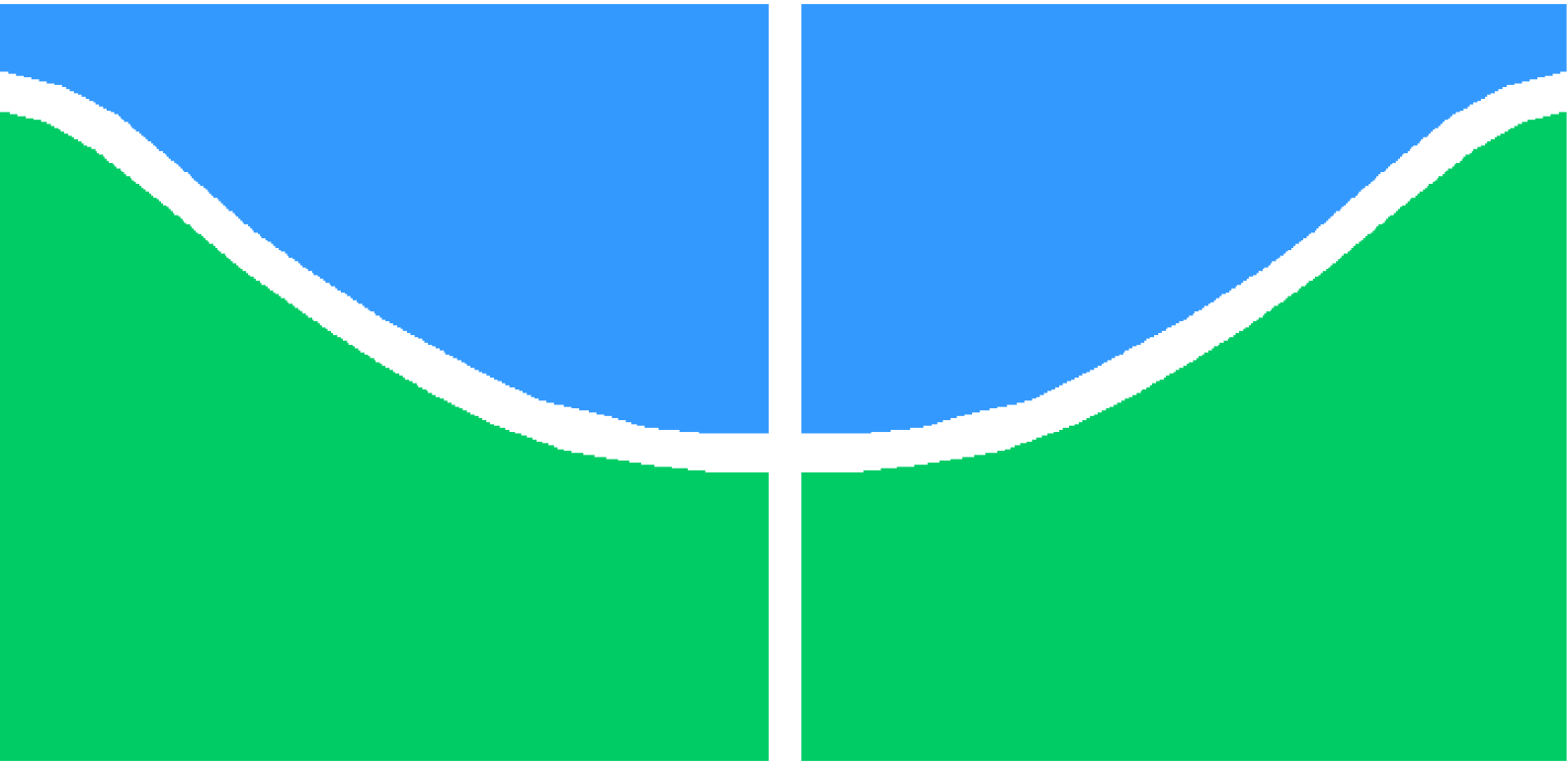

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Utilização de juízes eletrônicos e problemas oriundos da Maratona de Programação no ensino de programação da Faculdade UnB Gama: Um estudo de caso

Autor: Tiago Gomes Pereira

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2015

Tiago Gomes Pereira

Utilização de juízes eletrônicos e problemas oriundos da Maratona de Programação no ensino de programação da Faculdade UnB Gama: Um estudo de caso

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Brasília, DF

2015

Tiago Gomes Pereira
Utilização de juízes eletrônicos e problemas oriundos da Maratona de Programação no ensino de programação da Faculdade UnB Gama: Um estudo de caso/ Tiago Gomes Pereira. – Brasília, DF, 2015-
106 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Edson Alves da Costa Júnior

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2015.

1. Juiz Eletrônico. 2. Engenharia de Software. I. Prof. Dr. Edson Alves da Costa Júnior. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Utilização de juízes eletrônicos e problemas oriundos da Maratona de Programação no ensino de programação da Faculdade UnB Gama: Um estudo de caso

CDU 02:141:005.6

Tiago Gomes Pereira

Utilização de juízes eletrônicos e problemas oriundos da Maratona de Programação no ensino de programação da Faculdade UnB Gama: Um estudo de caso

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 06 de julho de 2013:

Prof. Dr. Edson Alves da Costa Júnior
Orientador

Prof. Dr. Maurício Serrano
Convidado 1

Prof. Dr. Augusto César de Mendonça Brasil
Convidado 2

Brasília, DF
2015

Dedico este trabalho aos futuros campeões que
diante da dificuldade encontraram um motivo para crescer.

Agradecimentos

Agradeço ao meu orientador, Edson Júnior, por ter me guiado neste trabalho e ensinar a olhar a vida com outros olhos. Agradeço aos meus pais, Rodrigo e Damaris, por me apoiarem e acompanharem durante toda a jornada sem nunca ter deixado que nada me faltasse e por sempre terem acreditado na minha capacidade Agradeço a minha companheira, Mariana Cristina, por ter me ajudado em tudo, desde das coisas pequenas até as coisas enormes. Sem você esse trabalho não teria sido possível. Obrigado pelo amor, carinho e compreensão.

Resumo

A computação é parte integrante de nossas vidas. Decisões importantes são tomada com base nas informações providas por softwares. Software de alta qualidade e eficiência é necessário para que estás decisões estejam bem fundamentadas. O Engenheiro de Software é o profissional responsável por garantir essa qualidade. Para que isso seja possível é necessário que ele conheça bem os fundamentos da ciências da computação. Mas ensinar programação é díficil. Manter os alunos motivados não é uma tarefa fácil e a taxa de evasão dos cursos de computação é alta. Para solucionar estes problemas foi implementada uma metodologia de avaliação baseada em competições de programação. Este trabalho tem como objetivo analisar a eficiência desta metodologia observando o impacto no desempenho dos alunos nas disciplinas de programação posteriores a ela.

Palavras-chaves: Maratona de programação. Análise de desempenho. Ensino. Programação. Engenharia de software.

Abstract

Computers are in our lives in a way that we can't live without it anymore. Important decisions are made based in information provided by software. The software which provides those information must be reliable and efficient, so the decisions made using them are well grounded. The Software Engineer are the professionals who are responsible to ensure this software quality. It is required from the software professional that his knowlodge in computer science are strong and his programming skill are sharp. But teaching programming is hard. Keep the students motiveted isn't a easy task and the drop rate in computer science courses in Brazil are high. In order to address this problem it was implemented in programming classes a teaching methodology inspired in programming competitions, like the ACM ICPC. This work objective is analyse the impact of this methodology in the students performances in future programming classes.

Key-words: Programming Marathon. Performance Analisys. Teaching. Programming. Software Engineering.

Lista de ilustrações

Lista de tabelas

Lista de Códigos

Lista de abreviaturas e siglas

AC	*Accepted*
WA	*Wrong Answer*
TLE	*Time Limit Exceeded*
RE	*Runtime Error*
CE	*Compilation Error*
PE	*Presentation Error*
ACM	*Association for Computing Machinery*
ICPC	*International Collegiate Programming Contest*
UnB	Universidade de Brasília
FGA	Faculdade do Gama
IME	Instituto Militar de Engenharia
USP	Universidade de São Paulo
IBM	*Internacional Business Machines*
PDF	*Portable File Document*
JSON	*Javascript Object Notation*
URI	Universidade Regional Integrada do Alto Uruguai e das Missões
UVa	*University of Valladolid*
ECMA	*European Computer Manufacturers Association*
ISO	*International Organization for Standardization*
TXT	Arquivo de texto
SR	Sem Rendimento
II	Inferior
MI	Médio Inferior

MM	Médio
MS	Médio Superior
SS	Superior
IRA	Índice de Rendimento Acadêmico
MVC	*Model View Controller*
HTML5	*Hypertext Markup Language*, versão 5
ID	Identificador
ISBN	*International Standard Book Number*
MIT	*Massachusetts Institute of Technology*
ISSN	*International Standard Serial Number*

Sumário

Introdução

Contexto

Os computadores estão presentes em grande parte dos locais e momentos do nosso dia a dia e, por conta disso, software se tornou profundamente inserido em praticamente todos os aspectos de nossas vidas (PRESSMAN, 2010).

A infraestrutura de um país é controlada por sistemas à base de computadores, e grande parte dos produtos elétricos incluem computadores e softwares de controle. Na indústria, tanto a manufatura quanto a distribuição dos produtos é completamente automatizada, assim como os sistemas financeiros. Praticamente todos os meios de entretenimento, música, jogos de computador, filmes e televisão, são completamente dependentes de software. Não é possível manter o mundo moderno sem software (SOMMERVILLE, 2011).

Isso torna necessário o conhecimento básico dos conceitos de computação para um profissional de praticamente qualquer área do conhecimento e do mercado de trabalho.

Problema

Por software ter se tornado indispensável para a infraestrutura, para a indústria, para a economia, para a ciência e para o entretenimento, o profissional de software capacitado se tornou essencial para o mercado (PRESSMAN, 2010).

A necessidade de profissionais mais capacitados na área da computação pode ser notada pela grande quantidade de iniciativas para popularização da programação. Como exemplo temos o Code.org[1], uma organização sem fins lucrativos que possui como objetivo popularizar o ensino de Ciência da Computação nas escolas e fazer com que ela seja parte do currículo escolar.

A computação está evoluindo e puxando os limites constantemente. É crescente a necessidade de softwares de qualidade e eficiência. Indivíduos, negócios e governos dependem de softwares para tomada de decisões e a realização de operações diárias (PRESSMAN, 2010).

Softwares de complexidade e dimensão crescentes se tornam necessários para auxiliar nessas tarefas. Ao análisar uma rede social por exemplo, devido ao grande número de usuários, realizar uma busca específica pode ser dispendioso tanto nas dimensões de

[1] <http://br.code.org/about>

tempo quanto na dimensão de recursos utilizados para tal. Um algoritmo mal escrito, pouco performático, pode demorar anos para devolver um resultado.

Apesar de grande parte dos problemas de velocidade (tempo de execução do programa) e eficiência (uso de memória, custo de hardware, gasto de energia/bateria) serem vistos como solucionáveis através da aquisição de mais unidades de hardware ou hardwares de melhor qualidade, é possível que, ao aprimorar o algoritmo da solução, o problema seja resolvido sem a necessidade de um hardware novo (SKIENA, 2008).

Portanto, programadores que possuam conhecimentos na área de projeto de algoritmos se tornam vitais para a sobrevivência da indústria e a evolução da tecnologia.

Porém, ensinar programação é complicado. O ensino de programação é uma grande dificuldade encontrada pelos professores de ensino superior de cursos de computação (FASSBINDER; PAULA; ARAUJO, 2012). Esta dificuldade contribui para o alto índice de evasão dos alunos de cursos de computação (INEP, 2007).

Os motivos para a alta evasão são diversos. A falta de conhecimento do aluno sobre os fundamentos básicos do curso, a dificuldade em perceber a lógica para resolver um problema, falta de dedicação do aluno, falta de interesse na linguagem de programação adotada ou a dificuldade de aprender através da metodologia de ensino adotada pelo docente (FASSBINDER; PAULA; ARAUJO, 2012).

A falta de dedicação por parte do aluno e a dificuldade em perceber a lógica necessária para resolução de problemas é, em grande parte, devido aos exercícios abordados estarem fora da realidade do aluno e não possuírem um contexto ou utilidade prática na visão dele, levando a uma falta de motivação em aprender (FASSBINDER; PAULA; ARAUJO, 2012).

Justificativa

Utilizar metodologias de ensino com base em juizes eletrônicos e em competições de programação já foi objeto de estudo em outras instituições de ensino.

Na Faculdade de Computação da Universidade Nacional de Singapura, um juiz eletrônico foi aplicado como forma de automatizar o processo de correção e avaliação das tarefas de programação dos cursos introdutórios de programação (CHEANG et al., 2003).

Na Mississauga Universidade de Toronto foram utilizadas competições de programação internas nos cursos introdutórios de Ciências da Computação como forma de motivar os alunos a aprender programação (ROSENBLOOM, 2009).

A utilização de competições de programação e juizes eletrônicos no ambiente de ensino tem como objetivo principal proporcionar a oportunidade e o incentivo necessários para o desenvolvimento dos alunos quanto à capacidade de resolução de proble-

mas (FASSBINDER; PAULA; ARAUJO, 2012) e automatizar o processo de avaliação (CHEANG et al., 2003).

A prática da Engenharia de Software tem em sua essência as etapas de solução de problemas. É necessário entender o problema, planejar uma solução, executar a solução planejada e examinar os resultados (PRESSMAN, 2010). Essas são precisamente as habilidades desenvolvidas por um programador competidor (HALIM; HALIM, 2013).

Entre as habilidades necessárias de um engenheiro de software estão a capacidade de entender o problema antes de elaborar uma solução, a capacidade de desenvolver uma solução adequada dentro dos limites do problema, garantir que a solução esteja devidamente testada e de acordo com os requisitos de qualidade exigidos (PRESSMAN, 2010).

Para ser capaz de atender estes requisitos o engenheiro de software necessita de uma base sólida de conhecimentos de programação (SOMMERVILLE, 2011). Por isso a qualidade do ensino dos fundamentos da Ciência da Computação são essenciais na formação do engenheiro de software.

Diante de tal cenário, o presente trabalho procura responder a pergunta: *A utilização de juízes eletrônicos e problemas oriundos de maratonas de programação melhora a formação dos alunos de Engenharia de Software?*

Objetivos

Este trabalho tem como objetivo analisar o desempenho dos alunos de Engenharia de Software nas disciplinas em que a ementa consiste, em sua maior parte, de assuntos relacionados a programação. Ou seja, disciplinas que trabalhem com técnicas de programação, projeto e análise de algoritmos e o estudo de estruturas de dados, para que então seja possível analisar o impacto que aplicar uma metodologia de ensino que utilize juiz eletrônico como ferramenta de avaliação e problemas baseados nos moldes dos problemas de maratonas de programação tem no desempenho dos alunos posteriormente.

Os objetivos específicos deste trabalho são:

- o desenvolvimento de scripts para automatizar o processo de leitura dos históricos dos alunos do curso de Engenharia de Software e montar uma base de dados com estes históricos;
- a extração de métricas da base de dados criada a partir dos históricos dos alunos;
- realizar a análise das métricas extraídas com base no desempenho dos alunos e tirar conclusões a respeito do impacto da metodologia de ensino analisada no desempenho dos mesmos em relação ao próprio aluno e aos outros alunos.

Estrutura do Trabalho

Este trabalho está estruturado em cinco capítulos: Introdução, Fundamentação Teórica, Metodologia, Resultados e Conclusão.

A Introdução apresenta o problema a ser estudado e a hipótese a ser validada neste trabalho. Na Fundamentação Teórica são expostos os conhecimentos necessários para o entendimento do estudo executado. A Metodologia apresenta a forma que o trabalho foi executado e o que foi necessário para realizar o estudo. Resultados explicita os resultados do estudo e discute seus significados. Na Conclusão são discutidos os resultados e são propostos os próximos passos a serem tomados.

1 Fundamentação Teórica

Este capítulo apresenta os conceitos necessários para o entendimento do trabalho e o estudo realizado para a sua execução.

1.1 Maratona de Programação

O ACM *International Collegiate Programming Contest – ICPC*[1], conhecido no Brasil como Maratona de Programação, é uma competição de programação internacional dividida em três etapas, na qual alunos de universidades do mundo inteiro competem com o objetivo de definir quais são os melhores programadores do mundo.

A competição envolve universidades de diversos países, que sediam etapas regionais nas quais são selecionados os times que avançarão para a final mundial do ICPC.

A Maratona de Programação conta com a participação de dezenas de milhares de estudantes de computação no mundo todo, oriundos de mais de duas mil e quinhentas universidades localizadas em mais de cem paises ao redor do mundo (ACM, 2015). Ela fomenta a criatividade, o trabalho em equipe e a inovação no desenvolvimento de novos algoritmos. Também é uma oportunidade para testar a habilidade dos estudantes em produzir sob pressão.

As instituições de ensino superior são representadas por times compostos de três alunos, um aluno reserva e um técnico.

Em cada uma das etapas da competição os times irão receber cadernos com diversos problemas a serem resolvidos durante as 5 horas de competição, utilizando as linguagens C, C++ ou Java. Os competidores têm à disposição um e apenas um computador e qualquer tipo de material escrito que prepararem e trouxerem de antemão, mas não poderão consultar nenhum recurso armazenado em meio digital ou acessar a internet. Para a final do campeonato mundial, cada time também disporá de uma calculadora.

Ao submeter uma solução ao juiz eletrônico (Seção 1.2), o código será compilado e executado, tendo como entrada uma bateria de testes desconhecida pelos times. O problema estará correto caso todos os resultados sejam iguais aos resultados esperados pelo juiz e a execução do programa finalize antes do término do tempo limite de execução definido para cada problema. Após a correção ser finalizada pelo juiz, é apresentada ao time uma mensagem composta por uma sigla que descreve o resultado da correção do algoritmo enviado. A Tabela 1 lista os principais retornos possíveis do juiz eletrônico.

[1] http://icpc.baylor.edu/

Tabela 1: Possíveis resultados retornados pelo juiz eletrônico

Código	Descrição
AC	*Accepted*. Significa que o algoritmo enviado produziu as saídas esperadas e passou em todos os casos de teste dentro do tempo limite aceito.
WA	*Wrong Answer*. Resposta rejeitada. As saídas geradas pelo algoritmo enviado não atenderam a todos os casos de teste.
TLE	*Time Limit Exceeded*. O tempo de execução do algoritmo enviado foi superior ao tempo limite estipulado para o problema.
RE	*Runtime Error*. Ocorreu um erro durante a execução do algoritmo. Os erros mais comuns são acesso a posições irregulares de memória ou divisões por zero.
CE	*Compilation Error*. O algoritmo possui erros de sintaxe e não compila com o compilador e as flags pré-estabelecidas (os quais são de conhecimento da equipe).
PE	*Presentation Error*. A saída do seu programa não está de acordo com a formatação exigida, presente na folha de questões.

O time que resolver corretamente a maior quantidade de problemas ao final do período de cinco horas de competição é o vencedor. Caso haja um empate, vence o time que houver solucionado os problemas no menor tempo corrigido. O tempo corrigido é calculado pela soma dos tempos de submissão das soluções corretas do time desde o inicio da competição até o momento da correção somado a uma penalidade de vinte minutos por submissão incorreta submetida para um problema que for posteriormente considerado aceito pelo juiz. Em caso de empate é considerado vitorioso o time que enviou mais cedo a última submissão correta e, se ainda assim houver um empate, será realizado um sorteio para definir o ganhador entre os empatados.

Os times vencedores são classificados para a final brasileira do torneio e os times campeões da etapa nacional se classificam para a final mundial.

Os problemas de maratonas de programação em sua maioria possuem o seguinte formato: descrição do problema, descrição das entradas e saídas, exemplos de entradas e saídas e notas de rodapé com dicas relacionadas ao problema ou descrições extras (HALIM; HALIM, 2013).

A descrição do problema apresenta e explica o problema, e costuma vir acompanhada de uma história ou explicação de um conceito. O problema é uma abstração de um problema computacional em uma situação real, e é a partir da análise da descrição do problema que o competidor será capaz de determinar em qual categoria de problema aquele problema se encaixa (SKIENA; REVILLA, 2003).

As seções de entradas e saídas do problema definem quais os limites das entradas e a formatação da entrada e da saída esperada. É importante se atentar aos limites dos dados de entrada, pois algoritmos diferentes resolvem o mesmo problema para diferentes

limitações.

Após as entradas e saídas são dados exemplos de entradas e as saídas esperadas, casos de teste para que o programador possa testar seu algoritmo. Porém, os casos de teste costumam ser triviais e cabe ao competidor elaborar casos de teste mais elaborados e validar seu algoritmo antes de enviá-lo para correção. É possível, porém raro, que ao final do problema o autor dê alguma dica ou uma explicação extra a respeito da solução do problema (HALIM; HALIM, 2013).

Uma pessoa que tem interesse em ser competitivo em uma maratona de programação precisa adquirir ou desenvolver uma série de características.

O objetivo dos idealizadores da Maratona de Programação é de produzir programadores que sejam mais preparados para desenvolver softwares com qualidade e enfrentar problemas da área da ciência da computação muito mais complexos e díficeis no futuro. A maratona é um meio para este fim (HALIM; HALIM, 2013).

Entre as características de um programador competitivo estão a habilidade de identificar o tipo de problema que está se tentando resolver, a capacidade de análisar algoritmos e entender as limitações de espaço/tempo necessárias para que um algoritmo seja eficiente, a capacidade de testar com eficiência um código abrangendo todos os cenários que o problema pode prover e a capacidade de trabalhar em equipe (HALIM; HALIM, 2013).

1.1.1 Histórico da Maratona de Programação ACM ICPC

A Maratona de Programação (International College Programming Contest – ICPC no exterior) tem origem em uma competição realizada na *Texas A&M University* em 1970. Entre 1977 e 1989 as equipes provinham, principalmente, dos Estados Unidos e do Canadá. Como consequência, o país que possui o maior número de titulos mundiais da Maratona de Programação são os Estados Unidos, com dezessete campeonatos.

Porém desde o início do patrocinio da IBM, em 1997, a competição foi crescendo ano a ano e tomando âmbito mundial.

Após a instituição de medalhas como forma de premiação em 2011, o país com o maior número de medalhas é a Rússia, com dezesseis medalhas, sendo seis destas medalhas de ouro. A universidade com o maior número de vitórias no mundial é a Universidade de São Petersburgo, com oito vitórias no total.

O evento ocorre anualmente e é dividido em duas ou três etapas, a depender do número de escolas (instituições de ensino superior) distintas inscritas no país no referido ano. Atualmente no Brasil são três etapas: a regional (que ocorre a nível de estados e municípios), a nacional (com os melhores colocados na etapa regional) e a final mundial,

com os melhores representantes de cada país. O número de vagas em cada uma das etapas varia a cada ano, a depender do número de inscritos e outros fatores descritos no regulamento.

O Brasil iniciou sua participação na Maratona de Programação em 1996 e sua melhor colocação foi em 2005 pelo IME-USP, alcançando a décima terceira colocação na etapa mundial.

A Universidade de Brasília esteve presente na primeira edição da etapa nacional da Maratona de Programação no país, em 1996, onde alcançou o segundo lugar entre as três equipes participantes. O melhor resultado alcançado foi o quinto lugar na etapa nacional da competição de 2004, entre trinta e cinco equipes.

A Faculdade do Gama – FGA, localizada no campus Gama da Universidade de Brasília, iniciou suas participações na Maratona de Programação em 2012 e esteve presente nas edições de 2013 e 2014, sendo que em 2014 pela primeira vez alcançou a classificação para a etapa nacional. As classificações detalhadas das participações da FGA em maratonas estão descritas na Tabela (2).

Em 2013, por iniciativa dos docentes que foram treinadores das equipes participantes na Maratona de Programação, ocorreu a Primeira Maratona UnB de Programação. Participaram equipes de diversos cursos de tecnologia da universidade e uma das equipes de Engenharia de Software do campus Gama foi a campeã. Feito que voltou a se repetir em 2014.

Tabela 2: Melhores classificações das equipes do curso de Engenharia de Software

Competição	**Melhor Colocação**
Regional Centro-Oeste 2012	7º Lugar (10 equipes)
Regional Centro-Oeste 2013	9º Lugar (16 equipes)
Maratona UnB de Programação 2013	1º Lugar
Regional Centro-Oeste 2014	3º Lugar (19 equipes)
Final Brasileira 2014	29º Lugar (60 equipes)
Maratona UnB de Programação 2014	1º Lugar

1.2 Juiz Eletrônico

Juizes eletrônicos são programas ou websites que fornecem mecanismos de correção automática para os problemas propostos. A correção é feita através de testes unitários, e contempla desde a compilação e execução até a validação dos resultados de cada teste unitário.

Juizes eletrônicos corrigem algoritmos. Algoritmos são procedimentos para resolver uma tarefa. É a idéia por trás de um programa de computador. Um algoritmo deve ser

projetado para resolver um problema geral e bem especificado. O problema a ser resolvido é especificado descrevendo o conjunto completo de entradas e o retorno de cada instância do problema (SKIENA, 2008).

Um algoritmo só é considerado correto se passar pelo processo de compilação e execução de forma bem sucedida e por todos os testes unitários.

As linguagens de programação aceitas para correção costumam variar de acordo com o juiz eletrônico. As linguagens mais comuns são C, C++, Java e Pascal, mas é possível encontrar juízes eletrônicos que aceitam Python e Haskell. O site URIOnlineJudge[2], por exemplo, aceita submissões em Python.

Juízes eletrônicos podem ser utilizados como ambientes online que facilitem o aprendizado de linguagens de programação (CAMPOS; FERREIRA, 2004).

A Faculdade de Computação da Universidade Nacional de Singapura começou a implementar a utilização de um juiz eletrônico como forma de apoio em julho de 1999, na disciplina *CS3233 Competitive Programming*. A disciplina possui como objetivo a preparação dos alunos para a Maratona de Programação ACM ICPC, portanto o juiz eletrônico utilizado estava de acordo com os critérios estabelecidos pela competição (CHEANG et al., 2003).

A tarefa do juiz no curso de programação competitiva onde foi aplicado era de apenas executar os programas enviados com quantidade restrita de memória dentro de um limite de tempo de execução em um conjunto de casos de testes armazenados em um local seguro. Após a execução, os resultados dos casos de teste eram comparados com as respostas esperadas (CHEANG et al., 2003).

Dois aspectos da resposta foram avaliados: o quão correta ela está e o quão eficiente ela é. O critério utilizado para avaliar o quão correta a resposta está é quantas das respostas produzidas pelo seu algoritmo combinam com as respostas esperadas. O critério para avaliar a eficiência da resposta é se o algoritmo é capaz de produzir os resultados esperados dentro dos limites de tempo e memória pré-estipulados (CHEANG et al., 2003).

1.3 Conteúdos e Problemas abordados em Maratonas de Programação

As maratonas de programação contam com um conjunto de problemas, os quais podem ser agrupados em categorias. As categorias definem a área de conhecimento onde os problemas se encontram. Elas são determinadas pelo tipo de algoritmo a ser utilizado para que se chegue a uma solução e pelas características do problema. Cabe ao competidor

[2] <urionlinejudge.com.br>

identificar em qual categoria o problema apresentado está inserido, de forma que seja possível elaborar uma solução correta.

Os problemas presentes nas maratonas de programação são problemas já solucionados e conhecidos na área da Ciência da Computação. Ou seja, não são problemas que ainda estão sendo estudados ou pesquisados: a solução para eles já foi encontrada (HALIM; HALIM, 2013).

1.3.1 Diretórios de problemas

Existem diversos diretórios onlines com milhares de problemas armazenados e que contam com juízes eletrônicos online, possibilitando que o usuário elabore soluções para os problemas e os use como material de estudo.

Podemos afirmar que o diretório online de problemas mais popular no Brasil é o URI Online Judge[3], da universidade URI Erechim, com mais de 700 problemas, ranqueamento dos usuários, fórum, competições online e um módulo acadêmico para professores e treinadores criarem listas e acompanhar o desenvolvimento dos alunos.

O diretório online mais popular em nível mundial entre os participantes da Maratona de Programação ACM ICPC é o UVa Online Judge[4], da Universidade de Valladolid.

Outra opção é o ACM ICPC Live Archive[5], onde estão armazenados centenas de problemas das etapas regionais e mundiais da Maratona de Programação.

1.3.2 Categorias dos problemas

1.3.2.1 Ad Hoc

São problemas que não possuem uma classificação específica entre as outras categorias, por existir um tipo de algoritmo específico para resolvê-los. Entram nesta categoria problemas nos quais, para que se alcance a solução, é necesário que seja implementado um conjunto de regras, considerando casos especiais.

Jogos de cartas e jogos de tabuleiro são exemplos de problemas que se encaixam nesta categoria.

1.3.2.2 Processamento de Strings

São problemas relacionados a manipulação de conjuntos de caracteres. Algoritmos de busca de substrings, contagem de frequência de letras ou palavras, cifrar e decifrar strings, expressões regulares, formatação de strings e comparação de strings são apenas

[3] <www.urionlinejudge.com.br>
[4] <uva.onlinejudge.org>
[5] <icpcarchive.ecs.baylor.edu>

algumas das aplicações de algoritmos de processamento de strings e são de fundamental importância para todo programador conhecer.

1.3.2.3 Matemática

Dentre todas as categorias, provavelmente é a que possui maior variedade de problemas. Os problemas da categoria de matemática exploram, entre outros temas, teoria dos números, álgebra linear, análise combinatória, números gigantes, fatoriais e relações de recorrência.

Muitos problemas da vida real podem ser modelados como problemas matemáticos (HALIM; HALIM, 2013) e a capacidade de modelar estes problemas em notações matemáticas faz com que o programador tenha a sua disposição uma gama enorme de ferramentas para solucioná-lo.

1.3.2.4 Estruturas de Dados e Bibliotecas

Os problemas nesta categoria costumam se limitar a escolher a estrutura de dados adequada para a resolução do problema. Para isso é necessário o conhecimento das vantagens e desvantagens de cada uma e o seu funcionamento. A eficiência em inserir, deletar, buscar e/ou alterar os dados armazenados nas estruturas costuma ter um papel fundamental na resolução do problema.

Porém, apenas a utilizar a estrutura de dados adequada não garante a solução do problema, o que resolve o problema: é o algoritmo utilizado.

1.3.2.5 Paradigmas de Resolução de Problemas

Paradigmas de resolução de problemas são formas de atacar um problema. Os paradigmas recorrentes são: *Backtracking* (SEDGEWICK, 1990), Dividir para Conquistar (HALIM; HALIM, 2013), Algoritmos Gulosos (CORMEN, 2009) e Programação Dinâmica (SKIENA; REVILLA, 2003). São técnicas de resolução de problemas fundamentais para todo participante de competições de programação.

1.3.2.6 Geometria Computacional

É um tópico que aparece frequentemente em competições. Problemas de geometria computacional envolvem a geometria básica, que engloba áreas, volumes, distâncias entre pontos, polígonos e problemas de geometria computacional, os quais envolvem algoritmos complexos, como encontrar o menor polígono convexo formado por um conjunto de pontos.

Normalmente possuem muitos problemas nos detalhes e nos testes de borda e são mais complicados de se chegar a uma resposta correta. É comum que, por exemplo, por problemas de arredondamento, um algoritmo correto não passe pelo juiz.

1.3.2.7 Grafos

Grafos são estruturas de dados, e muitos problemas em situações reais podem ser abstraídos na forma de grafos. Existem vários algoritmos de grafos, dentre eles, algoritmos de travessia e algoritmos que testam se o grafo possui alguma propriedade especial.

1.4 Tipos de Arquivo

1.4.1 *Portable Document Format* (PDF)

Criado pela Adobe Systems em 1993 como um formato proprietário, o PDF tem como objetivo representar documentos de forma independente do aplicativo, hardware ou sistema operacional que o criou. Em 2008 o formato do PDF foi publicado como formato aberto na ISO 32000-1.

Um arquivo PDF agrega toda a informação necessária para que seja possível visualizá-lo. Pelo fato de a especificação de arquivos no formato PDF ser aberta, é possível escrever aplicações que façam uso do formato. Apesar de ser um formato aberto, a patente pertence a Adobe System, porém ele foi licenciado pela própria Adobe como *royalty-free*, não sendo necessário pagar à Adobe para utilizá-lo em suas aplicações.

1.4.2 *JavaScript Object Notation* (JSON)

Baseado na 3ª Edição do Standard ECMA-262 de dezembro de 1999 da linguagem Javascript e posteriormente descrito nas publicações RFC7159 e ECMA-404, o JSON é um formato de arquivo de texto independente de linguagem de programação. Ele é de facíl legibilidade e tem como objetivo simplificar a transmissão de dados entre aplicações de diferentes linguagens. Por ser independente de linguagem e em formato de texto, ele é o formato ideal para troca de dados entre aplicações.

O JSON é fundamentado em duas estruturas:

1. Pares nome/valor, implementados em diversas linguagens e conhecidos por vários nomes, sendo eles: *object*, *record*, *struct*, dicionário, *hash table*, *keyed list* ou *arrays* associativas.

2. Lista ordenada de valores. Na maioria das linguagens é conhecida como *array*, lista, vetor ou sequência.

Os dados em JSON são formados por objetos, listas e valores.

Objetos são formados por um conjunto desordenado de pares nome/valor separados por vírgula e envoltos por chaves (“{”, “}”) (Figura 1). O nome é separado do valor por dois pontos (“:”).

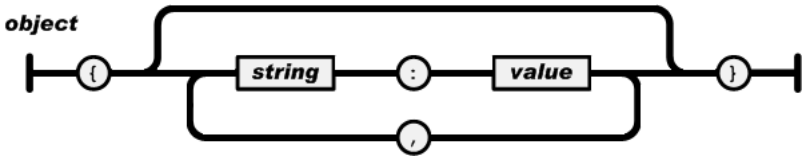

Figura 1: Estrutura de um Objeto JSON

Uma lista é formada por uma coleção de valores ordenados, separados por vírgula, entre colchetes (“[”, “]”), conforme ilustra a Figura 2.

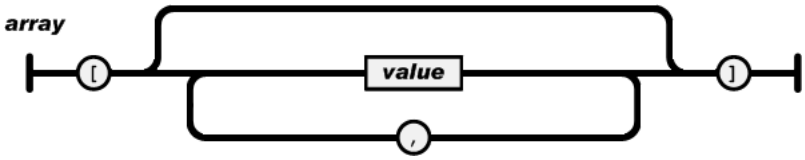

Figura 2: Estrutura de um Array JSON

JSON possui seis tipos básicos. Os valores podem ser de qualquer um dos tipos básicos listados na Figura 3 e descrito na Tabela 3.

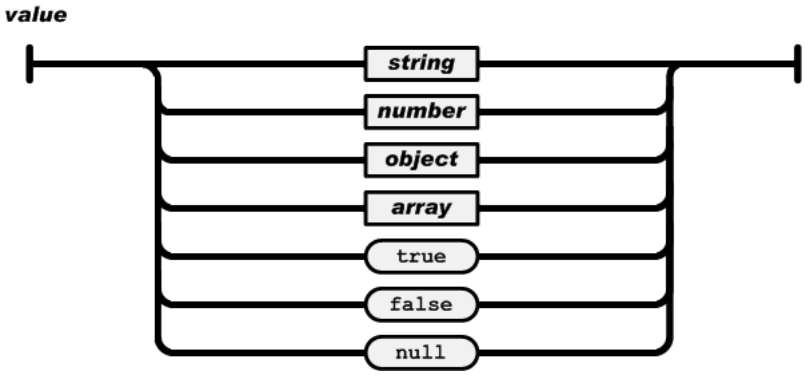

Figura 3: Valores de dados possíveis no JSON

A título de exemplo, a Figura 4 apresenta o formato válido para um número em JSON, enquanto que a Figura 5 mostra o formato válido para strings em JSON.

Tabela 3: Tipos Básicos do JSON

`Nome`	**Descrição**
`string`	Uma sequência de caracteres Unicode delimitados por aspas duplas.
`número`	Um número decimal sinalizado que pode conter parte fracionária e utilizar a notação exponecial E.
`objeto`	Um objeto JSON, composto por pares nome/valor.
`array`	Uma lista ordenada de valores.
`boleano`	Verdadeiro (`true`) ou falso (`false`).
`null`	Representa um valor vazio.

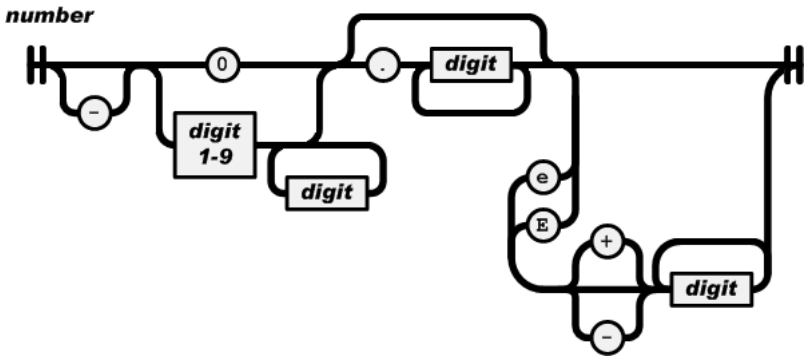

Figura 4: Formato válido de um número JSON

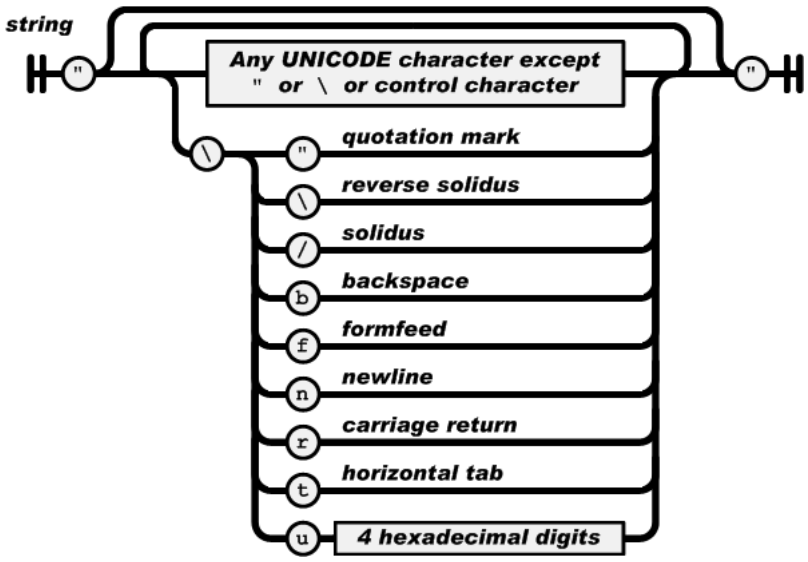

Figura 5: Formato válido de uma string JSON

2 Metodologia

Neste capítulo está descrita a metodologia utilizada para a realização deste trabalho.

2.1 Metodologia de Ensino com apoio de Juiz Eletrônico

O trabalho consiste na análise da utilização de juízes eletrônicos e problemas no formato de problemas utilizados em maratonas de programação na parte prática da metodologia de ensino.

As disciplinas que utilizaram juiz eletrônico foram disciplinas que tinham como foco o ensino de técnicas de programação ou conceitos relacionados a análise e construção de algoritmos e estruturas de dados. A parte teórica do curso foi dada pelo professor no formato de aula convencional, enquanto que a avaliação dos alunos foi realizada por meio de listas de exercícios e provas.

As provas foram realizadas com problemas no formato de problemas de maratona de programação abordando situações em que os conceitos ensinados em sala de aula fossem aplicados. Os alunos então elaboravam um algoritmo que soluciona o problema, enviavam o algoritmo para o juíz eletrônico compilar o código e executar os testes. O aluno recebia, em tempo real, a informação de se a resposta estava correta ou não de acordo com as respostas que um juiz eletrônico devolvia (Tabela 1). Exemplos de problemas utilizados nas avaliações se encontram nos Anexos.

Foi realizado um levantamento inicial das disciplinas do curso de Engenharia de Software e Ciências da Computação da UnB e observado quais categorias de problemas da maratona são abordados durante o curso e em quais disciplinas (Tabela 4).

É importante apontar que a disciplina Tópicos Especiais em Programação (110141) do curso de Engenharia de Software ficou de fora desta tabela. Está disciplina tem como objetivo ensinar os conhecimentos necessários para futuros maratonistas. A disciplina será ministrada todos os semestres e em cada semestre será abordado uma categoria de conhecimento. Ela teve início no primeiro semestre de 2015, com Matemática, portanto não faz parte deste estudo.

O objetivo desse levantamento foi compreender se, ao se preparar para uma maratona de programação, o aluno está ou não extrapolando os conhecimentos dados em sala de aula.

Tabela 4: Disciplinas de graduação vs. categorias de problemas

Categorias de Problemas	**Disciplinas do Curso de Engenharia de Software**	**Disciplinas do Curso de Ciência da Computação**
Ad Hoc	Introdução a Ciências da Computação (113913), Computação Básica (116301)	Computação Básica (116301)
Processamento de Strings	Programação Para Competições (103195)	Tradutores (116459)
Matemática	Estruturas Matemáticas para Computação (195405), Tópicos Especiais em Sistemas Críticos (107417)	Probabilidade e Estatística (115045), Álgebra Linear (113123), Teoria dos Números 1 (113115)
Estrutura de Dados	Estrutura de Dados e Algoritmos (193704), Estrutura de Dados e Algoritmos 2 (103209)	Estruturas de Dados (116319)
Paradigmas de Resolução de Problemas	Estrutura de Dados e Algoritmos 2 (103209), Programação Para Competições (103195)	Projeto e Análise de Algoritmos (117536)
Geometría Computacional	Programação Para Competições (103195)	
Grafos	Estrutura de Dados e Algoritmos 2 (103209)	Estruturas de Dados (116319), Introdução a Teoria dos Grafos (113930)

2.2 Metodologia de trabalho

O trabalho foi executado em três etapas. A primeira etapa consistiu na aquisição e tratamento dos dados. Na segunda etapa as métricas foram definidas e extraídas da base de dados formada. Na terceira foi feita a análise e discussão dos resultados.

2.2.1 Aquisição e tratamento dos dados

O objetivo do trabalho é de medir e avaliar o impacto da utilização de juízes eletrônicos e problemas oriundos de maratonas de programação na melhoria da formação acadêmica e profissional de alunos do curso de Engenharia de Software. O recurso necessário para a investigação deste impacto é o desenvolvimento de rotinas e procedimentos que permitam auferir o desempenho dos alunos antes e após a utilização desta metodologia em sala de aula. Como fonte primária de dados para a auferição desta informação foram utilizados os históricos acadêmicos dos alunos e ex-alunos de Engenharia de Software da Faculdade UnB Gama.

Junto à coordenação do curso de Engenharia de Software, foi realizado o pedido dos históricos de todos os alunos da graduação, desde o seu início (que data do segundo semestre do ano de 2009). Os históricos foram disponibilizados no formato PDF (Seção 1.4). Porém, devido ao fato dos arquivos PDF estarem em formato binário, foram detectados dois problemas.

Os atributos dos históricos que identificam os alunos devem ser removidos para que

o estudo fosse realizado de maneira impessoal e a privacidade dos alunos fosse preservada. A utilização dos arquivos em formato PDF diretamente implicaria na remoção manual de tais atributos. Uma vez que haviam 388 históricos a serem processados, remover tais atributos identificadores de cada aluno, um por vez, demandaria um grande intervalo de tempo e a tarefa estaria suscetível a erros humanos. Além disso, esta "limpeza" dos dados, por motivos de confidencialidade e privacidade dos alunos e ex-alunos, deveria ser realizada pelo professor orientador, antes que estes dados pudessem ser repassados para o orientando para fins de estudo. Este foi o primeiro problema.

O segundo problema encontrado foi popular a base de dados a ser analisada. O formato binário dos documentos impossibilita a automatização da tarefa. Portanto seria necessário a entrada manual dos dados de cada um dos históricos. Dado o volume de informações disponíveis, o tempo despendido na execução desta tarefa seria proibitivo e prejudicial para o andamento da pesquisa.

Para contornar os problemas encontrados foi realizada a conversão dos históricos em PDF para arquivos em formato de texto (TXT). Inicialmente, foi utilizado um serviço de conversão de arquivos online[1], mas esta tarefa foi automatizada por meio de dois scripts: um escrito em Shell Script (Código 2.1) e outro escrito em Linguagem Python (Código 2.2), que realizou a conversão dos históricos em PDF para TXT e apagou os atributos identificadores de cada um, substituindo-os por um identificador numérico único e sequencial, respectivamente.

Código 2.1: Conversão dos arquivos em PDF para TXT

```
#!/bin/bash

for pdf in *.pdf; do
    pdftotext "$pdf";
done
```

Código 2.2: Limpeza dos dados pessoais dos alunos dos históricos

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import glob

def find_files():
    files = glob.glob('*.txt')
    return sorted(files)

def find_name(path):
    name = False
    with open(path) as f:
```

[1] <cloudconvert.com/pdf-to-txt>

```
13          for line in f:
14              line = line.strip()
15              if line == 'NOME DO ALUNO':
16                  name = True
17                  continue
18              if line == 'MATRICULA' or len(line) == 0:
19                  continue
20              if name:
21                      return line
22      return 'Not found'
23
24  def is_matricula(text):
25      if len(text) < 3:
26          return False
27      if text[2] != '/':
28          return False
29      text = text[0:2] + text[3:]
30      return text.isdigit()
31
32  def remove_personal_data(path, sid):
33      name = find_name(path)
34      matricula = ''
35      lines = []
36      with open(path) as f:
37          for line in f:
38              line = line.strip()
39              if line == name:
40                  print 'Removed name: [{}] (SID {})'.format(line, sid)
41                  continue
42              if 'Pai' in line or 'Mae' in line:
43                  print 'Removed parents: [{}]'.format(line)
44                  continue
45              if is_matricula(line):
46                  matricula = line
47                  print 'Removed ID: [{}]'.format(line)
48                  lines.append('StudentID: {}'.format(sid))
49                  continue
50              lines.append(line)
51      output = '{}.data'.format(sid)
52      data = '\n'.join(lines)
53
54      with open(output, 'w') as f:
55          f.write(data)
56
57      with open('sid.list', 'a') as f:
58          f.write('{}: {}\n'.format(sid, matricula))
59
```

```
60 if __name__ == '__main__':
61     files = find_files()
62     for i in range(len(files)):
63         print '\nProcessing {}..'.format(files[i])
64         remove_personal_data(files[i], i + 1)
```

2.2.2 Definição de métricas e extração dos dados

Para realizar a análise do desempenho dos estudantes foi definida uma série de métricas a serem extraídas.

Para isso foi necessário definir alguns pontos importantes. O primeiro ponto foi a identificação de quais disciplinas utilizaram juiz eletrônico como metodologia de ensino e avaliação dos alunos e quando que elas foram aplicadas. O segundo ponto foi a discriminação das disciplinas que possuem o ensino da programação como foco, para que o desempenho nessas disciplinas fosse avaliado separadamente. As turmas que utilizaram juiz eletrônico estão listadas na Tabela 5.[2]

Tabela 5: Disciplinas que utilizaram juiz eletrônico

Código	Nome	Ano	Período
193704	Estruturas de Dados e Algoritmos	2012	2
193704	Estruturas de Dados e Algoritmos	2013	1
208493	Introdução aos Jogos Eletrônicos	2012	2
208493	Introdução aos Jogos Eletrônicos	2013	1
208493	Introdução aos Jogos Eletrônicos	2013	2
103195	Programação para Competições	2013	1
103195	Programação para Competições	2013	2
103195	Programação para Competições	2014	1
103195	Programação para Competições	2014	2

As disciplinas que possuem o ensino de programação como foco estão listadas na Tabela 6. Para decidir quais eram estas disciplinas, foi utilizada como referência a Matriz Curricular do Curso de Engenharia de Software da Faculdade UnB Gama. A inclusão das disciplinas no estudo foi consequência da análise de ementa das mesmas.

O desempenho dos alunos foi calculado utilizando uma média ponderada das menções em cada disciplina cursada, levando em consideração apenas as disciplinas onde os alunos obtiveram como menção SR, II, MI, MM, MS e SS, e o número de créditos da disciplina. Para cada uma dessas menções foi atribuído um peso, de acordo com a Tabela 7.

[2] Existem outras disciplinas que utilizaram juizes eletrônicos (por exemplo, Computação Básica 2014/2 e 2015/1), mas o juiz eletrônico não era o foco principal da metodologia e nem os problemas eram baseados/formatados conforme as maratonas de programação, como foi o caso das disciplinas citadas.

Tabela 6: Disciplinas de Programação

Código	**Nome**
113913	Introdução a Ciências da Computação
116301	Computação Básica
195405	Estruturas Matemáticas para Computação
206199	Inteligência Artificial
110141	Tópicos Especiais em Programação
195413	Métodos Numéricos para Engenharia
206601	Desenvolvimento Avançado de Software
101095	Fundamentos de Compiladores
208507	Introdução a Computação Gráfica
103209	Estrutura de Dados e Algoritmos 2
201286	Fundamentos de Sistemas Operacionais
206181	Sistemas Embarcados
195341	Orientação a Objetos
193704	Estrutura de Dados e Algoritmos
203904	Paradigmas de Programação
107409	Tópicos Especiais em Jogos Digitais
107417	Tópicos Especiais em Sistemas Críticos
206598	Manutenção e Evolução de Software
193640	Métodos de Desenvolvimento de Software
201294	Técnicas de Programação
208493	Introdução aos Jogos Eletrônicos
103195	Programação Para Competições
117552	Introdução ao Desenvolvimento de Jogos
203882	Desenho de Software
208701	Programação Web

Tabela 7: Pesos por menção

Menção	**Peso**
SR	0
II	1
MI	2
MM	3
MS	4
SS	5

Este o peso utilizado no cálculo do desempenho D do aluno (Equação 2.1), onde D é o desempenho do aluno, P_i o peso da menção obtida e C_i o número de créditos da disciplina i.

$$D = \frac{\sum_i P_i \cdot C_i}{\sum_i C_i} \tag{2.1}$$

Os Códigos 2.3, 2.4 e 2.5 apresentam as rotinas utilizadas para computar o desempenho dos alunos segundo três critérios: desempenho global, desempenho antes do

aluno cursar disciplinas que utilizavam juiz eletrônico e após o aluno cursar sua primeira disciplina com juiz eletrônico.

Código 2.3: Procedimento em Python para cálculo do desempenho do aluno

```
# Calcula o desempenho geral de um aluno
def calcularIRA(aluno):
    #Inicia o IRA e o contador
    ira = 0.0
    num_disciplinas = 0
    # Passa por cada semestre do aluno
    for semestre in aluno["historico"]:
      # Passa por cada disciplina do semestre
        for disciplina in semestre["disciplinas"]:
          # Se a mencao tiver um peso valido
            if pesos[disciplina["mencao"]] != -1:
              # Soma o peso da mencao multiplicada pelo numero de
                  creditos da disciplina
                ira += pesos[disciplina["mencao"]] * int(disciplina["
                    credito"])
                # Por ser uma media ponderada dividimos pelo numero de
                    creditos
                num_disciplinas += int(disciplina["credito"])
    # Retorna o desempenho
    return ira / num_disciplinas
```

Código 2.4: Procedimento em Python para cálculo do desempenho do aluno antes de usar juiz eletrônico

```
# Calcula o desempenho do aluno antes dele fazer uma materia com ejudge
def calcularIRAAntesEjudge(aluno):
  #Inicia o IRA e o contador
    ira = 0.0
    num_disciplinas = 0
    # Pega quando o aluno fez a materia com ejudge
    turma = aluno["ejudge_turma"]
    # Passa por cada semestre do aluno
    for semestre in aluno["historico"]:
      # Apenas os semestres anteriores a disciplina
        if semestre["ano"] <= turma["ano"] and semestre["periodo"] <=
            turma["periodo"]:
            # Passa por cada disciplina do semestre
            for disciplina in semestre["disciplinas"]:
              # Se a mencao tiver um peso valido
                if pesos[disciplina["mencao"]] != -1:
                  # Soma o peso da mencao multiplicada pelo numero de
                      creditos da disciplina
                    ira += pesos[disciplina["mencao"]]  * int(disciplina
                        ["credito"])
```

```
18                      # Por ser uma media ponderada dividimos pelo numero
                            de creditos
19                      num_disciplinas += int(disciplina["credito"])
20      # Se nao houver semestre antes, retorna 0
21      if num_disciplinas == 0:
22          return 0.0
23      # Retorna o desempenho
24      return ira / num_disciplinas
```

Código 2.5: Procedimento em Python para cálculo do desempenho do aluno depois de usar juiz eletrônico

```
1   # Calcula o desempenho do aluno depois dele fazer uma materia com ejudge
2   def calcularIRADepoisEjudge(aluno):
3       #Inicia o IRA e o contador
4       ira = 0.0
5       num_disciplinas = 0
6       # Pega quando o aluno fez a materia com ejudge
7       turma = aluno["ejudge_turma"]
8       # Passa por cada semestre do aluno
9       for semestre in aluno["historico"]:
10        # Apenas os semestres posteriores a disciplina
11          if semestre["ano"] >= turma["ano"] and semestre["periodo"] >=
                turma["periodo"]:
12              # Passa por cada disciplina do semestre
13              for disciplina in semestre["disciplinas"]:
14                  # Se a mencao tiver um peso valido
15                      if disciplina["codigo"] in disciplinasProgramacao and
                            pesos[disciplina["mencao"]] != -1:
16                          # Soma o peso da mencao multiplicada pelo numero de
                                creditos da disciplina
17                          ira += pesos[disciplina["mencao"]] * int(disciplina[
                                "credito"])
18                          # Por ser uma media ponderada dividimos pelo numero
                                de creditos
19                          num_disciplinas += int(disciplina["credito"])
20      # Se nao houver semestre depois, retorna 0
21      if num_disciplinas == 0:
22          return 0.0
23      # Retorna o desempenho
24      return ira / num_disciplinas
```

A segunda forma de avaliação dos resultados alcançados foi a análise do desempenho dos alunos em matérias de programação anteriores e posteriores as turmas que tiveram como metodologia de ensino o uso de juiz eletrônico. As rotinas para a obtenção destes dados são apresentadas nos Códigos 2.6, 2.7 e 2.8, respectivamente.

Código 2.6: Procedimento em Python para cálculo do desempenho do aluno em disciplinas de programação

```
# Calcula o desempenho de um aluno com base nas disciplinas passadas por parametro
def calcularIRASeletivo(aluno,disciplinas):
    #Inicia o IRA e o contador
    ira = 0.0
    num_disciplinas = 0
    # Passa por cada semestre do aluno
    for semestre in aluno["historico"]:
        # Passa por cada disciplina do semestre
        for disciplina in semestre["disciplinas"]:
            # Se a mencao tiver um peso valido
            # E o codigo da disciplina esta entre as disciplinas alvo
            if pesos[disciplina["mencao"]] != -1 and disciplina["codigo"] in disciplinas:
                # Soma o peso da mencao multiplicada pelo numero de creditos da disciplina
                ira += pesos[disciplina["mencao"]] * int(disciplina["credito"])
                # Por ser uma media ponderada dividimos pelo numero de creditos
                num_disciplinas += int(disciplina["credito"])
    # Se nenhuma disciplina faz parte, retorna zero
    if num_disciplinas == 0:
        return 0
    # Retorna o desempenho
    return ira / num_disciplinas
```

Código 2.7: Procedimento em Python para cálculo do desempenho do aluno em disciplinas de programação antes de usar juiz eletrônico

```
# Calcula o desempenho do aluno antes dele fazer uma materia com ejudge
def calcularIRAAntesEjudge(aluno):
  #Inicia o IRA e o contador
    ira = 0.0
    num_disciplinas = 0
    # Pega quando o aluno fez a materia com ejudge
    turma = aluno["ejudge_turma"]
    # Passa por cada semestre do aluno
    for semestre in aluno["historico"]:
      # Apenas os semestres anteriores a disciplina
        if semestre["ano"] <= turma["ano"] and semestre["periodo"] <= turma["periodo"]:
            # Passa por cada disciplina do semestre
            for disciplina in semestre["disciplinas"]:
              # Se a mencao tiver um peso valido
```

```
15                  if pesos[disciplina["mencao"]] != -1:
16                    # Soma o peso da mencao multiplicada pelo numero de
                          creditos da disciplina
17                    ira += pesos[disciplina["mencao"]]  * int(disciplina
                          ["credito"])
18                    # Por ser uma media ponderada dividimos pelo numero
                          de creditos
19                    num_disciplinas += int(disciplina["credito"])
20      # Se nao houver semestre antes, retorna 0
21      if num_disciplinas == 0:
22          return 0.0
23      # Retorna o desempenho
24      return ira / num_disciplinas
```

Código 2.8: Procedimento em Python para cálculo do desempenho do aluno em disciplinas de programação depois de usar juiz eletrônico

```
1   # Calcula o desempenho do aluno depois dele fazer uma materia com ejudge
2   def calcularIRADepoisEjudge(aluno):
3       #Inicia o IRA e o contador
4       ira = 0.0
5       num_disciplinas = 0
6       # Pega quando o aluno fez a materia com ejudge
7       turma = aluno["ejudge_turma"]
8       # Passa por cada semestre do aluno
9       for semestre in aluno["historico"]:
10        # Apenas os semestres posteriores a disciplina
11          if semestre["ano"] >= turma["ano"] and semestre["periodo"] >=
                turma["periodo"]:
12              # Passa por cada disciplina do semestre
13              for disciplina in semestre["disciplinas"]:
14                # Se a mencao tiver um peso valido
15                  if disciplina["codigo"] in disciplinasProgramacao and
                        pesos[disciplina["mencao"]] != -1:
16                    # Soma o peso da mencao multiplicada pelo numero de
                          creditos da disciplina
17                    ira += pesos[disciplina["mencao"]] * int(disciplina[
                          "credito"])
18                    # Por ser uma media ponderada dividimos pelo numero
                          de creditos
19                    num_disciplinas += int(disciplina["credito"])
20      # Se nao houver semestre depois, retorna 0
21      if num_disciplinas == 0:
22          return 0.0
23      # Retorna o desempenho
24      return ira / num_disciplinas
```

2.2.3 Análise

A segunda parte do estudo é a análise de turmas de disciplinas do curso. Para isso foram selecionadas as disciplinas Estruturas de Dados e Algoritmos 2 (103209) e Paradigmas de Programação (203904). Estas são disciplinas que possuem na ementa o foco em programação. A capacidade do aluno em desenvolver algoritmos eficientes está intimamente ligada ao desempenho alcançado nestas disciplinas.

Em cada uma destas turmas foi feito um histograma das menções e identificado quais alunos fizeram as disciplinas que utilizaram juiz eletrônico e quais alunos não fizeram antes da disciplina. Após essa separação foi tirada a média do desempenho dos alunos naquela disciplina que não tiveram contato com juiz eletrônico e dos alunos que tiveram contato.

Todos os scripts e resultados do estudo estão disponíveis em um repositório GIT púbico hospedado na plataforma GitHub[3] e estão no Apêndice deste trabalho.

2.3 Ferramentas

Para a execução deste trabalho foi utilizado um MacBook Pro (Retina, 13-inch, Late 2013) com OSX Yosemite 10.10.4. A Tabela 8 lista as ferramentas utilizadas para o tratamento, processamento e visualização dos dados.

Tabela 8: Ferramentas utilizadas para a realização do trabalho

Nome	**Versão**	**Descrição**
Python	2.7.6	Linguagem de programação utilizada para a escrita dos scripts para realizar o tratamento dos dados, criar a base de dados e extrair as métricas da base de dados.
pdftotext	0.24.5	Ferramenta parte do pacote poppler-utils do linux utilizada para realizar a conversão de todos os históricos em PDF para formato de texto, TXT.
GIT e GitHub	1.9	É uma ferramenta de controle de versão distribuída gratuitamente, de código aberto.
Brackets	1.3	Editor de texto desenvolvido pela Adobe utilizado para o desenvolvimento dos scripts em python e javascript deste trabalho.
gulp e gulp-connect	3.8.11 e 2.2.0	Ferramenta de automação de tarefas em Javascript.

[3] <github.com/runys/students_history_parsing>

3 Resultados

Este capítulo tem como objetivo apresentar os resultados do processamento das informações dos históricos dos estudantes do curso de Engenharia de Software da Universidade de Brasília até o segundo semestre de 2014, além de analisar e discutir as métricas extraídas e os resultados encontrados.

A análise foi realizada a partir de dois pontos de vista: a análise do desempenho individual dos alunos e a análise e comparação do desempenho dos alunos dentro de turmas específicas.

A primeira parte deste capítulo consiste nos resultados da análise do desempenho individual de cada aluno que teve contato com as disciplinas que utilizaram juiz eletrônico no apoio à metodologia de ensino da disciplina, chamadas a partir de agora de "disciplinas com juiz eletrônico". A segunda parte discute o desempenho destes alunos em disciplinas de programação posteriores as disciplinas com juiz eletrônico, comparando o seu desempenho na disciplina com o desempenho dos alunos na turma que não cursaram disciplinas com juiz eletrônico.

3.1 Métricas de Aluno

O grupo de alunos que fazem parte deste estudo consiste de todos os alunos do curso de Engenharia de Software, desde o início do curso no segundo semestre de 2009 até o segundo semestre do ano de 2014.

O objetivo de analisar o desempenho dos alunos individualmente é observar se houve ou não melhora em seu desempenho acadêmico nas matérias onde a ementa fosse composta, essencialmente, de assuntos relacionados a programação (Tabela 5).

O índice de desempenho anterior (IDA) contabiliza o desempenho do aluno levando em consideração as menções alcançadas por ele nas disciplinas cursadas no período anterior ao semestre, e o semestre, no qual cursou uma disciplina com juiz eletrônico.

O índice de desempenho posterior (IDP) contabiliza o desempenho do aluno levando em consideração as menções alcançadas por ele nas disciplinas cursadas no período posterior ao semestre, e o semestre, em que cursou a disciplina com juiz eletrônico.

O delta (δ) é a diferença entre o índice de desempenho posterior e o índice de desempenho anterior. O delta percentual (Δ) é quanto o índice de desempenho posterior aumentou ou diminui percentualmente em relação ao índice de desempenho anterior.

3.1.1 Desempenho geral antes e depois

Esta métrica tem como objetivo observar se a dinâmica de estudo necessária para obter aprovação em uma disciplina que utilize juiz eletrônico como método de avaliação impacta de alguma forma o desempenho geral do estudante. A Tabela 9 apresenta os desempenho de uma amostra de alunos que cursaram a disciplina com juiz eletrônico, ordenados por Δ.

Tabela 9: Amostra do desempenho geral dos alunos

Aluno (ID)	D	**IDA**	**IDP**	δ	Δ
221	3,34	3,05	4,34	1,30	42,56%
005	2,61	2,37	3,23	0,87	36,55%
251	3,11	2,72	3,67	0,95	35,10%
020	2,95	2,81	3,79	0,98	34,79%
002	2,32	1,82	2,45	0,63	34,42%
041	3,28	2,70	3,57	0,87	32,03%
132	3,12	2,64	3,46	0,82	31,13%
316	3,34	3,09	4,04	0,94	30,47%
312	3,44	3,12	4,04	0,92	29,52%
227	3,24	2,77	3,58	0,80	28,96%
236	2,43	2,43	3,11	0,68	28,10%
028	3,43	3,27	4,14	0,87	26,66%
378	3,32	2,97	3,73	0,76	25,58%
245	3,41	3,19	3,98	0,79	24,79%
256	2,80	2,64	3,29	0,65	24,69%
...	...	...	...	...	...
010	3,13	3,00	2,50	-0,50	-16,67%
339	2,67	2,69	2,20	-0,49	-18,32%
032	2,19	2,55	2,02	-0,52	-20,49%
017	3,16	3,36	2,65	-0,71	-21,15%
113	2,42	3,11	2,24	-0,87	-28,10%
139	2,84	3,68	2,61	-1,06	-28,93%
044	3,24	3,24	2,21	-1,03	-31,73%
054	2,09	2,88	1,89	-0,99	-34,26%
362	2,63	2,89	1,88	-1,01	-34,88%
343	1,96	2,37	1,40	-0,97	-40,89%
187	2,18	2,37	1,13	-1,23	-52,20%
299	2,50	2,10	1,00	-1,10	-52,38%
361	2,11	2,56	1,18	-1,39	-54,09%
379	1,81	2,00	0,89	-1,11	-55,56%
283	2,04	2,04	0,00	-2,04	-100,0%

Dos 388 alunos de Engenharia de Software incluídos no estudo, 133 fizeram disciplinas com juiz eletrônico. Os índices extraídos correspondem ao desempenho destes 133 alunos. No total:

- 93 alunos obtiveram um aumento no desempenho;
- 40 alunos obtiveram uma queda no desempenho;
- 73 alunos obtiveram um aumento no desempenho maior que 5%;
- 30 alunos obtiveram uma queda no desempenho maior que 5%;
- 30 alunos ficaram entre 5% de aumento e 5% de queda.

Com base nos resultados dessa análise de desempenho geral do aluno, ou seja, levando em consideração todas as disciplinas cursadas, podemos observar que 54.9% dos alunos tiveram um desempenho posterior ao contato com juiz eletrônico maior do que o desempenho anterior.

Isso indica que o contato com a metodologia de ensino pode ter tido um impacto positivo significativo no aluno. Observamos que o desempenho posterior IDP igual a zero significa que o aluno fez uma disciplina com juiz eletrônico no último semestre anterior a esse estudo, então não há disciplinas a serem analisadas.

3.1.2 Desempenho em matérias de programação antes e depois

Esta métrica tem como objetivo observar o impacto que o contato com uma metodologia que utilize juiz eletrônico como forma de avaliação exerce no desempenho do aluno nas disciplinas de programação a partir daquele momento.

O desempenho anterior em programação (IDA_P) é calculado levando em consideração o desempenho nas matérias, classificadas como de programação para este estudo até cursar a disciplina com juiz eletrônico, incluindo a própria disciplina.

O desempenho posterior em programação (IDP_P) é calculado levando em consideração a menção do aluno nas disciplinas de programação cursadas após ter cursado a primeira disciplina com juiz eletrônico.

A Tabela 10 apresenta estes dois novos índices para uma amostra dos alunos que cursaram disciplinas com juiz eletrônico, ordenados por Δ.

No total:

- 67 alunos obtiveram um aumento no desempenho;
- 60 alunos obtiveram uma queda no desempenho;
- 6 alunos mantiveram o desempenho;
- 49 alunos obtiveram um aumento no desempenho em matérias de programação maior que 5%;

Tabela 10: Amostra do desempenho dos alunos em matérias de programação

Aluno (ID)	D	IDA_P	IDP_P	δ	Δ
078	2,63	1,73	3,00	1,27	73,68%
029	2,33	1,75	2,64	0,89	50,65%
194	2,86	2,33	3,30	0,97	41,43%
030	3,62	3,62	5,00	1,38	38,16%
110	3,20	2,75	3,60	0,85	30,91%
164	3,84	3,63	4,71	1,08	29,88%
220	1,87	1,71	2,14	0,43	25,00%
067	3,22	2,70	3,33	0,63	23,46%
150	3,73	3,62	4,45	0,83	22,88%
245	3,31	3,00	3,67	0,67	22,22%
359	3,57	3,41	4,14	0,74	21,58%
312	3,00	2,77	3,33	0,56	20,37%
249	2,87	2,50	3,00	0,50	20,00%
234	3,77	3,33	4,00	0,67	20,00%
028	3,75	3,33	4,00	0,67	20,00%
...	...	...	...	...	...
272	4,00	4,00	3,00	-1,00	-25,00%
070	2,79	2,89	2,14	-0,75	-25,82%
146	2,71	3,43	2,43	-1,00	-29,17%
155	3,22	3,42	2,33	-1,09	-31,79%
279	2,41	2,22	1,50	-0,72	-32,50%
032	1,78	2,67	1,73	-0,94	-35,23%
244	3,00	3,80	2,33	-1,47	-38,60%
113	2,04	3,00	1,84	-1,16	-38,60%
139	2,61	4,00	2,22	-1,78	-44,57%
280	2,00	2,12	1,00	-1,12	-52,94%
044	2,33	3,00	1,33	-1,67	-55,56%
361	1,75	2,00	0,50	-1,50	-75,00%
283	2,00	2,00	0,00	-2,00	-100,00%
362	2,36	2,59	0,00	-2,59	-100,00%
339	2,33	2,62	0,00	-2,62	-100,00%

- 48 alunos obtiveram uma queda no desempenho maior que 5%;
- 36 alunos ficaram entre 5% de aumento e 5% de queda.

Novamente observamos que o desempenho posterior em programação igual a zero significa que o aluno fez a disciplina no último semestre anterior a esse estudo, então não há disciplinas a serem analisadas.

Com base nos resultados da análise de desempenho dos alunos em disciplinas de programação podemos observar que 36.84% dos alunos apresentaram um aumento superior a 5% no desempenho em disciplinas de programação.

Em contrapartida, ao retirarmos da conta os alunos com desempenho posterior igual a zero (3 alunos), 33.83% dos alunos apresentaram uma queda acima de 5% no desempenho.

Estes dados indicam que o contato com a metodologia de ensino pode ter sido responsável por fixar melhor os conceitos de programação no aprendizado do aluno e aumentado a motivação do aluno em aprender programação. Porém isso não é verdade para todos os alunos. Como o número de alunos com aumento é semelhante ao número de alunos com queda no desempenho, podemos concluir que a inserção dessa metodologia de ensino em disciplinas pontuais não foi suficiente para promover uma mudança na atitude do aluno.

3.2 Métricas de Turma

As turmas escolhidas para serem analisadas foram as turmas de Estruturas de Dados e Algoritmos 2 (103209) e Paradigmas de Programação (203904). Foram escolhidas as turmas onde havia entre os alunos alguns daqueles que cursaram disciplinas com juiz eletrônico anteriormente.

Estas disciplinas foram escolhidas por possuirem em sua ementa tópicos relacionados a programação, que necessitam de conhecimentos prévios providos pelas disciplinas que utilizaram juiz eletrônico como apoio na metodologia de ensino e na avaliação.

As turmas analisadas foram as turmas de Estruturas de Dados e Algoritmos 2 e Paradigmas de Programação do segundo período do ano de 2014.

3.2.1 Desempenho individual dos alunos

Está métrica busca analisar o desempenho individual dos alunos dentro da disciplina. Os alunos foram ordenados da maior para a menor menção. O objetivo era observar se os alunos que tiveram contato com juiz eletrônico estavam entre os alunos de maior menção.

Foram excluídos desta análise os alunos que trancaram a disciplina, aproveitaram a disciplina de outra forma (crédito concedido ou aproveitamento) e os alunos que obtiveram SR como menção. As Tabelas 11 e 12 apresentam as menções finais dos alunos destas disciplinas.

Com base nas menções dos alunos (Tabelas 11 e 12) nas disciplinas é possível observar que os alunos com as maiores menções são, em sua maioria, alunos que tiveram contato anteriormente com juiz eletrônico.

É possível inferir que os fundamentos necessários para o aprendizado de conceitos mais avançados foram melhor assimilados por estes alunos, resultando no desempenho

Tabela 11: Turma de Estruturas de Dados e Algoritmos 2, 2014/2

Aluno (ID)	`Menção`	`Juiz Eletrônico`
150	SS	SIM
271	SS	SIM
282	SS	SIM
006	MS	SIM
014	MS	NÃO
226	MS	SIM
359	MS	SIM
129	MM	SIM
346	MM	NÃO
170	MM	NÃO
199	MM	NÃO
208	MM	NÃO
216	MM	NÃO
223	MM	SIM
231	MM	SIM
232	MM	SIM
049	MI	SIM
077	MI	NÃO
184	MI	NÃO
029	II	SIM
113	II	SIM
262	II	NÃO
383	II	NÃO

elevado.

3.2.2 Média dos alunos que usaram ejudge vs. média dos outros alunos

Esta métrica busca comparar o desempenho médio dos alunos que fizeram disciplinas com juiz eletrônico com o desempenho. As Tabelas 13 e 14 reapresentam os dados anteriores, agora separando os alunos entre os que cursaram e os que não cursaram disciplinas com juiz eletrônico.

Para o cálculo da média foi usado o peso de cada uma das menções (Tabela 7). O peso de cada menção dos alunos foi somada e depois dividida pelo número de alunos. A média das menções será utilizada para comparação, conforme apresentado na Tabela 15.

A média de desempenho dos alunos que tiveram contato com juiz eletrônico apresentaram uma média 32.0% superior a média dos outros alunos na disciplina de Estrutura de Dados e Algoritmos 2 e 10.7% superior a dos outros alunos na disciplina de Paradigmas de Programação.

Como constatado na métrica anterior, o rendimento superior por parte destes

Tabela 12: Turma de Paradigmas de Programação, 2014/2

Aluno (ID)	**Menção**	**Juiz Eletrônico**
040	SS	SIM
228	SS	SIM
250	SS	SIM
041	MS	SIM
166	MS	NÃO
173	MS	NÃO
206	MS	NÃO
232	MS	SIM
004	MM	SIM
022	MM	NÃO
027	MM	NÃO
034	MM	NÃO
048	MM	NÃO
049	MM	SIM
053	MM	SIM
058	MM	NÃO
093	MM	NÃO
107	MM	SIM
109	MM	NÃO
119	MM	NÃO
132	MM	SIM
147	MM	NÃO
153	MM	NÃO
197	MM	SIM
201	MM	SIM
331	MM	NÃO
367	MM	NÃO
009	MI	NÃO
098	MI	NÃO
110	MI	SIM
142	MI	SIM
183	MI	NÃO
005	II	SIM
032	II	SIM
042	II	NÃO
097	II	NÃO
174	II	NÃO
188	II	SIM

alunos pode ser proveniente da melhor assimilação dos fundamentos necessários para o aprendizado de conceitos mais avançados.

Outro fator é a motivação desses alunos. Ao serem apresentados a problemas próximos à realidade para resolver os alunos passam a ver a programação como uma ferramenta

Tabela 13: Médias da turma de Estruturas de Dados e Algoritmos 2, 2014/2

Num. de alunos	**Menção**	**Juiz Eletrônico**
3	SS	SIM
3	MS	SIM
4	MM	SIM
1	MI	SIM
2	II	SIM
1	MS	NÃO
5	MM	NÃO
2	MI	NÃO
2	II	NÃO

Tabela 14: Médias da turma de Paradigmas de Programação, 2014/2

Num. de alunos	**Menção**	**Juiz Eletrônico**
3	SS	SIM
2	MS	SIM
7	MM	SIM
2	MI	SIM
3	II	SIM
3	MS	NÃO
12	MM	NÃO
3	MI	NÃO
3	II	NÃO

Tabela 15: Comparação de Desempenho

Disciplina	**Período**	**Média com juiz**	**Média sem juiz**
Estruturas de Dados e Algoritmos 2	2014/2	3,30	2,50
Paradigmas de Programação	2014/2	3,00	2,71

de trabalho com aplicação real e se tornam mais ávidos em aprender.

Conclusão

A utilização de juízes eletrônicos e problemas oriundos de maratonas de programação, com base nas métricas extraídas do desempenho dos alunos, melhora a formação dos alunos de Engenharia de Software.

Um engenheiro de software deve ser capaz de entender o problema a ser resolvido e os requisitos do problema antes de propor uma solução, habilidade provida pela necessidade de entender o problema a ser resolvido em uma maratona.

Um engenheiro de software deve conhecer os fundamentos básicos da computação para que possa propor e projetar a melhor solução para atender as necessidade dos clientes, habilidade provida pela gama de conhecimentos adiquiridos para ser competitivo em uma maratona de programação.

Um engenheiro de software deve testar o software exaustivamente para assegurar que ele está de acordo com os requisitos do cliente, habilidade desenvolvida pelos competidores de maratonas de programação devido a necessidade de testar seus algoritmos para se certificar de que ele funciona como necessário (HALIM; HALIM, 2013).

Um engenheiro de software deve ser capaz de trabalhar em equipe, pois nem sempre ele trabalha sozinho para solucionar problemas no mercado de trabalho. Como algumas competições são realizadas em equipes os competidores são encorajados a aprender a trabalhar em equipe para terem sucesso.

Entendimento do problema, capacidade de planejar e executar uma estratégia de solução e habilidade para testá-lo exaustivamente, caraterísticas fundamentais ao profissional da área, habilidades essenciais ao bom engenheiro de software, são desenvolvidas por participantes de maratonas de programação (HALIM; HALIM, 2013; PRESSMAN, 2010).

Foi observado que aproximadamente metade dos alunos teve um aumento significativo no rendimento em disciplinas de programação e aproximadamente metade teve uma queda. É possível que o motivo seja a adoção desta metodologia em disciplinas avançadas na grade do curso e em apenas uma disciplina obrigatória. Isto pode acarretar no estranhamento dos alunos com uma metodologia muito diferente das que eles estavam acostumados.

A quebra de paradigma pode ter sido traumática para parte dos alunos. Para evitar este problema é aconselhado o uso desta metodologia em disciplinas introdutórias do curso (CHEANG et al., 2003; ROSENBLOOM, 2009).

3.3 Trabalhos Futuros

Uma das dificuldades enfrentadas no desenvolvimento do estudo foi acessar os dados dos históricos dos alunos e processá-los. Por serem disponibilizados apenas no formato PDF, foi necessário um trabalho de conversão dos arquivos para formato de texto e retirar os dados pessoais dos alunos do corpo dos arquivos.

Caso as informações acerca do desempenho dos alunos fosse acessível na forma de uma base de dados unificada, seria possível estender o alcance do estudo para qualquer área do conhecimento para a avaliação do impacto da adoção de novas metodologias de ensino em qualquer curso, para qualquer disciplina.

Após unificada e disponibilizada a base de dados de histórico dos alunos da universidade será possível criar uma plataforma de consulta onde os docentes poderão aplicar filtros e recuperar métricas e informações a respeito do desempenho dos alunos. Isso possibilita a análise de metodologias com base no desempenho, a identificação de disciplinas com alto indice de reprovação e a análise da origem, se está na metodologia de ensino ou nos alunos.

É proposto como continuação deste trabalho o estudo da possibilidade de unificar a base de dados da universidade acerca do histórico dos alunos, sem o acesso a informações que os identifique. Assim, será possível a implementação de uma aplicação que consulte a base de dados de alunos e, a partir de um filtro aplicado pelo usuário, mostrar gráficos, tabelas e indíces para que seja possível tirar conclusões a partir delas.

Referências

ACM. *The ACM-ICPC International Collegiate Programming Contest*. 2015. Disponível em: <http://icpc.baylor.edu/>. Citado na página 27.

ADOBE.COM. *PDF Reference and Adobe Extensions to the PDF Specification | Adobe Developer Connection*. 2015. Disponível em: <http://www.adobe.com/devnet/pdf/pdf_reference.html>. Nenhuma citação no texto.

AMRAII, S. A. Observations on teamwork strategies in the acm international collegiate programming contest. *Crossroads*, v. 14, n. 1, p. 9, 2007. Nenhuma citação no texto.

AREFIN, A. S. Art of Programming Contest. *C Programming Tutorials, Data Struc-tures and Algorithms. ISBN*, p. 984–32, 2006. Disponível em: <http://files.duohuo.org/Docs/DHDocs/Computer%20Science/Book/%E7%90%86/Art%20of%20Programming%20Contest.pdf>. Nenhuma citação no texto.

CAMPOS, C. P.; FERREIRA, C. E. Boca: um sistema de apoio para competições de programação. *Anais do Congresso da SBC Salvador, BA*, 2004. Citado na página 31.

CHEANG, B. et al. On automated grading of programming assignments in an academic institution. *Computers & Education*, v. 41, n. 2, p. 121–131, set. 2003. ISSN 03601315. Citado 4 vezes nas páginas 24, 25, 31 e 57.

CODE.ORG. *A Hora do Código no Brasil está chegando*. 2015. Disponível em: <http://br.code.org/about>. Nenhuma citação no texto.

CORMEN, T. H. (Ed.). *Introduction to algorithms*. 3rd ed. ed. Cambridge, Mass: MIT Press, 2009. ISBN 9780262033848 0262033844 9780262533058 0262533057. Citado na página 33.

FASSBINDER, A. G. de O.; PAULA, L. C. de; ARAUJO, J. C. D. Experiências no estímulo à prática de programação através do desenvolvimento de atividades extracurriculares relacionadas com as competições de conhecimentos. In: *Congresso da Sociedade Brasileira de Computação (CSBC)*. [S.l.: s.n.], 2012. Citado 2 vezes nas páginas 24 e 25.

FITZGERALD, S.; HINES, M. L. The computer science fair: an alternative to the computer programming contest. *ACM SIGCSE Bulletin*, v. 28, n. 1, p. 368–372, 1996. Nenhuma citação no texto.

HALIM, S.; HALIM, F. *Competitive Programming 3*. [S.l.]: Lulu.com, 2013. Citado 6 vezes nas páginas 25, 28, 29, 32, 33 e 57.

HALIM, S. et al. Learning Algorithms with Unified and Interactive Web-Based Visualization. *Olympiads in Informatics*, v. 6, p. 53–68, 2012. Disponível em: <http://www.mii.lt/olympiads_in_informatics/pdf/INFOL099.pdf>. Nenhuma citação no texto.

ICPC.BAYLOR.EDU. *World Finals Rules*. 2015. Disponível em: <http://icpc.baylor.edu/worldfinals/rules>. Nenhuma citação no texto.

INEP. *Censo da Educação do Ensino Superior*. 2007. Citado na página 24.

KHERA, V.; ASTRACHAN, O.; KOTZ, D. The internet programming contest: a report and philosophy. *ACM SIGCSE Bulletin*, v. 25, n. 1, p. 48–52, 1993. Nenhuma citação no texto.

MANZOOR, S. Analyzing programming contest statistics. *Perspectives on Computer Science Competitions for (High School) Students*, 2006. Nenhuma citação no texto.

MARATONA.IME.USP.BR. *XIX Maratona de Programação*. 2015. Disponível em: <http://maratona.ime.usp.br/regras14.html>. Nenhuma citação no texto.

MARATONA.IME.USP.BR. *XX Maratona de Programação*. 2015. Disponível em: <http://maratona.ime.usp.br/>. Nenhuma citação no texto.

MATA, E. C. da et al. Proposta de sistema lúdico para ensino de programação a alunos do ensino médio. In: *Anais do X Congresso Brasileiro de Ensino Superior a Distância, Belém/PA*. [s.n.], 2013. Disponível em: <http://www.aedi.ufpa.br/esud/trabalhos/poster/AT4/114422.pdf>. Nenhuma citação no texto.

POSNER, E. A.; SPIER, K. E.; VERMEULE, A. Divide and conquer. *Journal of Legal Analysis*, v. 2, n. 2, p. 417–471, 2010. Disponível em: <http://jla.oxfordjournals.org/content/2/2/417.abstract>. Nenhuma citação no texto.

POUCHER, B. Giving students the competitive edge. *Communications of the ACM*, v. 55, n. 8, p. 5–5, ago. 2012. ISSN 00010782. Nenhuma citação no texto.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. 7th ed. ed. New York: McGraw-Hill Higher Education, 2010. ISBN 9780073375977 0073375977. Citado 3 vezes nas páginas 23, 25 e 57.

QUADROS, J. R. d. T. et al. Jogos e Aplicativos, Trabalhos de Conclusão de Curso e Competições em TI como Ferramentas de Desenvolvimento Educacional. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [s.n.], 2014. v. 3. Disponível em: <http://ceie-sbc.tempsite.ws/pub/index.php/wcbie/article/view/3206>. Nenhuma citação no texto.

ROSENBLOOM, A. Running a programming contest in an introductory computer science course. In: *ACM SIGCSE Bulletin*. [S.l.]: ACM, 2009. v. 41, p. 347–347. Citado 2 vezes nas páginas 24 e 57.

SANTOS, J. C.; RIBEIRO, A. R. Jonline: proposta preliminar de um juiz online didático para o ensino de programação. *Simpósio Brasileiro de Informática na Educação (SBIE 2011)*, v. 22, 2011. Nenhuma citação no texto.

SEDGEWICK, R. *Algorithms in C*. Reading, Mass: Addison-Wesley Pub. Co, 1990. (Addison-Wesley series in computer science). ISBN 0201514257. Citado na página 33.

SHILOV, N. V.; YI, K. Engaging students with theory through acm collegiate programming contest. *Communications of the ACM*, v. 45, n. 9, p. 98–101, 2002. Nenhuma citação no texto.

SKIENA, S. S. *The algorithm design manual*. [S.l.]: Springer, 2008. Citado 2 vezes nas páginas 24 e 31.

SKIENA, S. S.; REVILLA, M. A. *Programming challenges: the programming contest training manual*. New York: Springer, 2003. (Texts in computer science). ISBN 0387001638. Citado 2 vezes nas páginas 28 e 33.

SOMMERVILLE, I. *Software engineering*. 9th ed. ed. Boston: Pearson, 2011. ISBN 9780137035151 0137035152 0137053460 9780137053469. Citado 2 vezes nas páginas 23 e 25.

WIKIPEDIA, A. I. A. *ACM International Collegiate Programming Contest*. 2015. Disponível em: <http://en.wikipedia.org/wiki/ACM_International_Collegiate_Programming_Contest>. Nenhuma citação no texto.

WIKIPEDIA, J. A. *JSON*. 2015. Disponível em: <https://en.wikipedia.org/wiki/JSON>. Nenhuma citação no texto.

WIKIPEDIA, P. A. *Portable Document Format*. 2015. Disponível em: <https://en.wikipedia.org/?title=Portable_Document_Format>. Nenhuma citação no texto.

Apêndices

APÊNDICE A – Exemplo de um problema nos moldes de um problema de maratona de programação

APÊNDICE B – Regulamento da Maratona de Programação - ACM ICPC

APÊNDICE C – Matriz curricular do curso de Engenharia de Software

Anexos

ANEXO A – Código de criação da base de históricos

Código A.1: Script python da criação da base de dados

```
1  # -*- coding: utf-8 -*-
2
3  import json
4  from sets import Set
5
6  # Todos os codigos de mencao possiveis
7  mencoes_legenda = Set(['SR','II','MI','MM','MS','SS','TR','CC','AP','DP'
       ,'TJ'])
8  creditos_legenda = Set(['000','002','003','004','005','006','008','014',
       '015','020'])
9
10 # Pesos de cada mencao no calculo do rendimento do aluno
11 pesos = {
12     'SR':  0,
13     'II':  1,
14     'MI':  2,
15     'MM':  3,
16     'MS':  4,
17     'SS':  5,
18     'TR': -1,
19     'CC': -1,
20     'AP': -1,
21     'DP': -1,
22     'TJ': -1
23 }
24
25 # Codigo das disciplinas de programacao do curso
26 disciplinasProgramacao = Set([
27         "113913",
28         "116301",
29         "195405",
30         "206199",
31         "110141",
32         "195413",
33         "206601",
34         "101095",
35         "208507",
36         "103209",
37         "201286",
```

```
38          "206181",
39          "195341",
40          "193704",
41          "203904",
42          "107409",
43          "107417",
44          "206598",
45          "193640",
46          "201294",
47          "208493",
48          "103195",
49          "117552",
50          "203882",
51          "208701"
52      ])
53
54  # Turmas que usaram eJudge como auxilio a metodologia de ensino
55  disciplinas_ejudge = [
56      {"codigo":"193704","ano":"2012","periodo":"2"},
57      {"codigo":"193704","ano":"2013","periodo":"1"},
58      {"codigo":"208493","ano":"2012","periodo":"2"},
59      {"codigo":"208493","ano":"2013","periodo":"1"},
60      {"codigo":"208493","ano":"2013","periodo":"2"},
61      {"codigo":"103195","ano":"2013","periodo":"1"},
62      {"codigo":"103195","ano":"2013","periodo":"2"},
63      {"codigo":"103195","ano":"2014","periodo":"1"},
64      {"codigo":"103195","ano":"2014","periodo":"2"}
65  ]
66  # Metodos auxiliares
67  ### Confere se e ou nao um codigo de disciplina
68  def isCodigoDeDisciplina (codigo):
69      return codigo.isdigit() and len(codigo) == 6
70
71  ### Confere se e ou nao uma mencao
72  def isMencao (mencao):
73      return mencao in mencoes_legenda
74
75  ### Confere se e um numero de creditos
76  def isCredito (credito):
77      return credito in creditos_legenda
78
79  ### Calcula o Indice de rendimento de um aluno
80  def calcularIRA(aluno):
81      ira = 0.0
82      num_disciplinas = 0
83      for semestre in aluno["historico"]:
84          for disciplina in semestre["disciplinas"]:
```

```
85              if pesos[disciplina["mencao"]] != -1:
86                  ira += pesos[disciplina["mencao"]] * int(disciplina["
                        credito"])
87                  num_disciplinas += int(disciplina["credito"])
88
89      return ira / num_disciplinas
90
91  ### Calcula o desempenho de um aluno com base nas disciplinas passadas
        por parametro
92  def calcularIRASeletivo(aluno,disciplinas):
93      ira = 0.0
94      num_disciplinas = 0
95      for semestre in aluno["historico"]:
96          for disciplina in semestre["disciplinas"]:
97              if pesos[disciplina["mencao"]] != -1 and disciplina["codigo"
                    ] in disciplinas:
98                  ira += pesos[disciplina["mencao"]] * int(disciplina["
                        credito"])
99                  num_disciplinas += int(disciplina["credito"])
100     if num_disciplinas == 0:
101         return 0
102
103     return ira / num_disciplinas
104
105 ### Calcula o desempenho do aluno antes dele fazer uma materia com
        ejudge
106 def calcularIRAAntesEjudge(aluno):
107     ira = 0.0
108     num_disciplinas = 0
109     turma = aluno["ejudge_turma"]
110     for semestre in aluno["historico"]:
111         if semestre["ano"] <= turma["ano"] and semestre["periodo"] <=
                turma["periodo"]:
112             for disciplina in semestre["disciplinas"]:
113                 if pesos[disciplina["mencao"]] != -1:
114                     ira += pesos[disciplina["mencao"]]  * int(disciplina
                            ["credito"])
115                     num_disciplinas += int(disciplina["credito"])
116     if num_disciplinas == 0:
117         return 0.0
118     return ira / num_disciplinas
119
120 ### Calcula o desempenho do aluno depois dele fazer uma materia com
        ejudge
121 def calcularIRADepoisEjudge(aluno):
122     ira = 0.0
123     num_disciplinas = 0
```

```
124     turma = aluno["ejudge_turma"]
125     for semestre in aluno["historico"]:
126         if semestre["ano"] >= turma["ano"] and semestre["periodo"] >=
               turma["periodo"]:
127             for disciplina in semestre["disciplinas"]:
128                 if pesos[disciplina["mencao"]] != -1:
129                     ira += pesos[disciplina["mencao"]] * int(disciplina[
                           "credito"])
130                     num_disciplinas += int(disciplina["credito"])
131     if num_disciplinas == 0:
132         return 0.0
133     return ira / num_disciplinas
134
135 ### Calcula o desempenho do aluno antes dele fazer uma materia com
       ejudge
136 def calcularIRASeletivoAntesEjudge(aluno,disciplinas):
137     ira = 0.0
138     num_disciplinas = 0
139     turma = aluno["ejudge_turma"]
140     for semestre in aluno["historico"]:
141         if semestre["ano"] <= turma["ano"] and semestre["periodo"] <=
               turma["periodo"]:
142             for disciplina in semestre["disciplinas"]:
143                 if pesos[disciplina["mencao"]] != -1 and disciplina["
                       codigo"] in disciplinas:
144                     ira += pesos[disciplina["mencao"]]  * int(disciplina
                           ["credito"])
145                     num_disciplinas += int(disciplina["credito"])
146     if num_disciplinas == 0:
147         return 0.0
148     return ira / num_disciplinas
149
150 ### Calcula o desempenho do aluno depois dele fazer uma materia com
       ejudge
151 def calcularIRASeletivoDepoisEjudge(aluno,disciplinas):
152     ira = 0.0
153     num_disciplinas = 0
154     turma = aluno["ejudge_turma"]
155     for semestre in aluno["historico"]:
156         if semestre["ano"] >= turma["ano"] and semestre["periodo"] >=
               turma["periodo"]:
157             for disciplina in semestre["disciplinas"]:
158                 if disciplina["codigo"] in disciplinasProgramacao and
                       pesos[disciplina["mencao"]] != -1:
159                     ira += pesos[disciplina["mencao"]] * int(disciplina[
                           "credito"])
160                     num_disciplinas += int(disciplina["credito"])
```

```
161     if num_disciplinas == 0:
162         return 0.0
163     return ira / num_disciplinas
164 
165 ### Conta quantas materias com os codigos passados um aluno ja fez, com
        repeticoes
166 def contMaterias(aluno, codigos):
167     cont = 0
168 
169     for semestre in aluno["historico"]:
170         for disciplina in semestre["disciplinas"]:
171             if disciplina["codigo"] in codigos:
172                 cont += 1
173 
174     return cont
175 
176 ### Aluno cursou nas turmas?
177 def isTurma(aluno, disciplinas):
178     for semestre in aluno["historico"]:
179         for disciplina in semestre["disciplinas"]:
180             for e in disciplinas:
181                 if e["ano"] == semestre["ano"] and e["periodo"] ==
                        semestre["periodo"]:
182                     if e["codigo"] == disciplina["codigo"]:
183                         return True
184 
185     return False
186 
187 ### Semestre ejudge
188 def getTurmaEjudge(aluno, disciplinas):
189     for semestre in aluno["historico"]:
190         for disciplina in semestre["disciplinas"]:
191             for e in disciplinas:
192                 if e["ano"] == semestre["ano"] and e["periodo"] ==
                        semestre["periodo"]:
193                     if e["codigo"] == disciplina["codigo"]:
194                         return {
195                             "ano": e["ano"],
196                             "periodo": e["periodo"],
197                             "codigo": disciplina["codigo"]
198                         }
199 
200 
201 ####### Processamento dos Arquivos #######
202 # Array para armazenar todos os alunos
203 alunos = []
204 
```

```
205 # Passar por todos os arquivos da PRIMEIRA leva de dados
206 print 'Lendo arquivos de da pasta "./data_1" '
207 for i in range(388):
208     # Abre o arquivo
209     filename = "./data/data_1/" + str(i+1) + ".data"
210     file = open(filename, 'r')
211 
212     # Separa cada linha do arquivo em um array de strings
213     content = file.readlines()
214 
215     # Cria um aluno
216     aluno = {}
217 
218     # Ingresso na UnB
219     for line in content:
220         if 'Ingresso na UnB:' in line:
221             line = line.replace('Ingresso na UnB: ', '')
222             aluno['ano_ingresso'], aluno['periodo_ingresso'] = line.
                    split('/')
223             aluno['periodo_ingresso'] = aluno['periodo_ingresso'][:-1]
224             break
225 
226     # StudentID
227     for line in content:
228         if 'StudentID:' in line:
229             line = line.replace('StudentID: ','')
230             aluno['id'] = int(line[:-1])
231             break;
232 
233     # Data de Nascimento
234     for line in content:
235         if 'Nascimento:' in line:
236             line = line.replace('Nascimento: ', '')
237             aluno['data_nascimento'] = line[:-1]
238 
239     # Semestres
240     periodos = []
241     codigos_disciplinas = []
242     mencoes = []
243     creditos = []
244 
245     # Codigos de Disciplina
246     for line in content:
247         words = line.split(' ')
248         if len(words) > 1:
249             if isCodigoDeDisciplina(words[1][:-1]):
250                 codigos_disciplinas.append(words[1][:-1])
```

```
251         else:
252             if isCodigoDeDisciplina(words[0][:-1]):
253                 codigos_disciplinas.append(words[0][:-1])
254 
255     # Mencoes
256     for line in content:
257         if isMencao(line[:-1]):
258             mencoes.append(line[:-1])
259 
260     # Creditos
261     for line in content:
262         if isCredito(line[:-1]):
263             creditos.append(line[:-1])
264 
265     # Periodos
266     for line in content:
267         if 'Periodo:' in line:
268             periodo = {}
269             line = line.replace('Periodo: ', '')
270             periodo['ano'], periodo['periodo'] = line.split('/')
271             periodo['periodo'] = periodo['periodo'][:-1]
272             periodo['num_disciplinas'] = 0
273             if '(Continuacao)' in line:
274                 pass
275             else:
276                 # print line[:-1]
277                 periodos.append(periodo)
278 
279     # Contagem de disciplinas por periodo
280     periodo_atual = 0
281     cont = 0
282     for line in content:
283         if 'Periodo:' in line:
284             if '(Continuacao)' in line:
285                 pass
286             else:
287                 # Atribuir o cont pro periodo
288                 if periodo_atual > 0:
289                     periodos[periodo_atual - 1]['num_disciplinas'] =
                            cont
290                 # Atualizar o periodo que estou contando
291                 periodo_atual += 1
292                 # Zerar o contador
293                 cont = 0
294         else:
295             words = line.split(' ')
296             if len(words) > 1:
```

```
297                 if isCodigoDeDisciplina(words[1][:-1]):
298                     cont += 1
299             else:
300                 if isCodigoDeDisciplina(words[0][:-1]):
301                     cont += 1
302
303     periodos[periodo_atual-1]['num_disciplinas'] = cont
304 #     for d, m, c in zip(codigos_disciplinas, mencoes, creditos):
305 #         print '{} {} {}'.format(d,m,c)
306
307     # Montar Historico
308     historico = []
309     num = 1
310     for periodo in periodos:
311         semestre = {}
312         semestre['numero'] = num
313         num += 1
314         semestre['ano'] = periodo['ano']
315         semestre['periodo'] = periodo['periodo']
316         semestre['disciplinas'] = []
317         for i in range(periodo['num_disciplinas']):
318             disciplina = {}
319             disciplina['codigo'] = codigos_disciplinas.pop(0)
320             disciplina['mencao'] = mencoes.pop(0)
321             disciplina['credito'] = creditos.pop(0)
322             semestre['disciplinas'].append(disciplina)
323
324         historico.append(semestre)
325
326     aluno['historico'] = historico
327
328     #IRA
329     aluno["desempenho"] = calcularIRA(aluno)
330
331     #IRA Prog
332     aluno["desempenho_programacao"] = calcularIRASeletivo(aluno,
            disciplinasProgramacao)
333
334     #Quantidade de Materias de Programacao
335     aluno["num_disciplinas_programacao"] = contMaterias(aluno,
            disciplinasProgramacao)
336
337     #Fez materia com juiz eletronico?
338     if isTurma(aluno, disciplinas_ejudge):
339         aluno["ejudge"] = True
340         #Qual semestre que usou ejudge?
```

```
341         aluno["ejudge_turma"] = getTurmaEjudge(aluno, disciplinas_ejudge
                )
342
343         #IRA Antes do ejudge
344         aluno["desempenho_antes_ejudge"] = calcularIRAAntesEjudge(aluno)
345         #IRA depois do ejduge
346         aluno["desempenho_depois_ejudge"] = calcularIRADepoisEjudge(
                aluno)
347
348         #IRA Programacao Antes do ejudge
349         aluno["desempenho_programacao_antes_ejudge"] =
                calcularIRASeletivoAntesEjudge(aluno, disciplinasProgramacao)
350         #IRA Programacao Depois do ejduge
351         aluno["desempenho_programacao_depois_ejudge"] =
                calcularIRASeletivoDepoisEjudge(aluno, disciplinasProgramacao
                )
352
353     else:
354         aluno["ejudge"] = False
355
356     # Adiciona o aluno na lista de alunos
357     alunos.append(aluno)
358
359 print 'Dados lidos com sucesso!'
360
361 # Cria o JSON com os alunos
362 print 'Criando JSON com historicos dos alunos'
363 historicos = open('./json/historicos.json','w')
364 historicos.write(json.dumps(alunos, sort_keys=False, indent=4,
        separators=(',',': ')))
365
366 # INDEXADO
367 db_indexado = {}
368
369 for aluno in alunos:
370     db_indexado[aluno["id"]] = aluno
371
372 # Cria o JSON indexado com os alunos
373 print 'Criando JSON com historicos dos alunos indexado'
374 historicos_indexado = open('./json/historicos_indexado.json','w')
375 historicos_indexado.write(json.dumps(db_indexado, sort_keys=False,
        indent=4, separators=(',',': ')))
```

ANEXO B – Exemplo de um objeto aluno

Código B.1: Exemplo de um objeto JSON de um aluno

```
"3": {
        "num_disciplinas_programacao": 13,
        "periodo_ingresso": "1",
        "desempenho_programacao_depois_ejudge": 3.125,
        "ejudge": true,
        "desempenho_depois_ejudge": 3.4473684210526314,
        "desempenho_programacao_antes_ejudge": 3.6,
        "desempenho_antes_ejudge": 3.0277777777777777,
        "data_nascimento": "28/06/1993",
        "desempenho": 3.1204819277108435,
        "historico": [
            {
                "periodo": "1",
                "disciplinas": [
                    {
                        "mencao": "MI",
                        "codigo": "113034",
                        "credito": "006"
                    },
                    {
                        "mencao": "MM",
                        "codigo": "113093",
                        "credito": "004"
                    },
                    {
                        "mencao": "MM",
                        "codigo": "113913",
                        "credito": "004"
                    },
                    {
                        "mencao": "MM",
                        "codigo": "114626",
                        "credito": "004"
                    },
                    {
                        "mencao": "MM",
                        "codigo": "114634",
                        "credito": "002"
                    },
                    {
                        "mencao": "MM",
```

```
                        "codigo": "198005",
                        "credito": "004"
                    },
                    {
                        "mencao": "MS",
                        "codigo": "198013",
                        "credito": "004"
                    }
                ],
                "ano": "2011",
                "numero": 1
            },
        ],
        "ejudge_turma": {
            "codigo": "193704",
            "periodo": "1",
            "ano": "2013"
        },
        "ano_ingresso": "2011",
        "id": 3,
        "desempenho_programacao": 3.1818181818181817
    }
```

ANEXO C – Código de processamento dos alunos

Código C.1: Script python para extrair métricas da base de dados dos históricos

```
1  # -*- coding: utf-8 -*-
2
3  import json
4  from sets import Set
5  from pprint import pprint
6
7  # Abrindo o arquivo de alunos
8  alunos_file_path = "./json/historicos.json"
9  with open(alunos_file_path) as data_file:
10     historicos = json.load(data_file)
11
12 # Objeto que ira armazenar os resultados dos processamentos
13 data = {}
14
15 # Numero de alunos
16 data["num_alunos"] = len(historicos);
17
18 # Quantas materias de programacao feitas
19 # IRA
20 # IRA Programacao
21 # Desempenho antes do juiz
22 # Desempenho apos o juiz
23
24 # Todos os alunos que tiveram contato com ejudge
25 alunos_ejudge = []
26 for aluno in historicos:
27     if aluno["ejudge"]:
28         alunos_ejudge.append(aluno)
29
30 num_aluno_melhorou = 0
31 alunos_melhoraram = []
32 num_aluno_piorou = 0
33 alunos_pioraram = []
34 num_aluno_manteve = 0
35 alunos_mantiveram = []
36
37 for aluno in alunos_ejudge:
38     if float(aluno["desempenho_antes_ejudge"]) < float(aluno["desempenho
       "]):
```

```
39          num_aluno_melhorou += 1
40          alunos_melhoraram.append(aluno["id"])
41      elif float(aluno["desempenho_antes_ejudge"]) > float(aluno["
            desempenho"]):
42          num_aluno_piorou += 1
43          alunos_pioraram.append(aluno["id"])
44      else:
45          num_aluno_manteve += 1
46          alunos_mantiveram.append(aluno["id"])
47
48  print '+',num_aluno_melhorou,'\n-', num_aluno_piorou,'\n=',
        num_aluno_manteve
49
50  data["num_aluno_melhorou"] = num_aluno_melhorou
51  data["alunos_melhoraram"] = alunos_melhoraram
52  data["num_aluno_piorou"] = num_aluno_piorou
53  data["alunos_pioraram"] = alunos_pioraram
54  data["num_aluno_manteve"] = num_aluno_manteve
55  data["alunos_mantiveram"] = alunos_mantiveram
56
57  # Maior IRA
58  # Menor IRA
59  max_ira = 0
60  min_ira = 9999
61
62  # Maior IRA Programacao
63  # Menor IRA Programacao
64  max_ira_prog = 0
65  min_ira_prog = 9999
66
67  # Media IRA
68  # Media IRA Programacao
69  sum_ira = 0
70  media_ira = 0
71  sum_ira_prog = 0
72  media_ira_prog = 0
73
74  for aluno in historicos:
75      if aluno["desempenho"] > max_ira:
76          max_ira = aluno["desempenho"]
77
78      if aluno["desempenho"] < min_ira:
79          min_ira = aluno["desempenho"]
80
81      if aluno["desempenho_programacao"] > max_ira_prog:
82          max_ira_prog = aluno["desempenho_programacao"]
83
```

```
84      if aluno["desempenho_programacao"] < min_ira_prog:
85          min_ira_prog = aluno["desempenho_programacao"]
86
87      sum_ira += aluno["desempenho"]
88      sum_ira_prog += aluno["desempenho_programacao"]
89
90  media_ira = sum_ira / data["num_alunos"]
91  media_ira_prog = sum_ira_prog / data["num_alunos"]
92
93  data["max_ira"] = max_ira
94  data["max_ira_prog"] = max_ira_prog
95  data["min_ira"] = min_ira
96  data["min_ira_prog"] = min_ira_prog
97  data["media_ira"] = media_ira
98  data["media_ira_prog"] = media_ira_prog
99
100 # Top 25 alunos IRA
101 # IRA
102 # IRA Prog
103 # IRA Depois = IRA
104 # IRA Antes de Ejudge
105
106 #pprint(data)
107 print(data)
108
109 alunosDataJSON = open('./json/alunos_data.json','w')
110 alunosDataJSON.write(json.dumps(data, sort_keys=False, indent=4,
        separators=(',',': ')))
```

ANEXO D – Dados processados de todos os alunos que tiveram contato com juiz eletrônico

Tabela 16: Desempenho dos alunos - parte 1

Aluno (ID)	D	IDA	IDP	δ	Δ
141	4.62	4.64	4.5	-0.14	-2.94%
259	4.6	4.58	4.75	0.17	3.64%
384	4.54	4.52	4.57	0.05	1.15%
345	4.47	4.38	4.83	0.46	10.44%
145	4.34	4.11	4.63	0.52	12.72%
181	4.33	4.24	4.55	0.31	7.2%
271	4.25	4.16	4.69	0.53	12.8%
386	4.16	4.0	4.72	0.72	18.0%
115	4.14	3.82	4.51	0.7	18.27%
25	4.13	3.71	4.44	0.74	19.85%
4	4.13	4.09	4.06	-0.02	-0.57%
232	4.09	4.09	3.83	-0.26	-6.27%
24	4.08	4.1	4.17	0.07	1.63%
376	4.05	3.85	4.11	0.25	6.59%
282	4.04	3.85	4.13	0.28	7.26%
164	4.02	3.92	4.35	0.43	11.04%
284	3.98	3.85	4.38	0.53	13.83%
89	3.98	3.96	3.67	-0.29	-7.41%
37	3.97	4.12	3.44	-0.68	-16.56%
192	3.9	3.91	3.68	-0.22	-5.75%
330	3.89	3.78	4.44	0.67	17.71%
272	3.89	3.83	4.0	0.17	4.31%
156	3.87	3.78	3.9	0.12	3.19%
359	3.86	3.63	4.45	0.82	22.49%
152	3.86	3.8	4.1	0.29	7.69%
179	3.84	3.6	4.41	0.81	22.61%
107	3.84	3.86	3.98	0.12	2.99%
246	3.79	3.73	4.05	0.33	8.8%
317	3.76	3.61	3.83	0.22	6.13%
226	3.74	3.62	3.9	0.27	7.58%
370	3.72	3.61	4.19	0.58	16.07%
212	3.72	3.61	3.53	-0.08	-2.09%
234	3.68	3.42	4.16	0.73	21.38%
228	3.66	3.22	3.85	0.63	19.57%
323	3.66	3.49	3.84	0.35	9.89%
129	3.64	3.58	3.98	0.39	11.01%
324	3.59	3.64	3.83	0.2	5.37%
40	3.59	3.39	3.75	0.36	10.53%
310	3.58	3.36	4.0	0.64	18.88%
243	3.57	3.45	3.59	0.14	4.01%
288	3.56	3.59	3.52	-0.07	-1.82%
6	3.53	3.55	3.72	0.17	4.87%
71	3.52	3.42	3.74	0.32	9.34%
165	3.47	3.47	4.11	0.64	18.35%
265	3.47	3.61	3.86	0.26	7.17%
201	3.45	3.23	3.55	0.32	9.77%
8	3.45	3.57	3.67	0.1	2.8%

Tabela 17: Desempenho dos alunos - parte 2

Aluno (ID)	D	IDA	IDP	δ	Δ
231	3.44	3.23	3.71	0.48	14.86%
312	3.44	3.12	4.04	0.92	29.52%
150	3.44	3.53	4.19	0.66	18.65%
177	3.43	3.33	3.73	0.4	11.92%
28	3.43	3.27	4.14	0.87	26.66%
320	3.42	3.5	4.05	0.55	15.83%
245	3.41	3.19	3.98	0.79	24.79%
257	3.39	3.39	3.67	0.28	8.12%
168	3.35	3.28	2.9	-0.38	-11.67%
316	3.34	3.09	4.04	0.94	30.47%
221	3.34	3.05	4.34	1.3	42.56%
133	3.33	3.47	3.17	-0.3	-8.64%
378	3.32	2.97	3.73	0.76	25.58%
211	3.32	3.53	3.78	0.25	6.97%
162	3.31	3.48	3.36	-0.11	-3.3%
31	3.29	3.09	3.67	0.58	18.83%
41	3.28	2.7	3.57	0.87	32.03%
344	3.26	3.3	3.54	0.24	7.23%
33	3.26	2.96	3.53	0.56	19.07%
197	3.25	3.38	3.39	0.01	0.34%
194	3.25	3.0	3.45	0.45	15.03%
44	3.24	3.24	2.21	-1.03	-31.73%
227	3.24	2.77	3.58	0.8	28.96%
175	3.23	2.93	3.59	0.67	22.82%
218	3.23	3.09	3.54	0.45	14.61%
124	3.22	3.09	3.27	0.18	5.79%
30	3.19	3.19	3.4	0.21	6.54%
81	3.19	3.07	3.2	0.13	4.1%
67	3.17	3.01	3.3	0.29	9.48%
36	3.16	3.19	3.33	0.14	4.38%
17	3.16	3.36	2.65	-0.71	-21.15%
142	3.15	3.29	3.41	0.12	3.65%
10	3.13	3.0	2.5	-0.5	-16.67%
103	3.13	3.39	3.09	-0.3	-8.94%
132	3.12	2.64	3.46	0.82	31.13%
3	3.12	3.03	3.45	0.42	13.86%
251	3.11	2.72	3.67	0.95	35.1%
49	3.1	3.46	3.46	-0.0	-0.02%
244	3.1	3.41	3.08	-0.33	-9.7%
373	3.08	2.95	3.25	0.3	10.2%
223	3.07	3.07	3.0	-0.07	-2.25%
155	3.06	2.91	3.29	0.38	13.22%
66	3.05	2.92	3.12	0.2	6.96%
78	3.05	2.7	2.82	0.13	4.67%
151	3.04	3.07	3.17	0.1	3.26%
250	3.03	2.95	2.8	-0.15	-5.23%
21	3.03	3.06	3.46	0.4	13.17%

Tabela 18: Desempenho dos alunos - parte 3

Aluno (ID)	D	IDA	IDP	δ	Δ
11	3.02	3.0	3.31	0.31	10.26%
137	3.01	3.03	3.26	0.23	7.52%
157	2.97	3.07	3.06	-0.01	-0.36%
20	2.95	2.81	3.79	0.98	34.79%
247	2.95	2.88	2.69	-0.19	-6.45%
278	2.94	3.19	2.8	-0.39	-12.32%
146	2.91	3.02	2.88	-0.13	-4.47%
280	2.88	2.78	2.78	-0.0	-0.17%
255	2.86	3.1	3.14	0.04	1.16%
139	2.84	3.68	2.61	-1.06	-28.93%
53	2.82	2.74	2.8	0.06	2.12%
230	2.82	2.89	2.93	0.04	1.31%
279	2.8	2.67	2.52	-0.15	-5.65%
256	2.8	2.64	3.29	0.65	24.69%
339	2.67	2.69	2.2	-0.49	-18.32%
249	2.66	2.47	2.83	0.37	14.94%
362	2.63	2.89	1.88	-1.01	-34.88%
5	2.61	2.37	3.23	0.87	36.55%
105	2.58	2.7	2.83	0.13	4.71%
220	2.5	2.47	2.57	0.1	4.08%
299	2.5	2.1	1.0	-1.1	-52.38%
188	2.48	2.42	2.7	0.28	11.54%
39	2.48	2.31	2.67	0.36	15.59%
85	2.43	2.18	2.38	0.2	9.01%
236	2.43	2.43	3.11	0.68	28.1%
55	2.42	2.53	2.32	-0.21	-8.28%
113	2.42	3.11	2.24	-0.87	-28.1%
110	2.41	2.61	2.79	0.19	7.18%
70	2.39	2.56	2.17	-0.38	-14.93%
2	2.32	1.82	2.45	0.63	34.42%
32	2.19	2.55	2.02	-0.52	-20.49%
29	2.18	2.08	2.59	0.51	24.37%
187	2.18	2.37	1.13	-1.23	-52.2%
361	2.11	2.56	1.18	-1.39	-54.09%
54	2.09	2.88	1.89	-0.99	-34.26%
372	2.06	2.27	2.0	-0.27	-12.0%
283	2.04	2.04	0.0	-2.04	-100.0%
343	1.96	2.37	1.4	-0.97	-40.89%
379	1.81	2.0	0.89	-1.11	-55.56%

Tabela 19: Desempenho dos alunos em programação - parte 1

Aluno (ID)	D	IDA_P	IDP_P	δ	Δ
141	4.7	4.6	4.4	-0.2	-4.35%
384	4.62	4.62	4.4	-0.22	-4.78%
259	4.61	4.61	4.7	0.09	2.04%
271	4.58	4.33	4.85	0.51	11.83%
345	4.57	4.57	4.5	-0.07	-1.56%
152	4.49	4.33	4.87	0.53	12.31%
115	4.42	4.25	4.33	0.08	1.96%
25	4.4	4.0	4.4	0.4	10.0%
145	4.4	4.29	4.17	-0.12	-2.78%
181	4.36	4.0	4.4	0.4	10.0%
4	4.29	4.0	4.3	0.3	7.5%
107	4.21	4.0	4.09	0.09	2.27%
386	4.2	4.2	4.0	-0.2	-4.76%
221	4.18	4.0	4.25	0.25	6.25%
89	4.12	4.4	5.0	0.6	13.64%
284	4.12	4.09	4.0	-0.09	-2.22%
192	4.0	4.0	3.6	-0.4	-10.0%
272	4.0	4.0	3.0	-1.0	-25.0%
165	3.92	3.92	4.0	0.08	2.13%
232	3.91	3.91	3.5	-0.41	-10.47%
223	3.89	3.89	3.0	-0.89	-22.86%
24	3.88	3.83	3.5	-0.33	-8.7%
179	3.87	3.75	4.4	0.65	17.33%
168	3.85	3.76	3.5	-0.26	-7.03%
164	3.84	3.63	4.71	1.08	29.88%
330	3.84	3.76	3.0	-0.76	-20.25%
376	3.83	3.78	3.8	0.02	0.59%
282	3.83	3.69	3.29	-0.41	-11.01%
226	3.83	3.57	4.11	0.54	15.11%
211	3.79	3.83	3.67	-0.17	-4.35%
246	3.77	3.74	4.0	0.26	6.93%
234	3.77	3.33	4.0	0.67	20.0%
71	3.77	3.5	3.57	0.07	2.04%
28	3.75	3.33	4.0	0.67	20.0%
150	3.73	3.62	4.45	0.83	22.88%
317	3.73	3.46	3.75	0.29	8.33%
231	3.72	3.42	3.5	0.08	2.44%
156	3.71	3.5	3.8	0.3	8.57%
6	3.7	3.73	4.2	0.47	12.68%
288	3.69	3.54	4.2	0.66	18.7%
20	3.67	3.83	3.29	-0.55	-14.29%
324	3.63	3.96	3.92	-0.03	-0.85%
30	3.62	3.62	5.0	1.38	38.16%
227	3.62	3.0	3.29	0.29	9.52%
228	3.62	3.0	2.5	-0.5	-16.67%
359	3.57	3.41	4.14	0.74	21.58%
257	3.56	3.56	4.0	0.44	12.5%

Tabela 20: Desempenho dos alunos em programação - parte 2

Aluno (ID)	D	IDA_P	IDP_P	δ	Δ
316	3.55	3.52	4.0	0.48	13.64%
40	3.54	3.33	3.2	-0.13	-4.0%
37	3.54	3.67	3.33	-0.33	-9.09%
310	3.53	3.39	3.57	0.18	5.26%
162	3.5	3.25	3.56	0.31	9.4%
212	3.5	3.5	3.2	-0.3	-8.57%
8	3.5	4.0	3.5	-0.5	-12.5%
320	3.46	3.75	3.8	0.05	1.33%
17	3.4	3.25	2.5	-0.75	-23.08%
245	3.31	3.0	3.67	0.67	22.22%
142	3.29	3.67	3.17	-0.5	-13.64%
129	3.27	3.0	3.5	0.5	16.67%
175	3.25	3.0	3.14	0.14	4.76%
265	3.24	3.52	3.0	-0.52	-14.81%
67	3.22	2.7	3.33	0.63	23.46%
155	3.22	3.42	2.33	-1.09	-31.79%
110	3.2	2.75	3.6	0.85	30.91%
133	3.2	3.42	3.0	-0.42	-12.2%
378	3.19	3.1	3.44	0.35	11.28%
3	3.18	3.6	3.12	-0.48	-13.19%
31	3.18	3.14	3.27	0.13	4.13%
250	3.17	3.22	3.0	-0.22	-6.9%
370	3.17	3.2	3.33	0.13	4.17%
21	3.15	3.0	3.3	0.3	10.0%
41	3.15	2.71	2.75	0.04	1.32%
243	3.15	3.5	3.07	-0.43	-12.24%
251	3.13	3.0	3.5	0.5	16.67%
33	3.11	2.9	2.33	-0.57	-19.54%
323	3.1	3.1	3.0	-0.1	-3.33%
197	3.1	3.8	3.0	-0.8	-21.05%
177	3.09	3.1	3.22	0.12	3.83%
132	3.08	2.57	3.0	0.43	16.67%
49	3.07	3.2	3.0	-0.2	-6.25%
312	3.0	2.77	3.33	0.56	20.37%
373	3.0	2.5	2.8	0.3	12.0%
218	3.0	3.4	3.67	0.27	7.84%
344	3.0	3.44	3.5	0.06	1.61%
36	3.0	3.0	3.0	0.0	0.0%
230	3.0	3.0	3.0	0.0	0.0%
151	3.0	3.25	2.88	-0.38	-11.54%
256	3.0	3.08	2.67	-0.41	-13.33%
247	3.0	3.11	2.57	-0.54	-17.35%
244	3.0	3.8	2.33	-1.47	-38.6%
137	2.96	2.5	2.82	0.32	12.94%
81	2.92	2.67	3.0	0.33	12.5%
201	2.91	2.83	3.0	0.17	5.88%
157	2.9	2.8	3.0	0.2	7.14%

Tabela 21: Desempenho dos alunos em programação - parte 3

Aluno (ID)	D	IDA_P	IDP_P	δ	Δ
249	2.87	2.5	3.0	0.5	20.0%
194	2.86	2.33	3.3	0.97	41.43%
10	2.83	2.69	2.67	-0.02	-0.78%
278	2.83	3.0	2.67	-0.33	-11.11%
70	2.79	2.89	2.14	-0.75	-25.82%
66	2.77	2.8	2.43	-0.37	-13.27%
53	2.75	2.71	2.75	0.04	1.32%
146	2.71	3.43	2.43	-1.0	-29.17%
124	2.69	2.86	2.8	-0.06	-2.0%
103	2.67	3.0	2.33	-0.67	-22.22%
78	2.63	1.73	3.0	1.27	73.68%
139	2.61	4.0	2.22	-1.78	-44.57%
5	2.54	2.33	2.44	0.11	4.76%
255	2.53	2.6	2.67	0.07	2.56%
188	2.52	2.25	2.48	0.23	10.05%
39	2.41	2.0	2.0	0.0	0.0%
279	2.41	2.22	1.5	-0.72	-32.5%
362	2.36	2.59	0.0	-2.59	-100.0%
236	2.34	2.0	2.36	0.36	18.18%
29	2.33	1.75	2.64	0.89	50.65%
44	2.33	3.0	1.33	-1.67	-55.56%
339	2.33	2.62	0.0	-2.62	-100.0%
2	2.27	2.0	2.18	0.18	9.09%
54	2.25	0.0	3.0	3.0	0.0%
11	2.23	2.5	2.17	-0.33	-13.33%
105	2.08	2.33	2.2	-0.13	-5.71%
85	2.04	2.0	1.62	-0.38	-19.23%
113	2.04	3.0	1.84	-1.16	-38.6%
187	2.0	2.0	1.67	-0.33	-16.67%
280	2.0	2.12	1.0	-1.12	-52.94%
283	2.0	2.0	0.0	-2.0	-100.0%
220	1.87	1.71	2.14	0.43	25.0%
55	1.82	1.89	1.92	0.03	1.81%
32	1.78	2.67	1.73	-0.94	-35.23%
361	1.75	2.0	0.5	-1.5	-75.0%
372	1.69	1.4	1.56	0.16	11.11%
379	1.4	1.5	1.33	-0.17	-11.11%
299	0.75	0.0	0.0	0.0	0.0%
343	0.67	0.0	0.2	0.2	0.0%

ANEXO E – Código de criação da base de turmas

Código E.1: Script python da criação das turmas

```
1  # -*- coding: utf-8 -*-
2
3  import json
4  from sets import Set
5  from operator import itemgetter
6  from pprint import pprint
7
8  ######################################################################
9  # Pega um objeto em um dicionario cujo valor da chave e igual o valor
       desejado
10 def getObj(dictionary, key, value):
11     for obj in dictionary:
12         if obj[key] == value:
13             return obj
14     return None
15
16 # Pega uma turma de determinado periodo e ano de uma disciplina
17 def getTurma(disciplinas,codigo,ano,periodo):
18     for disciplina in disciplinas:
19         if disciplina["codigo"] == codigo:
20             for turma in disciplina["turmas"]:
21                 if turma["ano"] == ano and turma["periodo"] == periodo:
22                     return turma
23     return None
24
25 # Adiciona um aluno em uma determinada turma
26 def addAlunoIn(array_turmas,ano,periodo,codigo,aluno):
27     for t_ano in array_turmas:
28         if t_ano["ano"] == ano:
29             for t_periodo in t_ano["periodos"]:
30                 if t_periodo["periodo"] == periodo:
31                     for t_turma in t_periodo["turmas"]:
32                         if t_turma["codigo"] == codigo:
33                             t_turma["alunos"].append(aluno)
34
35 print 'INICIO: JSON Turmas'
36
37 ######################################################################
38 # Abrindo arquivos de historicos e disciplinas alvo
```

```
39 alunos_file_path = "./json/historicos.json"
40 disciplinas_file_path = "./json/disciplinas_programacao.json"
41
42 # Historicos de todos os alunos
43 with open(alunos_file_path) as data_file:
44     historicos = json.load(data_file)
45 # Disciplinas de programacao
46 with open(disciplinas_file_path) as data_file:
47     disciplinas = json.load(data_file)
48
49 ########################################################################
50 #Set com os codigos das disciplinas de programacao
51 disciplinas_programacao = []
52 for disciplina in disciplinas:
53     disciplinas_programacao.append(disciplina["codigo"])
54
55 disciplinas_programacao = Set(disciplinas_programacao)
56
57 ########################################################################
58 # Filtrando as mencoes de cada aluno por disciplina, por periodo
59 resultados = []
60 for aluno in historicos:
61     for semestre in aluno["historico"]:
62         ano = semestre["ano"]
63         periodo = semestre["periodo"]
64         for disciplina in semestre["disciplinas"]:
65             if disciplina["codigo"] in disciplinas_programacao:
66                 entrada = {}
67                 entrada["aluno"] = aluno["id"]
68                 entrada["mencao"] = disciplina["mencao"]
69                 entrada["codigo"] = disciplina["codigo"]
70                 entrada["periodo"] = periodo
71                 entrada["ano"] = ano
72                 resultados.append(entrada)
73
74 ########################################################################
75 # Pegar todos os anos na base de dados
76 anos = Set([])
77 for aluno in historicos:
78     for semestre in aluno["historico"]:
79         anos.add(semestre["ano"])
80
81 # Cria as turmas por ano e por semestre
82 turmas = []
83 for num in anos:
84     ano = {}
85     ano["ano"] = num
```

```
86      ano["periodos"] = []
87
88      temp_turmas = []
89      for codigo in disciplinas_programacao:
90          obj = {}
91          obj["codigo"] = codigo
92          obj["alunos"] = []
93          temp_turmas.append(obj)
94
95      primeiro_periodo = {}
96      primeiro_periodo["periodo"] = "1"
97      primeiro_periodo["turmas"] = temp_turmas
98
99      segundo_periodo = {}
100     segundo_periodo["periodo"] = "2"
101     segundo_periodo["turmas"] = temp_turmas
102
103
104     ano["periodos"].append(primeiro_periodo)
105     ano["periodos"].append(segundo_periodo)
106
107     turmas.append(ano)
108
109 ########################################################################
110 i = 1
111 for e in resultados:
112
113     aluno = {}
114     aluno["id"] = e["aluno"]
115     aluno["mencao"] = e["mencao"]
116
117     addAlunoIn(turmas,e["ano"],e["periodo"],e["codigo"], aluno)
118
119 ########################################################################
120 # Cria o JSON com as turmas
121 print '-- Criando JSON com as turmas das disciplinas de programacao'
122 turmasJSON = open('./json/turmas.json','w')
123 turmasJSON.write(json.dumps(turmas, sort_keys=False, indent=4,
        separators=(',',': ')))
124 print 'FIM: JSON Turmas'
```

ANEXO F – Exemplo de um objeto turma

Código F.1: Exemplo de um objeto JSON de uma turma

```
"201286_2014_2": {
        "alunos_ids": [
            71,
            80,
            155,
            162,
            256,
            257
        ],
        "nome": "Fundamentos de Sistemas Operacionias",
        "periodo": "2",
        "ano": "2014",
        "alunos_mencoes": [
            "MS",
            "MS",
            "MM",
            "MM",
            "MM",
            "TR"
        ],
        "has_ejudge": true,
        "histograma": {
            "CC": 0,
            "AP": 0,
            "SS": 0,
            "SR": 0,
            "TR": 1,
            "MI": 0,
            "II": 0,
            "MM": 3,
            "TJ": 0,
            "MS": 2,
            "DP": 0
        },
        "codigo": "201286"
    }
```

ANEXO G – Código de processamento das turmas

Código G.1: Script python para extrair métricas da base de dados das turmas

```
1  # -*- coding: utf-8 -*-
2
3  import json
4  from sets import Set
5  from pprint import pprint
6
7  # Abrindo o arquivo de turmas
8  turmas_file_path = "./json/turmas.json"
9  with open(turmas_file_path) as data_file:
10     turmas = json.load(data_file)
11
12 disciplinas_programacao_file_path = "./json/
       disciplinas_programacao_indexado.json"
13 with open(disciplinas_programacao_file_path) as data_file:
14     disciplinas_programacao_indexado = json.load(data_file)
15
16 #pprint(disciplinas_programacao_indexado)
17
18 turmas_data = {}
19
20 # Turmas com eJudge
21
22 # Histogramas
23 for ano in turmas:
24     for periodo in ano["periodos"]:
25 #         print '\n',ano["ano"], periodo["periodo"]
26         for disciplina in periodo["turmas"]:
27             turma = {}
28             turma["codigo"] = disciplina["codigo"]
29             turma["ano"] = ano["ano"]
30             turma["periodo"] = periodo["periodo"]
31             turma["nome"] = disciplinas_programacao_indexado[disciplina[
                   "codigo"]]["nome"]
32             turma["alunos_mencoes"] = []
33             for aluno in disciplina["alunos"]:
34                 turma["alunos_mencoes"].append(aluno["mencao"])
35
36             turma["alunos_ids"] = []
37             for aluno in disciplina["alunos"]:
```

```
38                  turma["alunos_ids"].append(aluno["id"])
39
40              turma["histograma"] = {}
41              # Inicializa o histograma
42              turma["histograma"]['SR'] = 0
43              turma["histograma"]['II'] = 0
44              turma["histograma"]['MI'] = 0
45              turma["histograma"]['MM'] = 0
46              turma["histograma"]['MS'] = 0
47              turma["histograma"]['SS'] = 0
48              turma["histograma"]['TR'] = 0
49              turma["histograma"]['CC'] = 0
50              turma["histograma"]['AP'] = 0
51              turma["histograma"]['DP'] = 0
52              turma["histograma"]['TJ'] = 0
53
54              for mencao in turma["alunos_mencoes"]:
55                  turma["histograma"][mencao] += 1
56
57              key = turma["codigo"] + '_' + turma["ano"] + '_' + turma["
                    periodo"]
58
59              turmas_data[key] = turma
60
61  pprint(turmas_data)
62
63  turmasDataJSON = open('./json/turmas_data.json','w')
64  turmasDataJSON.write(json.dumps(turmas_data, sort_keys=False, indent=4,
        separators=(',',': ')))
```