos afirmar que o modelo de acompanhamento apresentou resultados melhores do que o modelo tradicional, porque não há dados para fazer esta comparação, que é muito complexa, visto que um aprendiz pode ter resultados melhores em um determinado modelo, mas isso não implica que outros aprendizes também terão, mesmo assim visto a evolução positiva dos aprendizes, o modelo de acompanhamento é um modelo que pode ser considerado como uma boa alternativa no ensino de linguagens de programação.

Destaca-se alguns fatores que reforçam a avaliação do processo de aprendizagem e não somente o resultado final:

- A identificação de dificuldades é um ponto chave, por dois motivos:

    1. Muitos aprendizes não manifestam suas dificuldades em aula, seja por vergonha, timidez ou qualquer outro motivo e o professor só tem condições de detectar essa dificuldade em uma prova, o que pode ser tarde. Quanto mais cedo uma dificuldade é detectada mais tempo o professor terá para resolvê-la.
    2. O professor através dessas dificuldades pode rever a estratégia de como o tópico correspondente foi abordado, melhorando assim o curso de programação, apresentando o tópico de uma maneira mais fácil de ser entendido.

- Aprendizes com uma maior facilidade em lógica e programação muitas vezes podem se sentir desmotivados quando se deparam apenas com atividades de fácil resolução. O professor pode, a partir do perfil do aprendiz, teoricamente, propor atividades com diferentes graus de dificuldades.

- A identificação de aprendizes que copiam a atividade de um colega é outro ponto importante. Em curso introdutório de programação conceitos básicos são apresentados e a não assimilação destes, com certeza, influencia de forma negativa cursos intermediários e avançados de programação. Este aprendiz que copia a atividade é o maior prejudicado e cabe ao professor orientá-lo.

- Verificar que os aprendizes utilizam todos os recursos disponíveis, como debug e help, é extremamente importante. Estes recursos auxiliam, de forma direta, o aprendiz a resolver um problema.

Com o estudo de caso obteve-se alguns resultados interessantes:

- A quase totalidade dos aprendizes desenvolvem todo o código fonte e depois vão adaptando de forma intuitiva até atingir o resultado esperado. Isto pode ser um agravante em atividades mais complexas e com certeza pode ser um fator desestimulante, já que é uma dificuldade em transformar o problema em algo computável e não no conteúdo em si.

- Foi possível determinar com precisão, por exemplo, que tópicos relacionados a variáveis apresentam uma dificuldade de entendimento por grande parte dos aprendizes.

Por ser um conceito básico e fundamental uma nova estratégia de abordagem poderia ser considerada.

- Obter dados para poder comparar diferentes abordagens é extremamente útil para o professor, desta maneira ele pode escolher qual tem um impacto melhor na aprendizagem. O exemplo que tivemos, apesar de apenas compararmos o tempo total gasto na resolução do problema, foi expressivo. O tópico onde a abordagem foi diferenciada entre as turmas não era chave para a resolução do problema, mas mesmo assim o impacto pode ser percebido, não só pelo tempo, mas pelas dúvidas manifestadas pelos aprendizes.

- A comparação entre a avaliação realizada com o sistema e as impressões do professor foi positiva. Em sua maioria esta comparação coincidiu, o que permite o sistema ser utilizado em atividades que necessitem de mais de uma sessão de estudo e permita ao aprendiz utilizar o computador em casa ou qualquer lugar que possua acesso a internet. Houve um exemplo em que esta comparação não coincidiu, foi o caso do aprendiz que realizou mais do que a atividade pedia, gerando assim uma avaliação ruim. Neste exemplo a experiência do professor é fundamental para uma avaliação correta utilizando os dados do sistema e o código final entregue. Esta experiência pode gerar novas métricas e assim o sistema teria a capacidade de avaliar casos como este.

- A comparação de desempenho entre aprendizes é muito útil para determinar quais se sobressaem e quais têm mais dificuldades. Esta comparação só é possível porque a avaliação é feita de forma padronizada para todos os aprendizes. Isto implica que a pré-avaliação do sistema implementado auxilia o professor a direcionar os esforços nos aprendizes com maior dificuldade.

- Foi possível identificar aprendizes que copiaram a atividade.

- Também foi possível identificar aprendizes preocupados em testar o código e utilizar os recursos disponíveis para isso.

- Nenhum aprendiz realizou comentários no código, mesmo com a orientação do professor. Em um problema mais complexo que exija mais tempo para resolução isto é um agravante.

- Aprendizes não gostam de utilizar o help, apesar dele ser completo, seja por causa do idioma ou principalmente, pelo fato de existir um facilitador em tempo integral, mesmo que este não explique como funciona um comando, por exemplo, o aprendiz prefere a ”tentativa e erro”a pesquisar no help.

Capítulo 8

Conclusões e trabalhos futuros

Nesta dissertação criamos um sistema de coleta e análise de métricas que suporta o processo de avaliação de linguagens de programação. Mostramos, primeiramente, como elaboramos um modelo de acompanhamento do processo de aprendizagem que comporte este sistema.

Com este modelo, implementamos um sistema de coleta de métricas durante a fase de resolução de problema para podermos implementar a fase de validação e avaliação final, já que a avaliação formativa se baseia em todo o processo de aprendizagem.

O sistema implementado de coleta e análise de métricas, entre outras funções, provê informações que auxiliam o professor a identificar as principais dificuldades dos aprendizes. Com o estudo de caso realizado fica evidente a importância desta identificação, pois o professor pode adaptar a abordagem utilizada a fim de diminuí-las e conseqüentemente melhorar o desempenho do curso.

Com o uso dessas métricas foi possível padronizar a avaliação do processo de aprendizagem tornando viável a comparação do desempenho entre alunos, grupos e turmas. Esta comparação é extremamente importante para medirmos o impacto que a metodologia adotada está causando.

Apesar de não possuirmos dados de outras metodologias, o modelo de acompanhamento do processo de aprendizagem de linguagens de programação se mostrou uma boa alternativa para o ensino de linguagens de programação, pois estimula o aprendiz a raciocinar, isto é importantíssimo em programação, pois os problemas são distintos e sempre

possuem mais de uma solução.

Como contribuição desta dissertação temos um modelo de acompanhamento do processo de aprendizagem de linguagens de programação e um sistema de coleta de métricas que suporta a avaliação formativa, de forma padronizada.

8.1 Trabalhos publicados

Ao longo desta pesquisa foram publicados dois artigos listados a seguir:

1. Coleta e análise de métricas no processo de aprendizagem de programação (PêGAS; YANO, 2004a), publicado no XII Workshop de Educação em Computação (WEI) em agosto de 2004. Neste artigo foi dado enfoque sobre o processo de coleta de métricas durante o desenvolvimento de uma atividade diretamente no ambiente de desenvolvimento e o processo adotado para encontrar estas métricas.

2. O uso de data warehousing no processo de aprendizagem de programação (PêGAS; YANO, 2004b), publicado no XV Simpósio Brasileiro de Informática na Educação (SBIE) em novembro de 2004. Neste artigo o enfoque dado foi o uso de Data Warehouse sobre os dados coletados durante o desenvolvimento de uma atividade a fim de encontrar padrões de aprendizagem.

8.2 Trabalhos futuros

Diante dos estudos realizados e dos resultados obtidos deixamos como continuidade os seguintes trabalhos futuros:

- Estender o conceito de avaliar o processo de aprendizagem de programação, através da coleta de métricas, para outras metodologias, como por exemplo, a tradicional.

- Implementar a o modelo de acompanhamento em um curso regular de introdução de programação, afim de obter dados para a comparação com o modelo tradicional.

- Coletar métricas em mais turmas para obtermos uma quantidade mínima de dados para o uso de Data Warehouse, permitindo assim o descobrimento de novas métricas

- Estender o conceito de coleta de métricas diretamente do ambiente de desenvolvimento para uma fábrica de software, a fim de medir o desempenho dos desenvolvedores.

Referências Bibliográficas

ADAM, A.; LAURENT, J. A system to debug student programs. **Artificial Intelligence**, p. 75–122, 1980.

ALBANESE, M. Problem-based learning: why curricula are likely to show little effect on knowledge and clinical skills. **Med Educ**, v. 34, p. 729–738, 2000.

BAFFES, P. T. **Automatic student modeling and bug library construction using theory refinement**. Tese (Doutorado) — University of Texas, December 1994.

BINDER, F. V.; DIRENE, A. I. Conceitos e ferramentas para apoiar o ensino de lógica de programação imperativa. **Anais do X Simpósio Brasileiro de Informática na Educação**, 1999.

BOULAY, B. du; O´SHEA, T. Teaching novices programming. **Coombs, M. J. e Alty, J. L. Computing Skills and the User Interface**, p. 147–200, 1981.

BOULAY, B. du; SOTHCOOT, C. Computers teaching programming: an introdutory survey of the field. **LAWLER, R. W.; YAZDANI, M. Artificial intelligence and education: learning environment and tutoring systems**, v. 1, p. 345–372, 1987.

CORTELAZZO, I.; OLIVEIRA, V. B. Repensando a educação em função de mudanças sociais. **Oliveira, V. B.(org.):Informática em Psicopedagogia**, p. 111–130, 1996.

CRAW, S.; SLEEMAN, D. Automating the refinement of knowledge- based systems. **Proceedings of ECAI-90**, p. 167–172, 1990.

DIRENE, A. I.; GUEDES, A. L. P.; SANTOS, G. Autoria e interpretação tutorial de soluções alternativas para promover o ensino de programação de computadores. **Anais do XIV Simpósio Brasileiro de Informática na Educação**, v. 1, p. 623–632, 2003.

ELMASRI, R. **Fundamentals of Database Systems**. 4. ed. [S.l.]: Addison Wesley, 2003. ISBN 0321122267.

GOLDSTEIN, I. P. Summary of mycroft: a system for understanding simple picture programs. **Artificial Intelligence**, p. 249–288, 1975.

GREY, D. J. **Student Profiling and Course Management in Assignment Manager**. [S.l.], 2004. Disponível em: <http://www2.dcs.hull.ac.uk/people/cssdjg/Projects/AM/>. Acesso em: 22 abril 2004.

HASEMER, T. **An empirically-based debugging system for novice programmers.** Human Cognition Research Laboratory, 1983.

HAYDT, R. C. **Avaliação do processo ensino-aprendizagem**. São Paulo: Ática, 1995.

JúNIOR, M. C. R. **Como Ensinar Programação?** 2002. ULBRA.

JOHNSON, W. L. Intention-based diagnosis of novice programming errors. **Artificial Intelligence**, 1986.

KANTARDZIC, M. **Data Mining: Concepts, Models, Methods, and Algorithms**. 1a.. ed. 111 River Street, Hoboken, USA: IEEE, 2003. ISBN 0471228524.

KIMBALL, R. **The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling**. 2. ed. [S.l.]: Wiley, 2002. ISBN 0471200247.

LITTO, F. M. Repensando a educação em função de mudanças sociais. **Oliveira, V. B.(org.):Informática em Psicopedagogia**, p. 85–110, 1996.

LUKEY, F. J. Understanding and debugging programs. **International Journal Man-Machine Studies**, p. 189–202, 1980.

MARTIN, J. M. **Is Turing a better language for teaching programming than Pascal?** Dissertação (Mestrado) — University of Stirling, Scotland, 1996.

PAINE, C.; THOMAS, P. **AESOP-An Electronic Student Observatory Project.** [S.l.], 1999. Disponível em: <http://mcs.open.ac.uk/aesop/>. Acesso em: 27 abril 2004.

PERRENOUD, P. **Avaliação: da excelência à regulação da aprendizagem - entre duas lógicas**. 1999. Artes Médicas Sul.

PêGAS, D. S.; YANO, E. T. Coleta e análise de métricas no processo de aprendizagem de programação. **Anais do SBC 2004**, v. 1, p. 896–907, Agosto 2004.

PêGAS, D. S.; YANO, E. T. O uso de data warehousing no processo de aprendizagem de programação. **Anais do XV Simpósio Brasileiro de Informática na Educação**, v. 1, p. 568–577, Novembro 2004.

PROULX, V. Programming patterns and design patterns in the introductory computer science course. **Proc. of the 31st SIGCSE**, Auxtin, TX, USA, p. 7–12, 2000.

SHAPIRO, E. Y. **Algorithmic Program Debugging**. [S.l.]: MIT Press, 1983.

THOMAS, P.; PAINE, C. **Monitoring distance education students' practical programming activities**. [S.l.], 2002. Disponível em: <http://ifets.ieee.org/periodical/vol_3_2002/thomas.html/>. Acesso em: 27 abril 2004.

WENGER, E. **Artificial Intelligence and Tutoring Systems**. Los Altos, CA, USA: Morgan Kaufmann, 1987.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO N°	4. N° DE PÁGINAS
TM	14 de julho de 2005	CTA/ITA-IEC/TM-002/2005	110

5. TÍTULO E SUBTÍTULO:

Coleta e análise de métricas no processo de aprendizagem de linguagens de programação

6. AUTOR(ES):

Daniel dos Santos Pegas

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica. Divisão de Ciência da Computação – ITA/IEC

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Linguagens de programação, Modelagem cognitiva aplicada à educação, Observação Eletrônica

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Linguagens de programação; Aprendizagem; Modelagem (processos); Métricas (software); Métodos educacionais; Ensino; Educação; Engenharia de software

10. APRESENTAÇÃO: **X Nacional** **Internacional**

ITA, São José dos Campos, 2005, 109 páginas

11. RESUMO:

Uma tendência no processo de aprendizagem em geral é o professor cada vez mais exigir do aluno a capacidade de exploração e criação de novos conhecimentos e não, simplesmente, memorização do conteúdo de uma disciplina. Entretanto, essa nova abordagem requer novos instrumentos para acompanhamento e avaliação do processo de aprendizagem. Na abordagem tradicional de ensino de linguagem de programação, a avaliação se concentra no resultado, ou seja, provas e códigos prontos referentes a atividades propostas. Esta dissertação estende-se esta avaliação a todo o processo de aprendizagem, e para suportar isto, utiliza-se um modelo de acompanhamento e análise do processo de aprendizagem de linguagens de programação. Um estudo de caso foi elaborado e quatro cursos de introdução à linguagem de programação foram ministrados a fim de se obter dados que pudessem dizer que o uso do modelo de acompanhamento e análise do processo de aprendizagem é uma boa alternativa tanto para o ensino como para a avaliação da aprendizagem de linguagens de programação.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) CONFIDENCIAL** **( ) SECRETO**

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo