ISCTE IUL
Instituto Universitário de Lisboa

DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIAS DE INFORMAÇÃO

# ACODE: SISTEMA WEB PARA AVALIAÇÃO AUTOMÁTICA DE ALGORITMOS JAVA

**MARCOS ANDRÉ MOREIRA PINTO**

DISSERTAÇÃO SUBMETIDA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA DE INFORMÁTICA NO RAMO EM SISTEMAS DE INFORMAÇÃO E GESTÃO DO CONHECIMENTO

ORIENTADOR:
PROFESSOR DOUTOR ANTÓNIO LOPES, ASSISTENTE
ISCTE-IUL

SETEMBRO DE 2012

ISCTE IUL
Instituto Universitário de Lisboa

ACode: Sistema Web para Avaliação Automática de Algoritmos Java

Marcos André Moreira Pinto

Setembro

2012

ISCTE IUL
Instituto Universitário de Lisboa

DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIAS DE INFORMAÇÃO

# ACODE: SISTEMA WEB PARA AVALIAÇÃO AUTOMÁTICA DE ALGORITMOS JAVA

**MARCOS ANDRÉ MOREIRA PINTO**

DISSERTAÇÃO SUBMETIDA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA DE INFORMÁTICA NO RAMO EM SISTEMAS DE INFORMAÇÃO E GESTÃO DO CONHECIMENTO

ORIENTADOR:
PROFESSOR DOUTOR ANTÓNIO LOPES, ASSISTENTE
ISCTE-IUL

SETEMBRO DE 2012

# Resumo

A avaliação automática de código-fonte pode ser uma ferramenta importante na aprendizagem de conceitos e linguagens de programação uma vez que permite aos alunos obterem *feedback* automático em relação à solução que apresentam para determinado problema. As ferramentas existentes fazem uso de testes de entrada e saída de parâmetros para avaliar código submetido por alunos. No entanto, para uma avaliação mais completa é necessário acrescentar a esse processo de avaliação automática técnicas mais avançadas como métricas de engenharia de software.

Face às limitações dessas abordagens, o trabalho de investigação apresentado na presente dissertação descreve e avalia uma abordagem faseada para a avaliação automática de código-fonte: i) o código-fonte do aluno é primeiramente compilado e é analisado se existe algum erro; ii) o código-fonte compilado é testado através de um conjunto de testes JUnit fornecidos pelo docente; iii) é usado um conjunto de métricas de engenharia de software para comparar a solução do aluno com a solução do docente; iv) e, finalmente, com base nas etapas anteriores, o *feedback* é fornecido ao aluno para que possa auto-avaliar as suas capacidades e identificar as áreas em que necessita de mais estudo e/ou exercícios. O protótipo baseado nesta abordagem foi implementado num ambiente Web e foi testado recorrendo a alunos do público-alvo e com a utilização de um browser de internet. Os resultados da avaliação comprovam que o uso deste tipo de ferramentas facilita a aprendizagem e aquisição de conhecimentos por parte do aluno devido ao *feedback* imediato e constante fornecido pelo sistema.

**Palavras-chave:** *Feedback* ao aluno, avaliação automática, aplicação web, código java

# Abstract

The automatic evaluation of source code can be an important tool in the learning of concepts and programming languages since it allows students to obtain immediate feedback on the solutions submitted to a programming problem. The existing tools make use of input/output testing to evaluate the code submitted by students. However, for a more complete evaluation, it is necessary to add more advanced techniques to that process of automatic evaluation, such as software engineering metrics.

Given the limitations of these approaches, the research work presented in this dissertation describes and evaluates a phased approach for the automatic evaluation of source code: i) the student's source code is first compiled and checked for any errors; ii) the compiled code is then tested against a set of JUnit tests provided by the teacher; iii) a set of software engineering metrics is used to compare the student's solution against the teacher's solution; iv) and finally, based on the previous stages, feedback is provided to the students so they can self-evaluate and identify the areas in which they need further study and/or exercises. The prototype based on this approach has been implemented in a Web environment and has been tested by the student audience with the use of a web browser. The evaluation results show that the use of these tools improves the learning and the acquisition of knowledge by the student due to the constant and immediate feedback provided by the system.

**Keywords:** Student feedback, automatic evaluation, web application, java code

# Agradecimentos

Esta dissertação de mestrado não poderia ter sido feita sem a ajuda e o apoio de várias pessoas que, sem dúvida, merecem uma palavra de apreço.

Em primeiro lugar gostaria de agradecer ao meu orientador pois a sua sabedoria contribuíram para o meu crescimento pessoal e intelectual. O seu envolvimento foi fulcral para a concretização desta dissertação de mestrado.

Em segundo lugar, deixo uma palavra de apreço à Escola de Tecnologias e Arquitectura e, em particular, à assistente administrativa Fátima Maria Silva por demonstrar sempre boa disposição em ajudar em todo o processo burocrático necessário para a realização desta dissertação.

Não esquecendo o Instituto de Telecomunicações - Instituto Universitário de Lisboa, deixo também uma palavra de apreço pelo apoio e meios físicos utilizados na dissertação.

À minha família e em especial aos meus pais pelas palavras amigas e de incentivo, elevando o meu estado de espírito nas alturas mais complicadas, permitindo levar esta dissertação de mestrado a bom porto.

Aos meus amigos e a todos os que estiveram directamente ou indirectamente envolvidos na realização desta dissertação.

Aos meus pais
António e Maria

# Índice

# Índice de Tabelas

# Índice de Figuras

# Glossário

**TCP:** Transmission Control Protocol

**IP:** Internet Protocol

**HTTP:** HyperText Transfer Protocol

**TCL Script:** Tool Command Language Script

**CGI:** Common Gateway Interface

**MySQL:** My Structured Query Language

**IDE:** Integrated Development Environment

**LAMS:** Learning Activity Management System

**HTML:** HyperText Markup Language

**CSS:** Cascading Style Sheets

**PHP:** PHP Hypertext Preprocessor

# Capítulo I - Introdução

## *1.1 Introdução*

Uma linguagem de programação traduz-se num conjunto de regras sintáticas e semânticas para definir um programa de computador, permitindo executar um conjunto de acções previamente definidas.

"Using a programming language requires a mutual understanding between a person and a machine. This can be more difficult to achieve than understanding between people because machines are so much more literal than human beings." (Fischer, et al., 1993)

Aprender uma linguagem de programação pode ser uma actividade desafiante, especialmente para alunos do primeiro ano que nunca usaram os principais conceitos de programação. O sucesso nesta área está geralmente associado à elaboração de vários exercícios, disponibilizados pelo docente e resolvidos pelo aluno, de modo a avaliar o conhecimento de cada um deles e, portanto, existir uma maior concentração em áreas que o aluno não domina na totalidade. O docente avalia os alunos através de exercícios fornecidos ao longo do semestre, o que permite uma análise das capacidades actuais de cada um deles através das respostas apresentadas. Posteriormente, o docente atribui individualmente uma nota a cada exercício resolvido e um *feedback* para que o aluno possa fazer a sua auto-avaliação e determinar os conceitos que são necessários para exercícios futuros.

No entanto, quando existem muitos alunos, algo que é habitual nos primeiros anos de um curso do ensino superior, o docente não consegue realizar vários períodos de avaliação ao longo do semestre pois existem restrições de tempo e recursos que o impossibilitam. Desta forma, não existem períodos de avaliação suficientes que permita a cada aluno efectuar a sua auto-avaliação.

Para os alunos que frequentam cursos que requerem linguagens de programação, é fundamental obterem algum *feedback* dos exercícios que resolvem. Contudo, se for atribuída a possibilidade a cada aluno de poder entregar várias versões (cada uma melhor que a outra), poderão realizar a sua auto-avaliação da melhor forma possível, permitindo obter o resultado pretendido.

Sendo a internet um conglomerado de redes de comunicação em escala mundial que liga milhões de computadores ligados pelo protocolo de comunicação TCP/IP e que permite o acesso a informações e todo o tipo de transferência de dados, torna-se numa estrutura essencial para suportar um mecanismo de entrega de projectos remotamente, quer pelos alunos quer pelos docentes. No capítulo 2, e em particular na secção 2.2, verificamos que esta estrutura já é utilizada para dar resposta à necessidade dos docentes em disponibilizar e avaliar de forma automática exercícios que recorrem a linguagens de programação.

Apesar de existirem ferramentas que aliviam o trabalho do docente em avaliar os algoritmos fornecidos pelos alunos, apresentam algumas limitações (capitulo II secção 2.14). Com base nessas limitações foi pensado e desenvolvido um sistema Web, ACode, capaz de analisar e avaliar cada algoritmo simples (algoritmos existentes na unidade curricular de Introdução à Programação inserida em alguns dos cursos de ensino superior) fornecido pelos alunos. Neste sistema, cada estudante pode submeter várias soluções, o que permite melhorar o seu desempenho e capacidades na área da programação. Para tal, o docente apenas terá que fornecer o enunciado de cada exercício, a solução ideal e alguns parâmetros de classificação, nomeadamente testes JUnit[1] (capítulo III, secção 3.4, sub-secção 3.4.2.1) e as percentagens associadas a cada uma das métricas de engenharia de software analisadas pelo sistema. Posteriormente, o sistema fornece ao aluno um *feedback* completo após ser fornecida a resposta para um determinado exercício.

## *1.2 Âmbito*

A presente dissertação de mestrado assenta em dois objectivos importantes: o primeiro dedicado ao desenvolvimento de um módulo capaz de efectuar avaliações automáticas de algoritmos simples produzidos por alunos dos cursos que incorporam a unidade curricular de Introdução à Programação; e o segundo dedicado a construir um sistema Web que suporte a entrega de soluções de alunos e, por aplicação do módulo referido anteriormente, atribuir de forma automática a nota final e fornecer o respectivo *feedback* associado a cada exercício escolhido.

Em conformidade com as orientações definidas para as dissertações de mestrado e com a necessidade sentida pelos docentes que leccionam a unidade curricular de Introdução à Programação, surgiu a oportunidade de desenvolver este trabalho de pesquisa e implementação.

O desafio proposto pelo docente orientador deste trabalho traduz-se, numa primeira fase, pela análise cuidada de trabalho já desenvolvido na área para, mais tarde, poder analisar as falhas que cada um dos sistemas já desenvolvidos apresentam. Posteriormente, é proposto um mecanismo de avaliação automática de código-fonte que compare o algoritmo de um programa, fornecido pelo aluno, com um esquema da solução ideal para um determinado problema. Após o mecanismo de avaliação estar concluído e para finalizar, será necessário uma implementação Web do sistema para que esteja disponível aos alunos 24 horas por dia.

## *1.3 Problema*

Tal como foi referido, um dos problemas que afecta a aprendizagem do estudante é não receber o *feedback* das suas respostas aos problemas em tempo útil. A causa deste problema provém de um aumento significativo dos alunos na unidade curricular de Introdução à Programação o que nos

[1] http://www.junit.org/

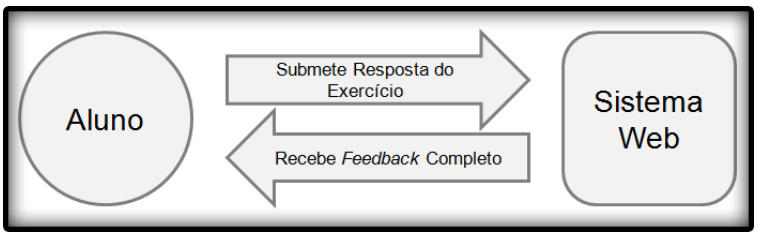
remete para a consequência de não poderem ser atribuídos, pelo docente, mais exercícios devido a restrições de tempo e recursos.

Alguns sistemas dão resposta e auxiliam as tarefas dos docentes (i.e. avaliação de exercícios), contudo não o fazem da forma mais correcta. Avaliam o comportamento do algoritmo em relação aos seus parâmetros de entrada e saída. Alguns destes sistemas também armazenam o algoritmo do estudante permitindo ao docente analisar cada um deles de forma manual.

Com isto, a principal questão que se coloca é: *como analisar e avaliar o algoritmo produzido pelo estudante de forma a que este receba o respectivo feedback em tempo útil?* A resposta a esta questão é relevante não só pelo uso de um sistema que atribui, de uma forma automatizada, *feedback* aos alunos do ISCTE - Instituto Universitário de Lisboa, como também poderá servir de base à avaliação de outras instituições de ensino, no que respeita ao processo de avaliação de algoritmos simples, ou até, quem sabe, para outras unidades curriculares após ter sido desenvolvido um novo módulo que suporte a respectiva avaliação.

A questão formulada anteriormente irá ser estudada e analisada ao longo deste documento, contudo, numa primeira fase, é possível identificar um sistema que irá receber uma solução do estudante e atribuir-lhe o respectivo *feedback* da solução (figura 01).

**Figura 01**: Comunicação do aluno com o sistema




Fonte: Marcos Pinto (2012)

Posto isto, torna-se importante desenvolver um módulo que automatiza o processo de avaliação de código-fonte através da análise dos dados de entrada e saída do algoritmo desenvolvido pelo aluno e através da analise de métricas de engenharia de software do código-fonte produzido, atribuindo, no final da análise, um *feedback* completo à respectiva resposta submetida. As métricas de engenharia de software que serão analisadas no algoritmo do aluno, estão detalhadas neste documento (capítulo III, secção 3.4, sub-secção 3.4.2.2), e, são usadas na avaliação para que o docente obtenha um resultado mais realista do código-fonte desenvolvido pelo aluno.

## *1.4 Objectivos*

Esta dissertação de mestrado tem como objectivo contribuir não só para o avanço da literatura relacionada com o tema através do desenvolvimento de um novo sistema, como também ajudar

alunos e docentes ligados à unidade curricular de Introdução à Programação na automatização de tarefas relacionadas com o processo de avaliação de exercícios.

O sistema de avaliação permite ao estudante submeter soluções para exercícios propostos pelo docente, obtendo, em tempo útil, um *feedback* completo. Este projecto analisa e avalia o código-fonte produzido pelo aluno, facilitando as tarefas do docente na atribuição de notas e respectivo *feedback* para cada exercício respondido. Para além do enunciado para um problema, o docente fornece também uma solução para o exercício, considerada ideal, e, fornece também, um conjunto de testes para que o algoritmo do estudante seja submetido a uma pré-verificação.

O processo de funcionamento deste sistema baseia-se num sistema Web semelhante a alguns sistemas apresentados ao longo do capítulo 2 (i.e. Mooshak - apresentado na secção 2.2). O aluno verifica quais os exercícios disponíveis (através do *front-office* do sistema) para resolver e, após desenvolver a sua resposta, submete-a para avaliação. Posteriormente, e através de um processo automatizado de avaliação, o aluno recebe o *feedback* completo da sua resposta. Este *feedback* é produzido pelo módulo responsável da avaliação e fica armazenado numa base de dados SQL, o que permite, a qualquer altura, que o relatório seja consultado pelo aluno e pelo docente. Através deste processo, o docente pode, a qualquer altura, analisar a solução do aluno caso verifique alguma anomalia referente ao processo automatizado da avaliação.

O trabalho realizado no âmbito desta dissertação de mestrado inclui a análise das principais métricas de engenharia de software utilizadas na avaliação de algoritmos simples, a implementação do módulo responsável pela avaliação automática de código-fonte, o desenho da arquitectura e implementação do sistema Web, fornecendo as funcionalidades esperadas tanto para o docente como para o aluno.

## *1.5 Motivações*

Sendo o foco desta dissertação a implementação de um módulo responsável pela análise e avaliação de algoritmos produzidos por alunos, muitos são os docentes que beneficiarão, de uma forma directa, com a produção desta ferramenta.

Para melhorar o ensino de linguagens de programação e para simplificar as tarefas dos docentes na avaliação de código-fonte, torna-se importante implementar um módulo que suporte a automatização do processo de avaliação de exercícios e um sistema Web (para permitir o funcionamento correcto do módulo de avaliação) com a finalidade de apresentar ao aluno o *feedback* das respostas fornecidas, melhorando os seus conceitos e técnicas de programação.

A tentativa de desenvolver um mecanismo capaz de efectuar a avaliação automática de algoritmos tem vindo a ser estudada e investigada por muitos investigadores. Desta forma, existe algum trabalho

relacionado, demonstrado no capítulo 2 deste documento. No entanto, o mecanismo de avaliação que os sistemas descritos no capítulo 2 apresentam não é o mais indicado para atribuição de notas finais, visto que apresenta algumas limitações. Uma das principais limitações e um dos principais motivos para a escolha deste tema na dissertação de mestrado é o facto do processo de avaliação estar restrito à comparação dos dados produzidos pelo algoritmo do aluno com os dados inseridos no sistema pelo docente. Por muito optimizado que seja o sistema de avaliação, este mecanismo não é, de todo, o mais apropriado para dar suporte à avaliação final do docente. Esta avaliação requer uma análise ao algoritmo funcional e uma análise de desempenho, ou seja, verificar se o algoritmo está desenvolvido de forma a que funcione, no final, da forma mais eficiente possível.

Com base na finalidade aqui descrita, o módulo desenvolvido analisa de forma automática o algoritmo produzido pelo estudante atribuindo, não só uma nota ao respectivo algoritmo, como também irá fornecer ao estudante o *feedback* necessário para que este possa melhorar o desenvolvimento dos algoritmos, aperfeiçoando, no final, o seu desempenho na unidade curricular de Introdução à Programação.

## *1.6 Estrutura do Documento*

Esta dissertação está organizada em 6 capítulos, subdivididos em diversos tópicos, para ajudar a sua estruturação e compreensão. No presente capítulo (Capítulo I - Introdução) foi feita uma introdução ao trabalho realizado onde é apresentado o âmbito do trabalho, os objectivos e as motivações. No capítulo II (Fundamentação Teórica) foi feita uma apresentação dos sistemas existentes na literatura, apresentando as limitações de cada sistema. Em seguida, no capítulo III (Conceptualização do Sistema) foi descrito o cenário utilizado para demonstrar o trabalho que foi produzido ao longo da dissertação de mestrado. O capítulo IV (Implementação do Sistema) utilizou o trabalho descrito no capítulo III para demonstrar a forma como foi implementado o sistema Web e o módulo responsável pela avaliação automática de algoritmos. O capítulo V (Avaliação do Sistema ACode) foi dedicado à avaliação do trabalho realizado ao longo desta dissertação de mestrado. Finalmente, no capítulo VI (Conclusões e Trabalho Futuro) conclui-se a dissertação apresentando as contribuições, as limitações e o trabalho futuro.

Na última parte deste documento, na secção relativa aos anexos são apresentados, para além dos anexos referidos ao longo da dissertação, o artigo escrito no período do trabalho realizado e que foi aceite num *workshop* da área relacionada.

# Capítulo II - Fundamentação Teórica

## 2.1 Introdução

Este capítulo apresenta uma revisão geral da literatura com o intuito de estabelecer o referencial teórico que sustenta o desenvolvimento desta dissertação de mestrado. Estão apresentados, neste capítulo, alguns sistemas de *e-learning* e de avaliação automática de algoritmos, divididos pela diferença dos seus objectivos intrínsecos para os quais foram desenvolvidos. No final deste capítulo será apresentada uma breve conclusão, com o intuito de fazer uma comparação entre os sistemas apresentados e identificar as limitações dos mesmos que motivaram a realização deste trabalho de investigação.

Os sistemas que têm como função avaliar um algoritmo específico, automaticamente, através de um conjunto de dados de entrada, permitem aliviar algum do trabalho do docente. Para que um sistema deste tipo seja aceite pelo docente (avaliação de algoritmos de acordo com dados específicos aquando da implementação do sistema), é necessário que seja capaz de receber um determinado algoritmo, compilá-lo, executá-lo com alguns dados de testes fornecidos pelo docente e, por último, comparar um conjunto de métricas de engenharia de software entre o algoritmo produzido pelo aluno e o algoritmo submetido com o problema pelo docente.

Neste capítulo estão apresentados alguns sistemas que têm como foco principal apoiar os concursos de programação, nomeadamente o Mooshak (secção 2.2), o PC[2] (secção 2.3), o UVa Judge e EduJudge (secção 2.4). Também estão apresentados dois mecanismos que efectuam a avaliação do código-fonte, sendo um através do sistema SICAS (secção 2.5) e o outro que recorre a um módulo desenvolvido para o Moodle (secção 2.6). Um sistema de apoio curricular e quatro sistemas com semelhanças a IDEs também estão presentes neste capítulo: o SAmbA (secção 2.7), o Portugol IDE (secção 2.8), o JavaTool (secção 2.9), o JEduc (secção 2.10) e o AMBAP (secção 2.11). Também é apresentado, não só, uma descrição de um sistema que permite a gestão académica, nomeadamente o Fénix (secção 2.12), como também, uma breve descrição do sistema e-Learning (secção 2.13). Para concluir, é apresentado uma conclusão (secção 2.14) relativa ao estado da arte.

## 2.2 Mooshak

O Mooshak[2] é um sistema Web desenvolvido no âmbito da gestão, automática, de concursos ligados à prática da programação (ver figura 02) e foi desenvolvido por professores da Faculdade de Ciências da Universidade do Porto: José Paulo Leal e Fernando Silva. Actualmente este sistema está a ser, cada vez mais, utilizado em cursos que requerem a aprendizagem de conceitos de

[2] http://mooshak.dcc.fc.up.pt/

programação, para fornecer um *feedback* automático relativo à resposta de cada exercício submetido pelo docente.

**Figura 02**: Sistema Mooshak


Fonte: http://mooshak.dcc.fc.up.pt/~mooshak-test (2011)

Estando este sistema inicialmente vocacionado para suportar a gestão de concursos, é importante apresentar um formulário de autenticação para cada equipa registada. Todas as equipas/estudantes podem aceder ao Mooshak em qualquer *browser* de internet, visto que este sistema está implementado sob o protocolo HTTP.

O modo de funcionamento deste sistema traduz alguma simplicidade para as diferentes equipas/estudantes pois permite a visibilidade dos exercícios através do *front-office*[3] da aplicação. Através deste sistema, as equipas/estudantes obtêm um conjunto de problemas fornecidos pelo professor (o professor submete problemas juntamente com respostas e dados de entrada e saída), e dispõem apenas de um computador para resolver os exercícios, podendo utilizar linguagens de programação como C, C++, Java ou Pascal (Leal, et al., 2003). Depois da equipa/estudante submeter a resposta ao problema, o Mooshak, avalia-a em duas fases distintas consoante o tipo de análise: estática e dinâmica.

Na primeira fase, fase estática, o sistema verifica se o autor da submissão do problema é válido, qual a linguagem de programação utilizada, qual o problema que está associado à submissão da resposta dada pela equipa/estudante, verifica se o tamanho da resposta fornecida pela equipa/estudante não excede um limite máximo predefinido e verifica também se a resposta compila correctamente. Se o Mooshak detectar alguma falha numa das verificações enumeradas na análise estática, então irá apresentar à equipa/estudante a respectiva classificação da falha ocorrida. A tabela 01 mostra-nos as classificações das falhas que poderão ocorrer nesta fase.

[3] Interface responsável por apresentar as informações do sistema a cada equipa

**Tabela 01**: Classificação atribuída na fase estática

| Verificação | Classificação |
|---|---|
| Autor da submissão | Submissão Inválida |
| Linguagem de programação | Submissão Inválida |
| Problema a resolver | Submissão Inválida |
| Tamanho da resposta submetida | Resposta demasiado grande |
| Compilação do resposta submetida | Erro no tempo de compilação |

Se o sistema não fornecer nenhum erro durante a fase estática, segue automaticamente para a fase dinâmica onde executa o algoritmo submetido pela equipa/estudante através de dados predefinidos para testes, apresentados sob a forma de um ficheiro (designado por ficheiro de *input*). Estes dados são seleccionados pelo sistema de acordo com o exercício a resolver e foram produzidos inicialmente com o problema submetido pelo professor. A execução do algoritmo submetido pela equipa/estudante origina um ficheiro de dados (designado por ficheiro de *output* da equipa/estudante) que, quando comparado com o ficheiro de dados correcto (designado por ficheiro de *output* do problema), fornece um resultado. A tabela 02 indica-nos as classificações possíveis que o Mooshak atribui ao longo da execução do algoritmo submetido. Cada classificação tem associado um grau de gravidade correspondente e, o resultado final fornecido pelo sistema, é o que apresentar maior grau (o grau de gravidade mais elevado é relativo a falhas de execução do sistema operativo, i.e. incapacidade de correr mais processos).

**Tabela 02**: Classificação atribuída na fase dinâmica

| Gravidade | Classificação |
|---|---|
| 6 | Requer reavaliação |
| 5 | Limite de tempo excedido |
| 4 | Resposta demasiado grande |
| 3 | Erro no tempo de execução |
| 2 | Resposta errada |
| 1 | Erro de apresentação dos resultados |
| 0 | Aceite |

O sistema irá dar uma classificação aceite se, e só se, os dados do ficheiro de *output* da equipa/estudante forem exactamente iguais aos dados do ficheiro de *output*, fornecido pelo docente, do problema específico. A comparação entre estes dois ficheiros de *output* é feita através do comando Linux *diff* que compara dois ficheiros de texto e apresenta as diferenças existentes entre eles (Company, 2012).

O Mooshak é um sistema *open-source* (sistemas que permitem alteração da parte do administrador), implementado sobre o sistema operativo Linux, que utiliza a linguagem de programação TCL Script e o protocolo CGI que serve de base para estabelecer a comunicação entre

o protocolo HTTP e a linguagem de programação (Leal, et al., 2002). A gestão dos dados que este sistema utiliza é feita através de ficheiros, embutidos no próprio sistema de ficheiros, não utilizando base de dados relacionais para guardar os dados persistentes. Este sistema apresenta uma arquitectura escalável que permite não só utilizar um único servidor para concursos pequenos como também permite utilizar vários servidores se os concursos forem de âmbito internacional.

Este sistema foi usado na Universidade de Murcia[4], em Espanha, o que permitiu reduzir a taxa de reprovação dos alunos para 40% (García-Mateos, et al., 2009) o que implica uma taxa superior de aprovação após o uso do Mooshak. Com isto, podemos desde logo verificar que este tipo de sistema incentiva de facto os alunos a praticar mais exercícios. Com o Mooshak o aluno obtém um *feedback* automático após submeter a resposta ao problema, permitindo melhorar os conceitos e técnicas adquiridas em cursos de programação e aliviar as tarefas do professor no que respeita às avaliações individuais de exercícios.

O Mooshak apresenta algumas características importantes para a elaboração do projecto. A presença de um formulário de autenticação permite que os algoritmos sejam armazenados no sistema sem consulta de terceiros; o sistema Web que permite uma disponibilidade 24 horas por dia; a simplicidade em fornecer respostas aos exercícios propostos; avaliação automática de código-fonte e atribuição automática do respectivo *feedback*, são algumas das características base para a implementação do projecto final. No entanto, o processo de avaliação que este sistema apresenta não é o mais indicado para fazer avaliações automáticas de algoritmos, pois baseia-se apenas na comparação dos dados de *output* com os dados para testes fornecidos pelo docente.

## *2.3 PC²*

O PC², *Programming Contest Control*, foi desenvolvido na Universidade do Estado da Califórnia, em Sacramento, com a finalidade de gerir as competições que envolvem programação. Este sistema foi desenvolvido em Java. Ao contrário do Mooshak (secção 2.2), o PC² é apenas aplicacional sendo necessário armazenar a aplicação em cada computador do estudante o que fornece à aplicação uma desvantagem. Contudo esta aplicação consegue suportar várias linguagens de programação se estas permitirem a compilação a partir da linha de comandos (para que seja possível produzir um ficheiro executável).

Inicialmente esta aplicação não tinha um mecanismo de avaliação automática sendo necessário responsáveis para desempenhar algumas tarefas como recompilar o programa do estudante, executá-lo, analisar o código-fonte e/ou os resultados de execução e ainda enviar uma mensagem com o *feedback* para o respectivo estudante. Mas, como este processo de avaliação requer enormes esforços da parte do responsável, surgiu uma nova versão do PC², a versão 9.1. Esta nova versão já

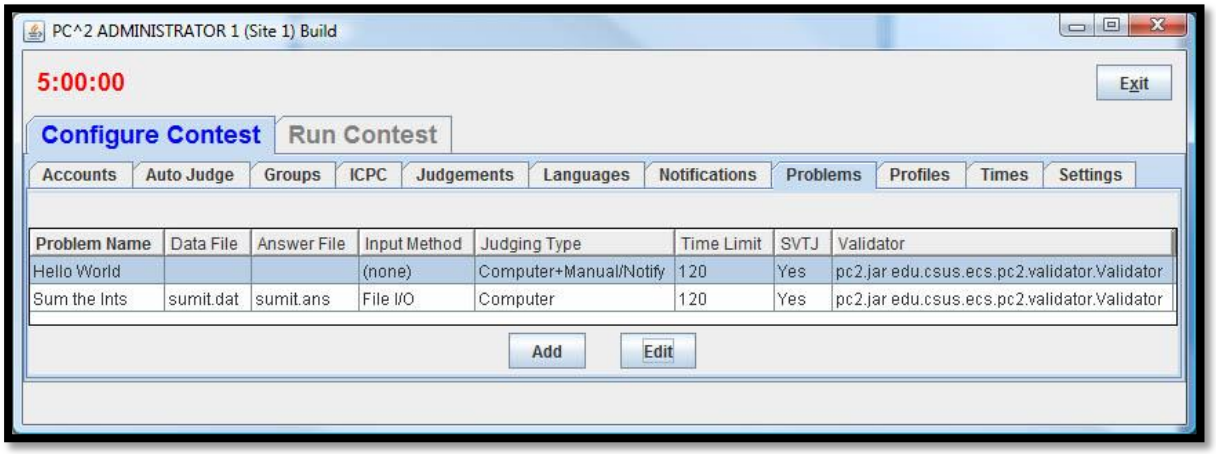
[4] http://www.um.es/

permite avaliação automática (Ashoo, et al., 2011) que poderá ser configurada previamente ao problema, ou seja, alguns problemas poderão ser avaliados de forma automática e outros poderão ser avaliados de forma manual.

Este sistema apresenta uma arquitectura cliente-servidor, em que é necessário uma ligação à rede local para que o estudante possa aceder aos problemas propostos. Depois da ligação estar estabelecida, os enunciados são automaticamente carregados para a aplicação do estudante (cliente) e, quando o exercício está resolvido pelo estudante, a resposta ao problema será enviada para a aplicação responsável pela avaliação (servidor) através da comunicação existente.

Os problemas podem ser armazenados na aplicação do servidor (ver figura 03) para mais tarde serem carregadas na aplicação do estudante. Neste interface também temos a informação da forma como o problema será avaliado. Apesar deste sistema permitir avaliação automática de algoritmos, não analisa o código-fonte, verificando apenas os dados de *output* que são produzidos pela solução submetida, através de um programa desenvolvido separadamente que aceita como entrada os dados produzidos pela solução do estudante.

**Figura 03**: Interface do sistema PC$^2$




Fonte: (Ashoo, et al., 2011)

À semelhança do Mooshak (secção anterior), este sistema também apresenta algumas características importantes para a elaboração do projecto, entre as quais: definição de problemas; armazenamento de dados e avaliação automática de código-fonte com base nos dados produzidos pelo algoritmo do estudante. No entanto, o PC$^2$, para além de suportar apenas a avaliação de algoritmos através dos dados produzidos pela resposta do estudante (o que não é a forma ideal de avaliação), também apresenta uma grande desvantagem em estar desenvolvido sob uma ferramenta que não é propícia para a Web, sendo necessário cada aluno transportar a aplicação para que possa visualizar e responder a cada exercício proposto pelo docente.

## *2.4 UVa Judge e EduJudge*

O sistema UVa Judge[5], *University of Valladolid Online Judge*, foi desenvolvido por um estudante de Informática da Universidade de Valladolid: Ciriaco García de Celis (Revilla, et al., 2008). Começou por ser desenvolvido na linguagem de programação Shell Script, contudo, estes *scripts* não eram verdadeiramente aconselháveis para integrar um sistema de avaliação automática uma vez que não era possível limitar a memória que seria usada pela resposta do estudante e acima de tudo não produzia relatórios sobre o estado de cada evento.

Este sistema foi modificado para suportar não só competições de programação como também permitir ao estudante estudar autonomamente os exercícios, pondo em prática os seus conhecimentos. As informações ficam armazenadas numa base de dados SQL, para um carregamento posterior através da linguagem de programação PHP, permitindo a actualização dos ficheiros. Com as modificações completas, este sistema também se tornou num sistema Web.

O EduJudge[6] pode ser incorporado em processos de aprendizagem de cursos de matemática e programação (Leal, et al., 2009) tendo como público-alvo estudantes do ensino secundário e ensino superior. Este sistema tem como principal objectivo aderir ao sistema de avaliação do UVa Judge, e inserir os seus problemas num ambiente educacional mais eficaz, permitindo melhorar o desempenho dos estudantes.

A arquitectura deste sistema tem suporte a três componentes - *Learning Objects Repository* (responsável pela parte da gestão e partilha, em que o docente armazena as atribuições fornecidas aos alunos), *Learning Management System* (permite a ligação a diversos sistemas de *e-learning* como o Moodle, Dokeos, etc.) e *Evaluation Engine* (responsável por realizar a avaliação automática das soluções apresentadas pelos alunos) - o que permite um conjunto específico de serviços. No entanto, pretende-se criar e implementar um repositório de problemas de programação e inseri-lo num outro sistema (i.e. sistemas que permitam a gestão de competições de programação), com a finalidade de obter um conjunto mais abrangente de serviços (Leal, et al., 2009).

O UVa Judge e EduJudge apresenta uma característica importante para esta dissertação de mestrado: suporta cursos do ensino superior e cursos do ensino secundário. Este sistema também apresenta outras características, nomeadamente o uso de uma linguagem de programação PHP para suportar um sistema disponível 24 horas por dia (à semelhança do Mooshak - secção 2.2) e o uso de base de dados relacionais utilizando o SQL para armazenar dados.

[5] http://uva.onlinejudge.org/
[6] http://www.edujudge.eu/

## *2.5 SICAS*

O SICAS, *Interactive System for Algorithm Development and Simulation*, é um sistema para a aprendizagem de conceitos básicos de programação (Bravo, et al., 2005) e foi implementado na Linguagem de programação Java. O objectivo principal deste sistema traduz-se em desenvolver competências de alunos, através de respostas a diversos problemas, nomeadamente na utilização de estruturas de controle e subprogramas para resolver um problema específico. Apresenta duas formas de funcionamento, estando uma direccionada para o estudante (em que permite construir algoritmos e executá-los, analisando cada linha do código) e outra para o professor. Ambas as formas são semelhantes no seu funcionamento, contudo acresce às funcionalidades do professor a possibilidade de adicionar problemas com os respectivos dados de teste para fornecer ao aluno a capacidade de verificar se as suas respostas estão correctas.

Quando um professor define um problema através de um enunciado, define também uma ou mais resoluções e poderá adicionar valores de entrada e valores de saída. Estes valores podem ser necessários para permitir ao aluno fazer a sua auto-avaliação dos conhecimentos, isto é, depois do estudante concluir a sua resposta ao problema, poderá inseri-la na aplicação e receber posteriormente um *feedback*. Este *feedback* é o resultado da comparação dos dados de *output* produzidos pela algoritmo do estudante (através dos dados de *input*) com os dados de *output* definidos pelo docente. Este mecanismo permite preparar melhor o aluno para os exercícios que são propostos na avaliação do plano curricular. As respostas aos problemas ficam armazenadas no sistema para que, numa fase posterior, o docente possa verificar, em caso de dúvida, as respostas dos alunos.

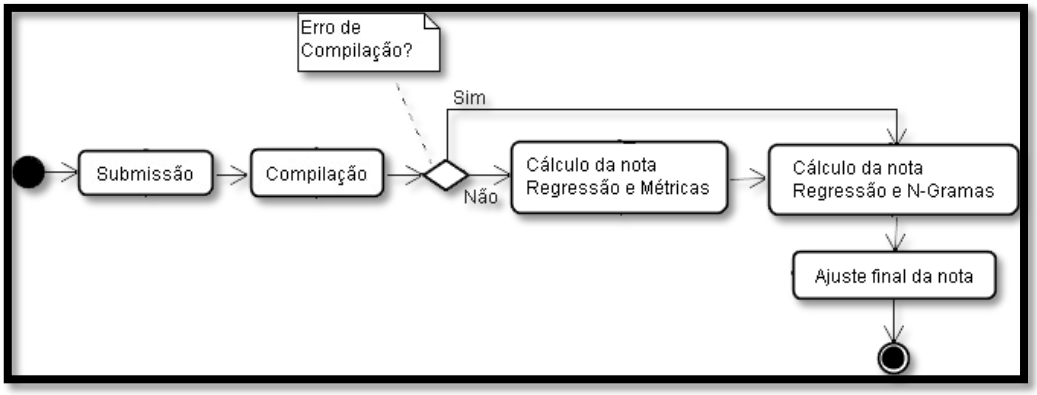
Este sistema, para além de efectuar avaliações de acordo com os dados de *output*, também fornece ao estudante a possibilidade de observar, analisar e simular a forma como um determinado algoritmo funciona. A simulação também pode ser analisada de forma animada e interactiva (Gomes, et al., 2004) através da presença de fluxogramas onde permitem a atribuição, a entrada/saída de valores, a repetição e a selecção como estruturas. Após o sistema implementado e todas as suas funcionalidades disponíveis, os docentes utilizaram-no nas suas aulas (Mendes, et al.) com o objectivo de melhorar o processo de aprendizagem.

Esta ferramenta apresenta a desvantagem de não funcionar em ambiente Web, contudo poderá ser utilizada em aulas de Introdução à Programação pelos docentes. Esta aplicação para além da avaliação automática através da comparação dos dados produzidos pelo algoritmo do estudante com os dados definidos pelo docente inicialmente também permite armazenar problemas com os respectivos dados de teste, permitindo ao aluno resolvê-los da melhor forma possível.

## *2.6 Módulo Desenvolvido para o Moodle*

Este ambiente é integrado com o sistema Moodle[7] através do desenvolvimento de um módulo com base em *Web Service*[8]. Contém um servidor Java que espera por uma solicitação do cliente PHP[9]. Quando o estudante submete uma resposta, esta é enviada para o servidor e submetida a vários processos (figura 04). Quando o servidor obtém uma resposta automática, envia-a para o Moodle e fica assim disponível para o estudante.

**Figura 04**: Processos que o sistema executa após receber uma resposta do estudante




Fonte: (Moreira, et al., 2009)

Este módulo propõe dois modelos de regressão linear que permitem facilitar e automatizar a avaliação de algoritmos nos cursos de programação. Um dos modelos é composto por métricas de engenharia de software sendo o outro composto por semelhanças que existem nos algoritmos e é baseado na pesquisa de *n* caracteres (sendo *n* uma variável que pode assumir os valores 1, 2 ou 3). Este último modelo permite comparar um conjunto de caracteres num texto ou numa palavra identificando no final quantos conjuntos de caracteres estão partilhados no texto do estudante e do docente. Com estes dois mecanismos, o módulo consegue atribuir facilmente as notas aos estudantes e tem como principal objectivo a análise e comparação da resposta do estudante e do docente, isto é, verifica até que ponto a solução do estudante é idêntica à solução do docente.

Este módulo permite também ao estudante submeter várias respostas permitindo-lhe a melhoria constante do seu algoritmo e a possibilidade de chegar à solução ideal fornecida pelo docente, o que facilita bastante a tarefa do docente uma vez que não terá que analisar algoritmo a algoritmo. Normalmente os algoritmos são avaliados de acordo com os dados que produzem, tendo como entrada um conjunto de valores pré-definidos pelo docente, e pela sua complexidade.

[7] *Modular Object-Oriented Dynamic Learning Environment* - http://moodle.org/
[8] Solução que permite a comunicação entre sistemas diferentes
[9] *PHP: Hypertext Preprocessor*, originalmente *Personal Home Page*

A regressão linear múltipla é uma técnica que está a ser cada vez mais utilizada na avaliação automática de algoritmos, apresentando bons resultados (Hearst, 2002). A utilização de algumas métricas de engenharia de software como a quantidade de funções utilizada, o número de palavras reservadas, o número de declarações, o número de linhas do algoritmo, entre outras, é uma das muitas técnicas que podem ser utilizadas para obter resultados satisfatórios.

Este módulo para o Moodle foi testado com 20 exercícios de programação onde cada exercício foi seleccionado tendo em conta a presença de estruturas de controle, estruturas de repetição, declarações de variáveis, uso de funções e respostas com um tamanho razoável. Foram criadas 5 respostas para cada exercício (estando apenas uma com a cotação máxima) para que pudessem ser comparadas com a solução do estudante. Depois de concluídos os problemas que foram propostos para testar este módulo, verificou-se que o grau de aproximação à nota real (atribuída pelo docente manualmente) foi bastante satisfatório, apresentando 90,42% para métricas de engenharia de software e 95,22% pela comparação de um conjunto de caracteres (Moreira, et al., 2009).

Com o final do estudo, conclui-se que ambos os modelos podem ser utilizados pela aproximação dos valores, contudo e para que o resultado não fique comprometido pela complexidade do algoritmo, dever-se-á combinar o valor dos modelos, ficando no final com uma precisão de 92,82% à nota atribuída manualmente pelo docente.

Este mecanismo apresenta características fundamentais que devem fazer parte da dissertação de mestrado. O uso de duas técnicas que permitem uma avaliação automática do código-fonte produzido pelo estudante é fundamental para equilibrar a nota final atribuída pelo sistema e assim conseguir atribuir ao docente um *know-how* das capacidades adquiridas pelo estudante. Para além disto e à semelhança dos sistemas já analisados em secções anteriores, verificamos que este mecanismo, por estar incorporado num sistema colaborativo (i.e. Moodle), atribuí ao estudante a possibilidade de submeter várias respostas através da utilização de um *browser* de internet.

## *2.7 SAmbA*

Este sistema, SAmbA, pode ser utilizado recorrendo a um *browser* de internet onde integra uma base de dados para armazenar comentários, problemas e respectivas soluções. O aluno pode utilizar esta base de dados para melhorar a sua capacidade de desenvolvimento de algoritmos, respondendo a cada problema a um ritmo desejado.

Quando este sistema foi desenvolvido, diversas estratégias foram definidas com vista a melhorar a aprendizagem individual de cada estudante. Estas estratégias são baseadas essencialmente na distribuição das respostas de cada estudante por outros, permitindo analisar todas as respostas; comparar soluções manualmente pelos estudantes, permitindo melhorar o seu algoritmo da melhor

forma e identificando erros noutros algoritmos; fornecer várias soluções ao sistema para que seja considerada a mais correcta; entre outras (Nobre, et al., 2002).

O SAmbA permite dois modos de funcionamento: estudante e docente. No modo estudante podem ser submetidas soluções individuais ou em grupo que ficam armazenadas na base de dados do sistema para posterior acesso e reutilização do algoritmo; pode existir comparação e análise de soluções, isto é, permite a troca de informações entre estudantes ou entre grupos; pode existir uma comunicação em tempo real com outros estudantes ou até mesmo com o docente, permitindo esclarecer dúvidas pontuais de exercícios, entre outros. Já no modo do docente, este sistema permite que sejam preparadas as aulas e problemas; permite organizar grupos de estudantes de acordo com o limite máximo de alunos; permite seleccionar um conjunto de problemas de uma base de dados para preparar melhor os estudantes sobre uma determinada matéria e permite também a selecção de soluções para uma posterior comparação ao algoritmo do estudante.

Este sistema apresenta funcionalidades que permitem ao estudante um maior desempenho na resolução de exercícios fornecidos pelo docente. O facto deste sistema estar desenvolvido para a Web é, por si só, uma característica fundamental no uso de ferramentas que dão suporte à aprendizagem, por estarem disponíveis 24 horas por dia. O armazenamento de enunciados e respostas dos vários exercícios que o docente disponibiliza, também se torna importante incorporar em sistemas que visam melhorar o processo de avaliação, pois permitem ao aluno a reutilização de código já produzido por este. Uma das características que este sistema apresenta e que não o favorece como sendo um sistema ideal para avaliação de código-fonte, é a partilha interna de soluções entre alunos.

## *2.8 Portugol IDE*

O Portugol IDE[10] foi desenvolvido na Escola Superior de Tecnologia de Tomar do Instituto Politécnico de Tomar e está implementado na linguagem de programação Java. Este sistema está totalmente em português e foi desenvolvido com o principal objectivo de captar o raciocínio algorítmico do estudante. Esta ferramenta suporta a codificação de algoritmos através de pseudo-código e uma linguagem gráfica baseada em fluxogramas.

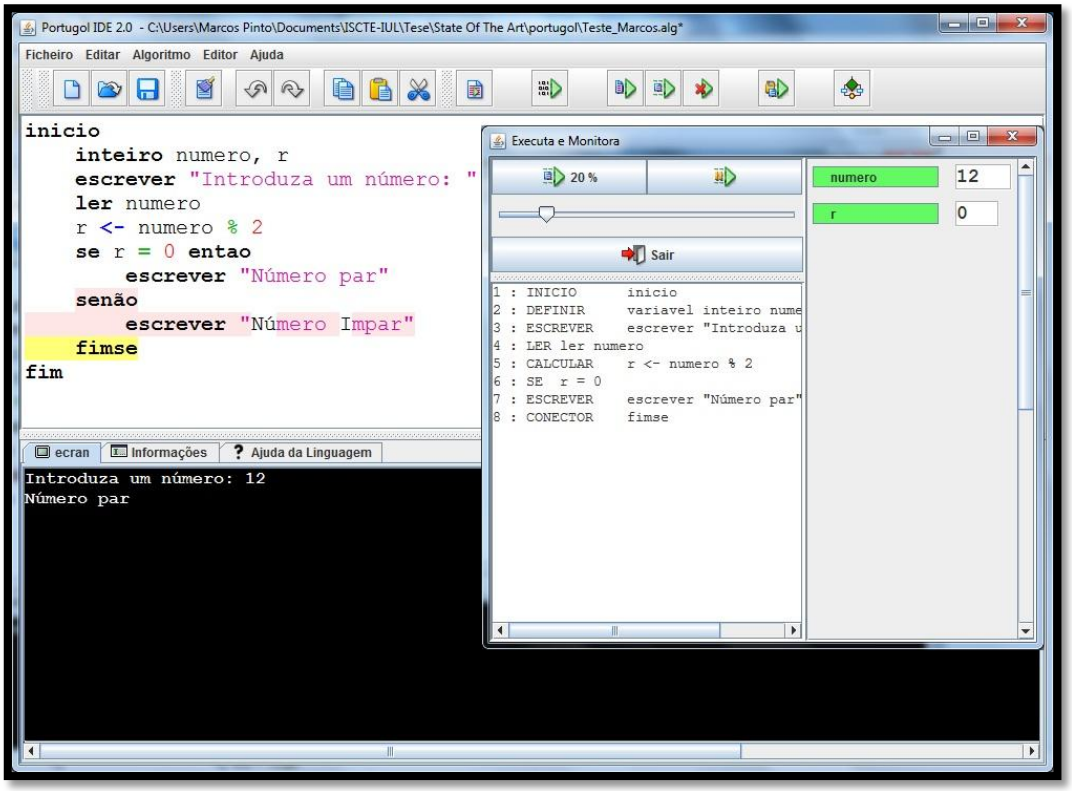
O estudante pode desenvolver os seus algoritmos através deste IDE e executá-los posteriormente, permitindo analisar de forma crítica cada passo do código. Este ambiente facilita o estudante a compreender, de uma forma mais eficaz, a lógica computacional e permite-lhe transitar para ambientes de programação mais tradicionais como a linguagem de programação Java ou C. O Portugol IDE não permite avaliação automática de algoritmos mas disponibiliza um conjunto de

[10] http://www.dei.estt.ipt.pt/portugol/

ferramentas pedagógicas e inovadoras para a aprendizagem de técnicas de programação (Manso, et al., 2011).

Uma das técnicas inovadoras que este sistema disponibiliza é a análise de algoritmos através de fluxogramas possibilitando a cada instância verificar qual o valor das variáveis. Uma outra técnica que ajuda bastante os estudantes transmite-se pela conversão do pseudo-código em fluxogramas. Com isto, o estudante compreende facilmente as instruções que o Portugol IDE permite executar (i.e. ler, escrever, repete até, etc. - ver figura 05) podendo melhorar o seu desempenho nos exercícios futuros.

**Figura 05**: Interface Portugol IDE




Fonte: Marcos Pinto (2012)

No estudo que foi efectuado aos alunos do primeiro ano da licenciatura em engenharia informática - Escola Superior de Tecnologia de Tomar do Instituto Politécnico de Tomar - (Manso, et al., 2011) verificou-se que este sistema é prático e fácil de utilizar.

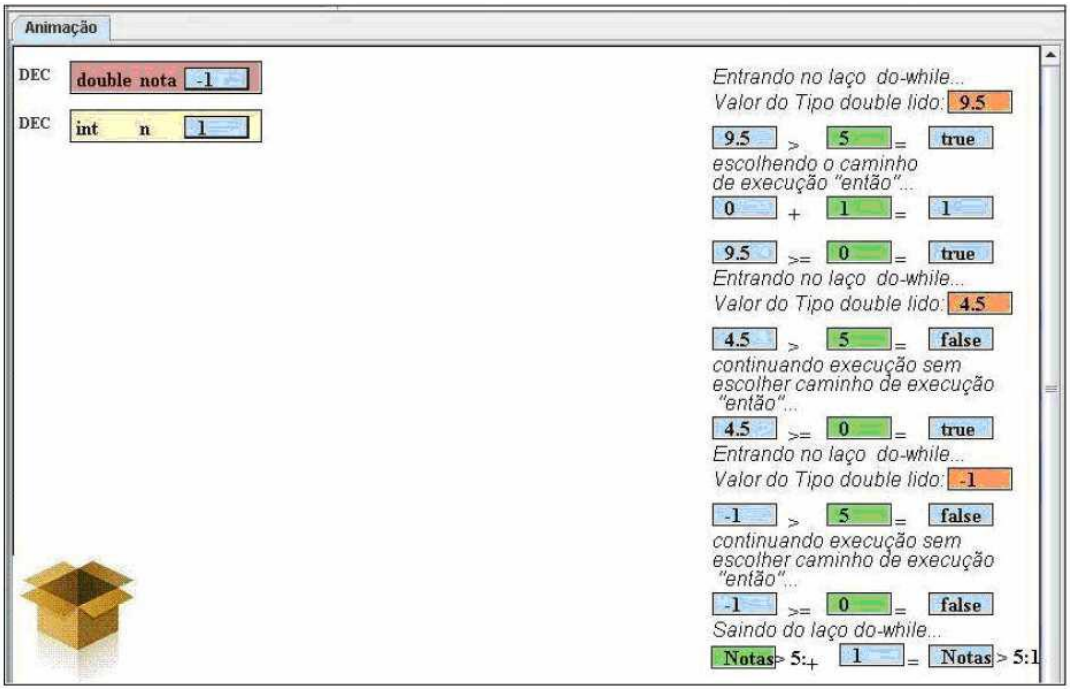
Este sistema, apesar de não efectuar avaliação automática de código-fonte, permite analisar o funcionamento do algoritmo produzido pelo aluno, verificando passo-a-passo cada linha de código. Para este projecto é importante reter a capacidade que este sistema tem em melhorar o desenvolvimento dos algoritmos produzidos pelo estudante o que, de certa forma, irá permitir melhorar os conceitos e técnicas de algoritmia de cada estudante. O Portugol IDE, por ser um IDE, avalia a sintaxe do algoritmo produzido pelo estudante, verificando se existe algum erro de compilação.

## *2.9 JavaTool*

O JavaTool é um IDE, desenvolvido em Java, vocacionado para estudantes que frequentam as disciplinas de Introdução à Programação. O principal objectivo desta ferramenta é facilitar o ensino da linguagem de programação Java e deverá ser integrada num ambiente Web (i.e. Moodle), podendo ser utilizada como ferramenta de trabalho virtual. Quando esta ferramenta estiver na internet o estudante poderá analisar o algoritmo, seleccionar exercícios propostos pelo docente e produzir uma solução para submeter no sistema. Enquanto IDE, o JavaTool apenas transmite para o estudante mensagens de erro na compilação e execução.

Com esta ferramenta o aluno pode escolher uma série de funcionalidades como editar o código-fonte, compilar algoritmos e analisar o resultado ou até mesmo verificar a execução através de uma animação, verificando o valor de cada variável (ver figura 06).

**Figura 06**: Resultado da execução do algoritmo utilizando o JavaTool




Fonte: (Mota, et al., 2008)

A animação da execução dos algoritmos capta mais a atenção dos estudantes (uma vez que permite analisar cada expressão) e por esta razão insere-se como uma característica fundamental para os IDEs adoptados para as disciplinas de Introdução à Programação. Tem incorporada uma análise léxica, sintáctica e semântica que permite ao estudante encontrar facilmente os erros presentes no algoritmo. A sintaxe que o JavaTool utiliza é bastante reduzida (Mota, et al., 2008) comparada com a sintaxe global da linguagem de programação Java o que facilita o desenvolvimento dos primeiros algoritmos do estudante.

Este sistema para além de ser um IDE, pode ser incorporado num sistema Web, permitindo atribuir não só uma disponibilidade de 24 horas por dia ao estudante, como também permitir o acesso centralizado à informação. Uma das características fundamentais presentes neste sistema é a análise sintáctica, léxica e semântica dos algoritmos, pois, permite ao estudante, ter acesso a informações de erros no código-fonte produzido fazendo assim a sua auto-análise. O JavaTool foi pensado para alunos que frequentam aulas que requerem os conceitos básicos de programação, possibilitando ao aluno aprender, da melhor forma, os conceitos e técnicas iniciais de uma linguagem de programação.

## *2.10 JEduc*

O JEduc foi desenvolvido na linguagem de programação Java e é uma ferramenta de código aberto. Foi desenvolvido com o principal objectivo de reduzir os problemas que as linguagens de programação Java apresentam para os estudantes de Introdução à Programação reduzindo algumas características e funcionalidades. Com isto, os estudantes têm apenas as funcionalidades básicas para o desenvolvimento dos algoritmos introdutórios melhorando assim a sua concentração e desempenho.

Esta ferramenta é um IDE que permite um conjunto de características como um editor de programas, análise sintáctica, tratamento de mensagens, assistente para o docente (o que permite visualizar um conjunto de exercícios didácticos e adaptá-los ou usá-los durante as aulas), entre outras. No entanto, não faz avaliação automática de algoritmos. Tal como o eclipse[11], o JEduc também permite a inserção de código automático (i.e. if/else, while/do, etc.) através de *wizards* (inserção de código automaticamente) para que o estudante aplique de forma correcta algumas instruções, ficando no final com o conhecimento das diversas sintaxes.

Em disciplinas de Introdução à Programação o uso de ambientes de desenvolvimento não devem ser complexos (Parego, 2002). Para além do professor ter que explicar cada mecanismo do IDE, o aluno perde a concentração no que é realmente importante e tenta descobrir novos mecanismos que o ambiente permite.

Apesar do JEduc não permitir a atribuição de uma nota final aos algoritmos produzidos pelo estudante, devolve uma mensagem de erro, caso exista algum erro de compilação no código-fonte, o que atribui a esta ferramenta uma característica importante. Apesar de ser uma ferramenta para desenvolver novos algoritmos, está restrita às funcionalidades básicas da linguagem de programação Java, possibilitando ao estudante um foco superior nos principais conceitos (i.e. declarar variáveis, chamada de funções e procedimentos, etc.) de programação.

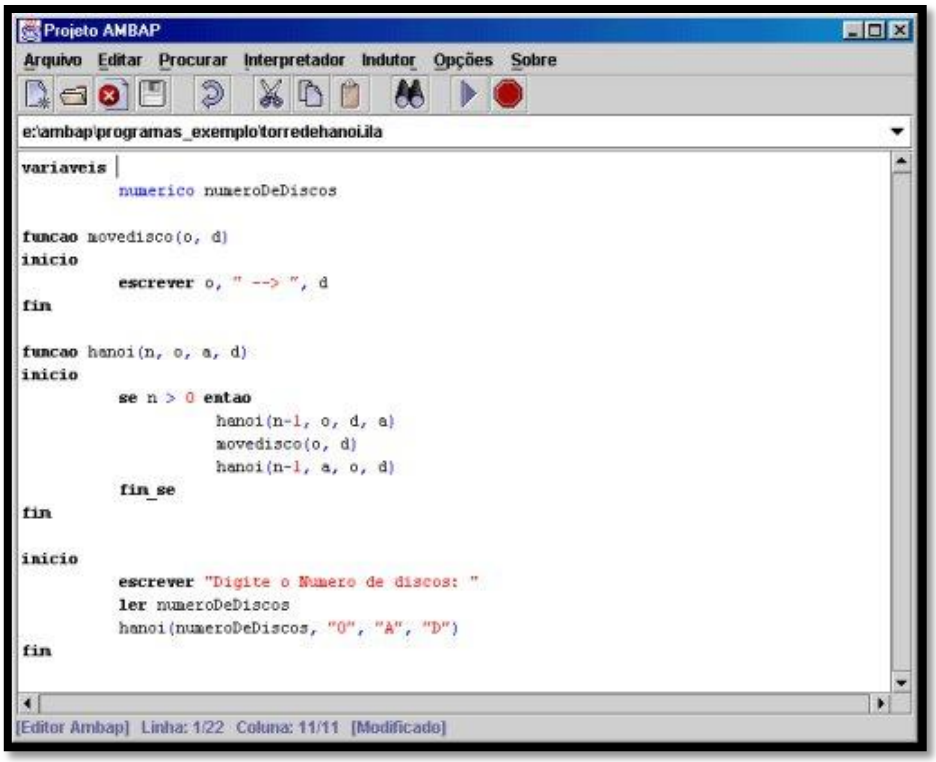
[11] http://www.eclipse.org/

## *2.11 AMBAP*

O AMBAP[12] foi desenvolvido no Centro de Ciências Exactas e Naturais do Departamento de Tecnologias de Informações da Universidade Federal de Alagoas e permite ao estudante desenvolver os seus conceitos e técnicas de programação abrangendo desde a fase de compreensão do problema até à sua resolução através de um algoritmo. Este sistema foi desenvolvido em Java e é útil para estudantes de Introdução à Programação onde aprendem o conceito de variáveis, funções, métodos e recursividade.

Assemelha-se a um IDE e por esta razão não faz avaliação automática de algoritmos com atribuição de nota. Contudo, permite ao estudante utilizar uma linguagem algorítmica simples (figura 07) possibilitando a sua execução e análise. Como em todos os projectos, este sistema também teve um levantamento de requisitos detalhado. A interacção, configuração, representações alternativas, recursos animados, simplicidade, sistema intuitivo e portabilidade (Almeida, et al., 2011) são requisitos que predominam neste sistema o que facilita a motivação para a aprendizagem autónoma do estudante.

**Figura 07**: Interface AMBAP




Fonte: http://www.ufal.br/tci/ambap/images/AmbapScreenShot1.jpg

Mais tarde será implementado uma nova versão deste sistema, permitindo ter uma arquitectura cliente-servidor onde centraliza diversas informações como o perfil dos alunos e estratégias

[12] http://www.ufal.br/tci/ambap/

pedagógicas. Com isto, o professor poderá acompanhar cada estudante e perceber as dificuldades encontradas nos problemas.

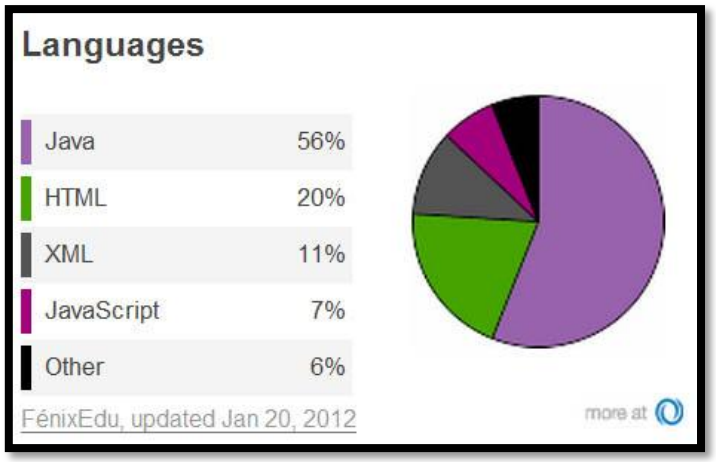
Este sistema pode ser utilizado nas primeiras aulas de Introdução à Programação, permitindo ao estudante perceber melhor os conceitos iniciais da programação. O AMBAP foi objecto de estudo e verificou-se que os sistemas, para dar suporte à aprendizagem, requerem um conjunto de requisitos fundamentais como a interação, simples e intuitiva, o que, de certa forma, assemelha-se a este projecto.

## *2.12 Fénix*

O sistema Fénix foi desenvolvido no Instituto Superior Técnico da Universidade Técnica de Lisboa e permite a gestão de informações dos respectivos cursos através do *browser* de internet. Teve uma expansão para outras Universidades pelo sucesso que obteve. Actualmente é utilizado por vários alunos e professores para gerirem as informações dos respectivos cursos. Contudo este sistema também permite efectuar uma gestão dos cursos, inscrições, gestão de horários, renovar matrículas, agrupar conteúdos por zonas, fornecer avisos para os utilizadores, entre outros.

Através da figura 08 podemos verificar que o Java é a linguagem de programação predominante neste sistema. Utiliza uma base de dados MySQL para armazenar dados persistentes. Utiliza também uma *Framework*[13] que ajuda no desenvolvimento de aplicações Java em que é necessário um modelo de domínio transaccional e persistente (Software, 2012).

**Figura 08**: Linguagens de programação utilizadas no sistema Fénix




Fonte: http://fenix-ashes.ist.utl.pt

Este sistema, apesar de não permitir avaliação automática de código fonte produzido por alunos, permite armazenar dados numa base de dados MySQL, gerir diversos conteúdos e alunos, entre

[13] Estrutura que junta projectos fornecendo uma funcionalidade genérica

outras. É importante considerar estas características aquando da implementação de um novo sistema de avaliação automática, pois, com isto, é possível apresentar um sistema bastante completo para que seja utilizado não só pelos alunos, como pelos docentes.

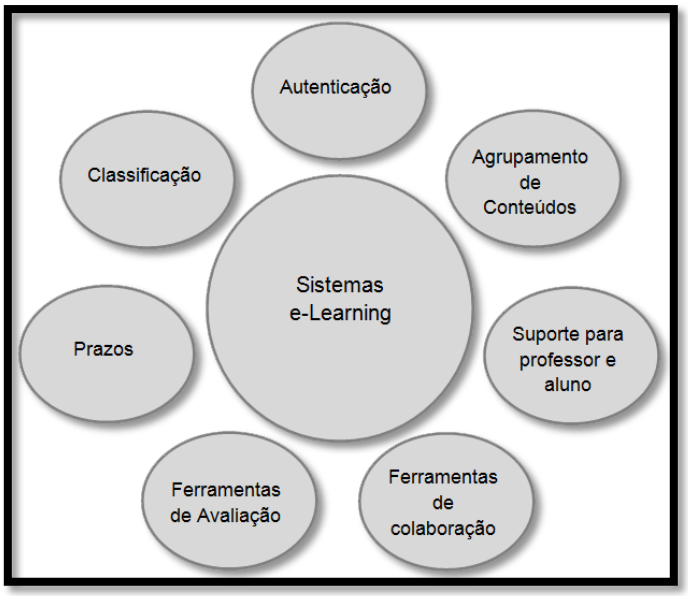
## *2.13 Sistemas e-Learning*

Os sistemas *e-Learning* ou sistemas de gestão de aprendizagem foram desenvolvidos com o principal objectivo de ajudar na aprendizagem através da internet. Os conteúdos são fornecidos ao estudante através de sistemas Web e podem ser entregues em qualquer formato (e.g. áudio, vídeo, texto, etc.). Estes sistemas são baseados essencialmente na organização da informação, permitindo a cada estudante expandir os seus conhecimentos a qualquer altura.

Existem vários sistemas que suportam a aprendizagem à distância, como o Moodle, o Blackboard Learning System[14], o Dokeos[15], o Sakai[16], o Atutor[17], o RCampus[18], entre outros. Todos estes sistemas são baseados no LAMS pois permitem que exista uma sequência do estudo da parte do estudante (Dalziel, 2003). Também suportam a reutilização de conteúdos de anos anteriores.

Através da figura 09 podemos verificar algumas das características que estes sistemas partilham. No entanto estes sistemas não suportam avaliação automática suportando apenas a entrega de ficheiros num período de tempo definido pelo professor.

**Figura 09**: Características dos sistemas e-Learning




Fonte: Marcos Pinto (2011)

[14] http://www.blackboard.com/
[15] http://www.dokeos.com/
[16] http://sakaiproject.org/
[17] http://atutor.ca/
[18] http://www.rcampus.com/

O sistema e-Learning apresenta um conjunto de características que são essenciais para um projecto cujo objectivo é desenvolver um mecanismo capaz de avaliar o código-fonte do aluno de uma forma automática. Este tipo de sistema, para além de estar desenvolvido para a Web, também apresenta funcionalidades que os sistemas de avaliação automática devem conter, nomeadamente a presença de um formulário que permite a autenticação do estudante, definir prazos de entrega das respostas do estudante, etc.

## *2.14 Conclusões*

São poucos os mecanismos que efectuam avaliação automática de código-fonte produzido pelo aluno. Contudo, neste capítulo, foram apresentados alguns sistemas que utilizam métodos simples para produzirem um *feedback* automático para o estudante. Esse *feedback* geralmente é atribuído através da comparação dos dados produzidos pelo algoritmo do aluno com os dados previamente estabelecidos pelo docente.

O processo de avaliação que os sistemas actualmente usam, embora seja necessário e relevante, é bastante insuficiente, pois não fornecem uma visão clara da resposta do aluno, ou seja, não analisam o código-fonte nem o seu processo de funcionamento. Estes sistemas, apesar de úteis para o docente, não conseguem utilizar métodos abrangentes que permitam utilizar módulos de avaliação com o objectivo de extrair as informações relativas à resposta fornecida pelo estudante.

Com base nas limitações que o processo de avaliação apresenta perante este tipo de sistemas, este trabalho visa desenvolver um método mais avançado de análise do código-fonte e, assim, proporcionar uma avaliação precisa de todo o algoritmo fornecido pelo aluno. Assim, o módulo responsável pela avaliação automática do algoritmo do aluno incorpora o processo de avaliação actual dos sistemas (comparação dos dados de entrada e saída do código-fonte) e também, o uso de métricas de engenharia de software através da comparação do algoritmo do aluno com uma possível solução fornecida pelo docente. Os dois modos de avaliação presentes neste novo módulo irão proporcionar ao docente uma visão completa da qualidade da solução do estudante.

# Capítulo III - Conceptualização do Sistema

## *3.1 Introdução*

Este capítulo apresenta a arquitectura utilizada no sistema com a finalidade de obtermos, não só, uma percepção do seu comportamento funcional, como também, da localização dos componentes de software e as formas de comunicação entre eles. É também apresentado, neste capítulo, a forma como o módulo responsável pela avaliação do algoritmo produzido pelo estudante funciona.

Para que o sistema funcione sem recurso a aplicações instaladas no computador do aluno, é essencial que funcione numa arquitectura cliente-servidor, garantindo uma disponibilidade ao estudante e ao docente de 24 horas por dia.

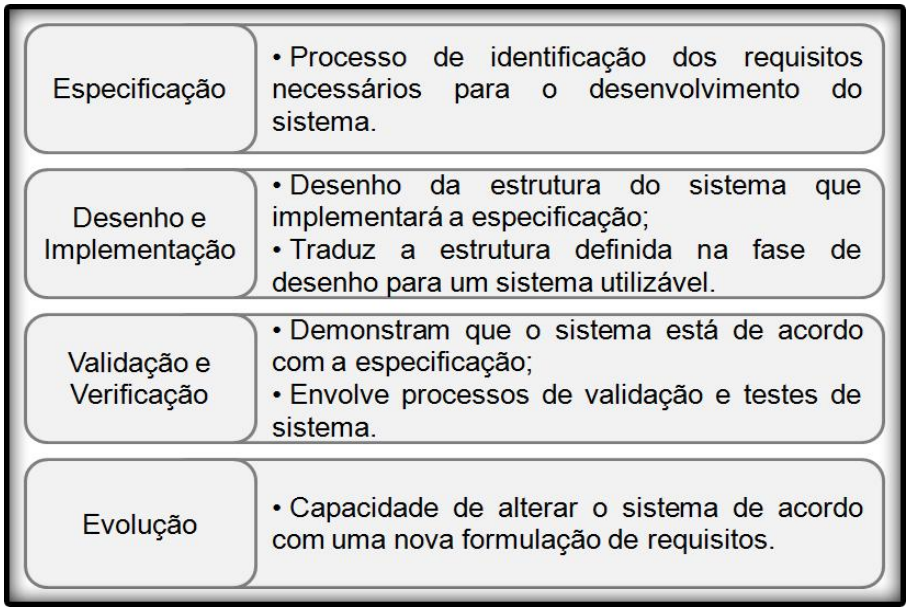
Neste capítulo está apresentado uma análise de requisitos ao sistema (secção 3.2) entre os quais podem ser divididos em requisitos funcionais (sub-secção 3.2.1) e requisitos não funcionais (sub-secção 3.2.2). É também apresentado a caracterização do sistema Web (secção 3.3), sobre a qual está definida a sua arquitectura (sub-secção 3.3.1) e as camadas do sistema (sub-secção 3.3.2). Está também apresentada a caracterização do módulo de avaliação (secção 3.4), onde está presente a sua arquitectura (sub-secção 3.4.1) e a caracterização do processo de avaliação (sub-secção 3.4.2). O processo de avaliação utiliza testes JUnit (sub-secção 3.4.2.1), métricas de engenharia de software (sub-secção 3.4.2.2) e o conjunto de todas as expressões utilizadas (sub-secção 3.4.2.3) na nota final atribuída ao aluno.

## *3.2 Análise de Requisitos do Sistema*

A análise de requisitos é um processo fundamental no desenvolvimento de um novo sistema. Como o próprio nome indica, a análise de requisitos transmite-se numa descrição rigorosa das características de um sistema com a finalidade de funcionar de acordo com um objectivo esperado.

Um requisito é uma condição cuja exigência deve ser satisfeita por parte de um sistema. Se a condição é produzir algo, então, estamos perante um requisito chamado funcional. Caso contrário, se a condição é caracterizar algo (atributo, comportamento, propriedade, restrições, etc.), estamos perante um requisito denominado não funcional. A descrição dos requisitos, apesar de ser um dos processos iniciais de cada projecto (figura 10), é uma actividade indutiva e requer análises constantes, ou seja, a descrição dos requisitos deve estar presente ao longo do desenvolvimento de todo o sistema para que este responda ao problema inicial.

**Figura 10**: Actividades do Processo de um Sistema




Fonte: Marcos Pinto (2012)

Para implementar um sistema de avaliação automática de código-fonte produzido por estudantes, é fundamental fazer um levantamento de requisitos. Esta secção apresenta os requisitos essenciais do sistema e estão organizados consoante dois tipos: funcionais (sub-secção 3.2.1) e não funcionais (sub-secção 3.2.2).

### 3.2.1 Funcionais

Os requisitos funcionais definem, de uma forma geral, funções e/ou componentes que um determinado sistema realiza, especificando resultados particulares. Estes requisitos são responsáveis por definir o comportamento do sistema consoante os dados de entrada fornecidos pelo utilizador. Com base nisto, os requisitos funcionais que o sistema de avaliação automática de código-fonte apresenta são:

- **Autenticação:** O sistema fornece uma percentagem da avaliação global de cada estudante e, como tal, torna-se necessário englobar no sistema um processo de autenticação de cada utilizador. Sendo a autenticação um processo que visa verificar a identidade de cada utilizador no sistema, o docente pode verificar, não só, a evolução de cada estudante, como também, pode verificar qual ou quais os conceitos que necessitam de ser melhorados, apresentando no final exercícios que foquem essas matérias.

- **Adicionar novas unidades curriculares:** Apesar do projecto estar centralizado para unidades curriculares de Introdução à Programação, deve ser capaz de incorporar novos módulos que permitam a avaliação automática de exercícios de outras unidades

curriculares. Com isto, o sistema não fica restrito apenas à unidade curricular de Introdução à Programação, podendo dar suporte a outras.

- **Avaliação automática individual:** Este sistema permite aliviar parte do processo de avaliação da unidade curricular de Introdução à Programação, ajudando nas tarefas do docente na atribuição das notas individuais para um determinado exercício. Através desta funcionalidade do sistema o docente insere novos exercícios que visam a melhoria contínua das capacidades e técnicas de programação dos alunos.

- **Definir prazos de disponibilidade dos exercícios:** Para que os exercícios fiquem disponíveis apenas quando a matéria é leccionada, torna-se importante esta característica. Com isto, é possível para o docente atribuir um conjunto de exercícios no início da unidade curricular permitindo adicionar exercícios esporádicos para colmatar os conhecimentos de cada estudante.

- **Definir prazos de entrega dos exercícios:** À semelhança da definição dos prazos da disponibilidade dos exercícios, também se torna importante adicionar, a cada exercício, um prazo de entrega. Estes prazos para entrega das respostas aos exercícios propostos vão permitir ao docente analisar a evolução dos alunos, fornecendo-lhe a possibilidade de adicionar mais exercícios que vão ao encontro das dificuldades dos estudantes.

- **Submissão de ficheiros:** O sistema deve fornecer, a cada aluno, a possibilidade de entregar ficheiros para que possa existir uma avaliação automática do algoritmo produzido. Caso o exercício fornecido pelo docente seja apenas resolvido através de uma classe (no caso particular deste trabalho, uma classe Java), o sistema deve também disponibilizar ao aluno a possibilidade de desenvolver o seu algoritmo directamente no sistema, dando origem no final a uma classe.

- **Acesso a submissões de ficheiros anteriores:** Uma vez que o sistema permite a submissão de ficheiros, é importante que também suporte o seu armazenamento. Este requisito é fundamental pois permite ao aluno/docente verificar e analisar o resultado da avaliação automática sempre que desejado. O acesso aos ficheiros submetidos, para cada exercício, também se torna importante pois irá permitir a cada aluno utilizar uma parte do algoritmo já desenvolvido sem recorrer a cópias locais (o aluno terá que fazer o *download* de um algoritmo já submetido para poder analisar o seu código-fonte). Com isto, o aluno pode ter acesso aos ficheiros já submetidos através de qualquer computador, bastando para isso uma ligação à internet.

- **Definir e analisar métricas de engenharia de software:** A avaliação de código-fonte produzido por alunos através de métricas de engenharia de software é uma das formas da

avaliação que o sistema permite. Quando o docente insere um novo exercício, insere também uma resolução (denominada ideal) para uma posterior comparação com o código-fonte submetido pelo aluno. O sistema deve ser capaz de comparar algumas métricas de engenharia de software (secção 3.4.2, sub-secção 3.4.2.2) do código-fonte do aluno com o código-fonte submetido pelo docente. O resultado da análise é visível através do relatório produzido no acto da avaliação automática e está detalhado métrica por métrica.

- **Definir testes para avaliação automática:** O sistema deve permitir a especificação de testes automáticos para cada exercício, garantindo que cada resposta submetida pelo aluno funcione de acordo com o enunciado definido pelo docente. Cada teste deve ser composto pelo valor esperado (definido pelo docente) e pelo valor obtido (através do código-fonte do aluno). O sistema deve ser capaz de avaliar os testes automaticamente mostrando, no final, a quantidade de testes correctos e a nota final atribuída. No final, não é visível para ambas as partes (aluno e docente) qual ou quais ou testes que foram executados correctamente.

- **Executar testes automaticamente:** O principal objectivo deste projecto é avaliar algoritmos automaticamente aliviando algum trabalho do docente. Como tal, é fundamental que cada algoritmo produzido englobe uma série de testes para que o sistema tenha parte da classificação da resposta submetida. Após a avaliação estar concluída, a nota dos testes é visualizada no relatório que o sistema fornece ao utilizador.

- **Definir estratégias de avaliação para cada exercício:** Quando o docente insere novos exercícios no sistema pode definir cada percentagem associada às métricas de engenharia de software e a percentagem associada aos testes. Este requisito é fundamental pois permite ao docente distribuir as percentagens em cada exercício submetido. Esta estratégia é responsável também por definir qual ou quais as componentes que devem ser consideradas na avaliação do código-fonte submetido pelo aluno, ou seja, o docente pode remover algumas componentes responsáveis pela avaliação do estudante através das percentagens que atribui ao exercício. Com isto, o docente pode inserir uma percentagem superior a componentes de avaliação que considere mais críticas na resolução de cada exercício.

### 3.2.2 Não Funcionais

Os requisitos não funcionais, ao contrário dos requisitos funcionais, especificam condições e restrições que devem prevalecer em todo o sistema. Estes requisitos estão normalmente associados a requisitos de desempenho, requisitos da interface do sistema (apresentação dos dados ao utilizador), restrições que o sistema deve utilizar e atributos de qualidade. Os requisitos não funcionais podem ser mais críticos do que os requisitos funcionais pois, caso não sejam

considerados, o sistema torna-se inútil. Com base nisto, os requisitos não funcionais que o sistema de avaliação automática de código-fonte apresenta são:

- **Disponibilidade:** Para que o sistema possa ser utilizado em qualquer local e sem recurso a aplicações, é importante que esteja disponível 24 horas por dia recorrendo a uma página de internet. Com isto a questão da portabilidade não se coloca pois basta ter acesso à internet para poder aceder ao sistema e submeter o código-fonte para um exercício respectivo.

- **Simplicidade:** A simplicidade deve ser uma característica fundamental num novo sistema. Esta característica vai permitir ao aluno/docente focar-se essencialmente nos exercícios e na análise do *feedback* automático, deixando para trás a forma como o sistema se comporta. As características do sistema são intuitivas e não é necessário um grande esforço para perceber como este funciona. Com isto, o sistema fornece ao aluno uma forma simples de efectuar a auto-avaliação das suas capacidades e não uma ferramenta para ser estudada.

- **Escalável:** Este é um requisito que todos os sistemas devem respeitar. Apesar de ser difícil de definir, um sistema escalável permite um aumento do desempenho do sistema. Um factor crítico num sistema de avaliação automática de código-fonte transmite-se no número elevado de utilizadores simultaneamente. Caso o número de utilizadores ultrapasse um determinado limite e exista impacto no desempenho, espera-se que o sistema seja alterado facilmente.

- **Desempenho:** Este requisito é fundamental pois o sistema deverá garantir que o aluno tenha acesso ao *feedback* automaticamente após submeter a sua resposta. Este comportamento também se deve manter mesmo quando o sistema ultrapassa o limite dos utilizadores. No entanto, o desempenho do sistema varia consoante o servidor onde o sistema está hospedado.

- **Segurança:** O sistema deve conter uma autenticação segura não permitindo um utilizador analisar código-fonte submetido por terceiros. Visto que este sistema suporta avaliação de código-fonte produzido por alunos, deve executar cada teste de forma externa ao sistema não comprometendo a sua estrutura. O sistema deve também garantir a confidencialidade de cada teste fornecido pelo docente para os respectivos exercícios.

## *3.3 Caracterização do Sistema*

Com as limitações apresentadas dos sistemas já desenvolvidos (capítulo II) e com os objectivos incorporados neste trabalho de investigação, é apresentada nesta secção a caracterização do

sistema implementado cuja função é avaliar código-fonte Java das respostas fornecidas pelos alunos da unidade curricular de Introdução à Programação. Este sistema tem como foco principal ajudar cada estudante a melhorar as suas capacidades e técnicas de programação, permitindo, a cada estudante, realizar a auto-avaliação autonomamente através de exercícios fornecidos pelo docente. A auto-avaliação do aluno é feita com base num *feedback* produzido automaticamente pelo sistema não sendo necessário existir uma avaliação individual por parte do docente.

Desta forma, os alunos podem manter uma avaliação precisa das suas capacidades de programação e optimizar o tempo durante as aulas práticas, desenvolvendo algoritmos cuja matéria não está bem compreendida.

Esta secção apresenta a arquitectura que este sistema utiliza (sub-secção 3.3.1) para dar suporte ao módulo responsável pela avaliação de algoritmos Java e as camadas utilizadas no sistema (sub-secção 3.3.2).

### 3.3.1 Arquitectura

A arquitectura de um sistema consiste na definição dos diversos componentes, das propriedades externas e dos relacionamentos com outras ferramentas/sistemas.

"*If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process.*" (Bass, et al., 2003)

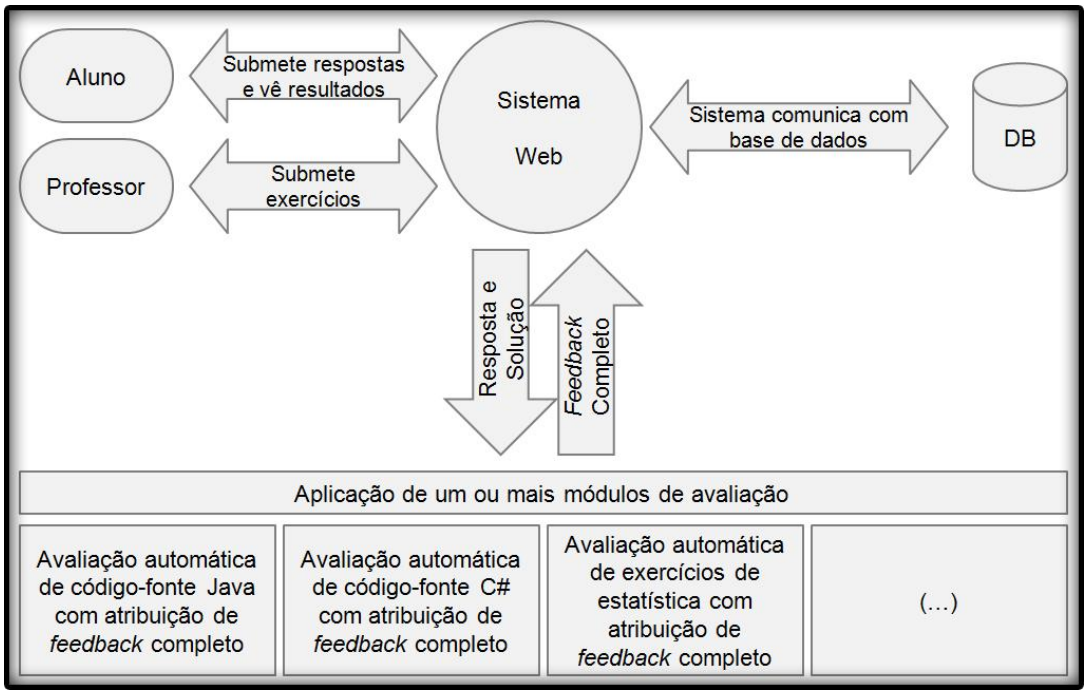
Uma arquitectura de sistema refere-se principalmente à estrutura, aos elementos internos, às relações existentes, a decisões e a componentes que todo o sistema suporta. No entanto, uma arquitectura do sistema também deve ter em atenção a evolução do próprio sistema. Geralmente, a evolução de um sistema é um fenómeno que se transmite pela sua mudança com o decorrer dos anos e das várias versões implementadas, desde o início da implementação até ao final do tempo de vida do sistema.

A mudança de um sistema não está relacionada única e exclusivamente em inserir e/ou remover funcionalidades, mas também, está relacionada com a manutenção necessária ao código-fonte do sistema enquanto estiver em funcionamento. Esta manutenção do sistema pode servir para melhorar ou danificar os atributos externos (vocacionados para os utilizadores, i.e. desempenho, tolerância a falhas, disponibilidade do sistema, etc.) e internos (vocacionados para os responsáveis do desenvolvimento, i.e. legibilidade do código-fonte, reutilização de código, etc.) de qualidade.

Com base no que está descrito acima, podemos verificar através da figura 11 a arquitectura que o sistema ACode utiliza. Esta arquitectura suporta vários módulos de avaliação permitindo uma

expansão das funcionalidades do sistema e a possibilidade de ser utilizado no processo de avaliação de outras unidades curriculares.

**Figura 11**: Arquitectura do Sistema ACode




Fonte: Marcos Pinto (2012)

Esta arquitectura, apesar de estar desenhada para suportar vários módulos que permitem a avaliação automática, é implementada com foco no módulo que permite a avaliação automática de código-fonte Java para satisfazer os objectivos propostos no início do projecto de mestrado.

A arquitectura desenhada para este sistema compreende quatro aspectos fundamentais: i) *stakeholders*; ii) sistema Web; iii) base de dados e iv) um módulo capaz de avaliar respostas fornecidas por alunos de forma automática atribuindo, no final, um *feedback* completo juntamente com a nota atribuída.

- ***Stakeholders*:** Os *stakeholders* referem-se a todos aqueles que estão interessados no sistema. Neste caso particular, podemos identificar dois *stakeholders*: alunos e professores. Genericamente, um professor tem a capacidade de inserir novos exercícios, a respectiva solução, um conjunto de testes que permitam avaliar o funcionamento da resposta do aluno e as percentagens respectivas para as várias formas de avaliação permitidas pelo módulo. Na perspectiva do aluno, é possível a sua inscrição nas disciplinas disponíveis no sistema, verificar quais os exercícios que estão disponíveis para que possa submeter uma resposta e, por último, avaliar de forma autónoma as suas capacidades através da submissão de respostas aos exercícios.

- **Sistema Web:** Um sistema Web é responsável por fornecer um conjunto de informações disponíveis 24 horas por dia através de um *browser* de internet. O sistema ACode está desenvolvido para suportar um ambiente Web permitindo a cada aluno submeter respostas aos exercícios a um ritmo desejado. Este sistema é responsável por fazer a ligação entre os utilizadores e os diversos componentes (base de dados e módulo de avaliação automática).

- **Base de Dados:** Como o próprio nome indica, as bases de dados permitem armazenar dados cuja informação pode ser consultada posteriormente pelos utilizadores do sistema. Neste caso concreto o sistema recorre a um sistema de base de dados para permitir aos utilizadores, não só, o acesso aos relatórios dos exercícios já avaliados, como também, o acesso a novos exercícios disponibilizados pelo docente.

- **Módulo para avaliação automática:** O módulo para avaliação automática permite que cada aluno, após submeter a resposta a um exercício disponível no sistema, fique automaticamente com a nota e um relatório completo. Este módulo vai permitir aliviar as tarefas do docente no processo de avaliação individual e fornece a capacidade ao aluno de fazer a auto-avaliação das suas capacidades em desenvolver um algoritmo.

### 3.3.2 Camadas do Sistema

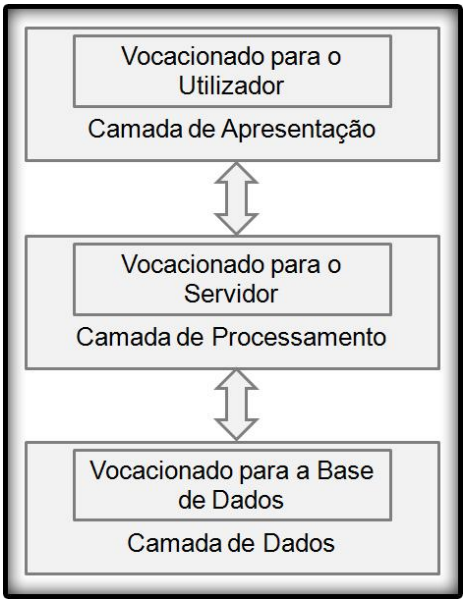
Como identificado anteriormente, o sistema ACode recorre a um sistema cliente-servidor para permitir uma disponibilidade de 24 horas por dia sem recurso a aplicações externas. Sendo este sistema, um sistema que permite a avaliação automática de código-fonte dos alunos através de algumas estratégias no processo de avaliação, nomeadamente a distribuição das percentagens entre os diferentes componentes englobados no processo, é fundamental separar o sistema em camadas.

Ao implementar o sistema, verificou-se que é necessário separar a zona que é apresentada ao aluno/docente (secção do sistema onde são apresentados todos os dados), da zona que faz a chamada ao módulo de avaliação automática e da zona responsável por efectuar consultas à base de dados. Com isto, é identificado um modelo de três camadas para que o sistema tenha alguma segurança na troca de dados entre os *stakeholders* e o armazenamento da informação.

Este sistema está implementado sob um modelo de três camadas pois deve funcionar de forma a que o sistema responda correctamente após ser substituída uma das camadas existentes. Da mesma forma, o sistema não deve sofrer alterações no seu funcionamento quando uma das camadas é actualizada, isto é, se existir uma alteração na camada que suporta a apresentação das informações ao utilizador do sistema, a camada que faz a ligação entre o módulo de avaliação automática e a base de dados não deve ser comprometida.

A figura 12 mostra um esquema da arquitectura utilizada no modelo de três camadas.

**Figura 12**: Modelo de três camadas




Fonte: Marcos Pinto (2012)

A partir da figura 12, verificamos que o mecanismo de funcionamento começa com uma requisição da parte do utilizador para o servidor que, por sua vez, comunica com a base de dados, modificando a informação desejada. Após isto, a base de dados envia a informação para o servidor e este disponibiliza-a para que o utilizador tenha acesso a essa informação.

- **Camada de Apresentação:** A camada de apresentação é conhecida pelo termo *Graphical User Interface* ou GUI. Esta camada está vocacionada para os utilizadores do sistema pois é através desta que são feitas as requisições em todo o sistema. Estas requisições podem ser consultas à base de dados (disponíveis sob um processo intermédio), *upload* das respostas do respectivo exercício para o servidor, *download* de ficheiros já submetidos, etc.

- **Camada de Processamento:** A camada de processamento é responsável pela implementação de todas as funcionalidades permitidas pelo sistema. É nesta camada que a ligação entre o utilizador e a base de dados é efectuada; permite o carregamento de ficheiros de forma segura, organizando-os por pastas, e protege os ficheiros submetidos contra consultas de utilizadores que não sejam os autores das respostas. Para cada acção executada no sistema ACode, é feito uma comunicação através dos processos contidos nesta camada.

- **Camada de Dados:** A terceira camada, camada de dados, representa toda a informação contida no sistema. É através desta camada que o utilizador consegue ter acesso aos

dados armazenados e assim poder consultar informações dos exercícios submetidos. Com isto, o aluno consegue obter um histórico da sua evolução caso submeta várias respostas para o mesmo exercício. No entanto, também é atribuído ao docente a capacidade de conseguir analisar todas as respostas submetidas no sistema, assim como o relatório da respectiva submissão.

## *3.4 Caracterização do Módulo de Avaliação*

Como já foi referido, para que o aluno consiga melhorar as suas capacidades e técnicas de programação é necessário obter um *feedback* da parte do docente sobre os exercícios que resolve. No entanto, muitas vezes, o *feedback* atribuído pelo docente demora bastante tempo uma vez que este tem que efectuar avaliações individuais para cada aluno.

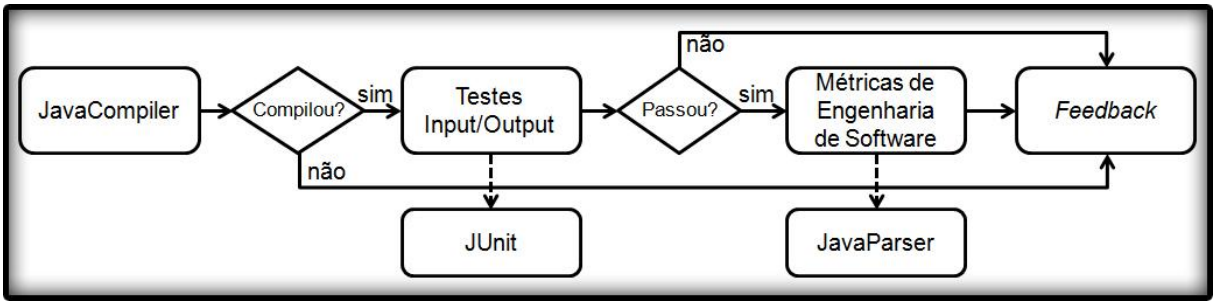
Com base neste problema, desenvolveu-se um módulo capaz de atribuir um *feedback* quase instantâneo ao aluno. O aluno apenas terá que esperar uns segundos até que o módulo produza o *feedback* e o resultado final da resposta para um determinado exercício. Desta forma e com recurso ao sistema Web (secção 3.3), os alunos conseguem efectuar a auto-avaliação, praticando exercícios sobre os quais não estão muito à vontade na respectiva teoria.

Esta secção apresenta a arquitectura utilizada no módulo de avaliação automática de algoritmos simples desenvolvidos em Java (sub-secção 3.4.1) e qual o processo que este módulo utiliza para atribuir a nota final e o respectivo *feedback* ao aluno (sub-secção 3.4.2).

### 3.4.1 Arquitectura

Como já foi referido anteriormente, o docente insere no sistema novos exercícios para que o aluno submeta as respectivas soluções. Após isto, é atribuído um *feedback* individual a cada exercício com base num módulo responsável pelo processo de avaliação. Na figura 13 verifica-se o fluxograma que o módulo responsável pela avaliação de código-fonte Java utiliza.

**Figura 13**: Fluxograma do módulo de avaliação automática




Fonte: Marcos Pinto (2012)

Através do fluxograma apresentado, verifica-se que o módulo responsável pela avaliação automática de código-fonte recorre a 4 fases para que o aluno tenha acesso ao *feedback*:

1. **JavaCompiler:** O código-fonte enviado pelo aluno é compilado pela primeira vez pelo sistema para verificar se não existe nenhum erro de compilação. Caso existam erros de compilação, o sistema atribui uma nota final de 0 valores (zero valores) e o relatório com os erros de compilação encontrados no algoritmo submetido. No entanto e após o aluno analisar o *feedback* atribuído pelo sistema, pode voltar a apresentar uma nova solução, caso não exceda o prazo de entrega das respostas para o exercício respectivo.

2. **Testes Input/Output:** A fase de testes de *input*/*output* apenas é executada caso não existam erros na fase da compilação do algoritmo submetido. Estes testes de *input*/*output* são baseados em testes JUnit (secção 3.4.2, sub-secção 3.4.2.1) fornecidos pelo docente. Nesta fase, caso não exista nenhum teste correcto, o sistema retorna ao aluno uma nota final de 0 valores (zero valores) juntamente com o respectivo relatório com a indicação de que não acertou nenhum teste. Caso exista pelo menos um teste correcto, o módulo de avaliação automática contínua no seu processo de avaliação.

3. **Métricas de Engenharia de Software:** Depois da fase de testes de *input*/*output* estar concluída, o módulo de avaliação automática passa automaticamente para a avaliação de algumas métricas de engenharia de software (secção 3.4.2, sub-secção 3.4.2.2). Nesta fase, o módulo recupera dados importantes a partir do código-fonte compilado e da solução previamente submetida pelo docente. Após ter recuperado esses dados de ambos os algoritmos, o módulo vai compara-los de forma a obter uma nota final para cada dado analisado, ou seja, o módulo obtém uma nota final para cada métrica de engenharia de software simples (secção 3.4.2, sub-secção 3.4.2.2).

4. ***Feedback*:** Depois das três fases anteriores estarem concluídas, o sistema reúne todas as informações e, através de um conjunto de fórmulas (secção 3.4.2, sub-secção 3.4.2.3), calcula a nota final da resposta submetida pelo aluno. O módulo de avaliação automática produz, juntamente com a nota final do aluno, um relatório detalhado com todas as métricas de engenharia de software analisadas e o resultado dos testes de *input*/*output*. Este *feedback* produzido pelo módulo é disponível ao aluno através do sistema e serve para que o aluno possa fazer uma auto-avaliação do seu desempenho.

### 3.4.2 Caracterização do Processo de Avaliação

Para que o módulo cumpra os objectivos para os quais foi desenvolvido, é necessário definir qual o processo de avaliação automática utilizado. Através da análise de alguns sistemas (capítulo II), verifica-se que existe a avaliação de testes de *input*/*output* ao algoritmo do aluno e, com isso, atribui-

se uma nota final. No entanto, este não é o melhor método para extrair uma nota do algoritmo produzido pelo aluno. A execução dos testes automáticos verifica apenas se o algoritmo submetido pelo aluno funciona, não avaliando o desempenho de cada resposta.

De acordo com os objectivos propostos para esta dissertação de mestrado, não é proposto o desenvolvimento de uma forma de avaliação em específico. No entanto, optou-se por englobar neste processo a avaliação de testes de *input*/*output* e algumas métricas de engenharia de software. Para este módulo não é analisada a hipótese da duplicação de código desenvolvido pelo aluno, mas, existem ferramentas para analisar essa duplicação (i.e. PMD - utilizado para verificar variáveis não declaradas, código duplicado, etc.).

Desta forma, e para que o processo de avaliação de código-fonte Java seja mais realista, este módulo para além de verificar um conjunto de testes definidos pelo docente com recurso à *framework* JUnit (sub-secção 3.4.2.1), também recorre a algumas métricas de engenharia de software (sub-secção 3.4.2.2), avaliando métrica a métrica. No final desta secção são apresentadas as fórmulas utilizadas para agrupar todas as informações do algoritmo submetido e produzir uma nota final (sub-secção 3.4.2.3).

#### 3.4.2.1 Testes JUnit

O JUnit é uma *framework open-source* e tem como principal objectivo proporcionar apoio à criação de testes automáticos na linguagem de programação Java. Esta *framework* facilita a criação automática de testes unitários ao código-fonte do aluno, apresentando os resultados consoante um valor esperado e um valor obtido para cada teste executado ao algoritmo.

Os testes JUnit possibilitam-nos analisar se cada método de uma classe funciona de acordo com o esperado, devolvendo possíveis erros ou folhas. Através destas informações devolvidas pelo JUnit o aluno pode melhorar o seu exercício para mitigar possíveis erros no método analisado.

O JUnit foi escolhido para incorporar este módulo de avaliação automática de código-fonte por apresentar algumas vantagens. A criação rápida de testes automáticos possibilita ao docente a capacidade de introduzir novos exercícios num curto espaço de tempo; depois de introduzidos os testes, são executados sem que seja interrompido o processo de execução do algoritmo submetido; fornecimento de uma resposta imediata e ser uma *framework* livre são algumas das vantagens que levaram à escolha deste mecanismo para avaliação dos testes de *input*/*output*.

#### 3.4.2.2 Métricas de Engenharia de Software

Uma métrica transmite-se na medição de um atributo, propriedade ou característica de uma determinada entidade, produto, processo ou recurso. É importante medir um algoritmo pois como

Lord William Thomson Kelvin diz se não conseguirmos medir, não conseguimos melhorar um sistema (Cardoso, 2006). Com base nisto, se aplicarmos algumas métricas de engenharia de software ao algoritmo submetido pelo aluno, este vai conseguir melhorá-lo e melhorar cada vez mais o seu desempenho no desenvolvimento de algoritmos simples.

Sendo o módulo implementado para dar suporte a alunos da unidade curricular de Introdução à Programação, é importante aplicarmos métricas para avaliar algoritmos simples. Posto isto, as métricas que são importantes e que fazem parte do módulo de avaliação automática de algoritmos são:

- **Complexidade ciclomática:** A complexidade ciclomática é uma métrica de engenharia de software já utilizada há algum tempo. Foi pensada e estudada por Thomas J. McCabe e mede e controla essencialmente a quantidade de caminhos de um algoritmo (McCabe, 1976). Esta métrica torna-se importante para este módulo de avaliação automática pois permite ao aluno verificar se o seu algoritmo tem mais ou menos processamento do que a solução ideal disponibilizada pelo docente.

- **Número de linhas:** O número de linhas é uma métrica de engenharia de software importante pois permite verificar o tamanho do algoritmo submetido pelo aluno através da contagem do número de linhas no código-fonte. Esta métrica também analisa a quantidade do esforço necessário para desenvolver o algoritmo prevendo qual o esforço necessário para manter o algoritmo. Com isto, se o número de linhas do código-fonte do aluno difere muito da solução submetida pelo docente, então é penalizado na nota final atribuída pelo módulo de avaliação automática.

- **Número de métodos:** O número de métodos presente num algoritmo é usado para analisar a quantidade de operações inseridas na classe. Esta informação torna-se relevante quando as classes são pouco mais do que tipo de dados, isto é, definem um novo objecto. O aluno tem a capacidade de recorrer a métodos auxiliares para resolver um determinado exercício. No entanto, isto implica ter mais capacidade de processamento e memória. Em comparação do algoritmo submetido e da solução fornecida pelo docente, cada classe apenas deve conter o número de métodos indicados em cada exercício. Caso isto não aconteça, o aluno será automaticamente penalizado na sua nota no processo de avaliação inserido no módulo.

- **Número de atributos da classe:** Os atributos das classes são variáveis cujos valores caracterizam o objecto dessa classe. Esses atributos representam o estado do objecto sendo que, cada objecto, guarda os seus respectivos valores. Esta métrica é importante ser analisada pois permite verificar se o aluno não excede o limite de atributos permitidos na resposta. É de referir que quanto maior for o número de atributos, maior será a

memória utilizada na execução do código-fonte. Em comparação com a solução fornecida pelo docente, podemos comparar o número de atributos e verificar se essa capacidade de memória permanece igual em ambos os algoritmos.

- **Número de classes:** O número de classes torna-se importante na avaliação de algoritmos Java pois uma classe representa um tipo de objectos. Para que o algoritmo esteja desenvolvido da melhor forma, não podem existir objectos desnecessários para o bom funcionamento do código-fonte. Caso o aluno implemente classes que não são pedidas no exercício, significa que vai ser penalizado na atribuição da nota final. A métrica calculada através do algoritmo vai ser também calculada na solução fornecida pelo docente de forma a existir uma comparação e atribuir assim a nota final para esta métrica de engenharia de software.

Apesar de existirem muitas métricas que possam ser utilizadas neste módulo, as cinco métricas apresentadas são as essenciais para avaliação de algoritmos simples. Caso o sistema seja utilizado para avaliação de algoritmos um pouco mais complexos (i.e. Programação Orientada a Objectos e Programação Concorrente e Distribuída), é importante adicionar mais métricas que permitam efectuar a avaliação do aluno da melhor forma possível e atribuir ao docente um resultado mais realista das capacidades de cada aluno.

#### 3.4.2.3 Expressões Utilizadas

Como já foi referido, o sistema ACode realiza a avaliação de código-fonte apresentado pelo aluno, em primeiro lugar, através do resultado da compilação, isto é, compila o algoritmo submetido para verificar se existem erros. Seguidamente, verifica se a solução apresentada pelo aluno passa por um conjunto de testes JUnit que são fornecidos pelo docente previamente à submissão de todas as respostas. A expressão (1) descreve como esta parte da avaliação é realizada no interior do sistema ACode, relacionando o número total de testes com o número de testes que o algoritmo do aluno executou correctamente. Este relacionamento (nota da fase de testes para o algoritmo do aluno), fica armazenado numa variável denominada *result_tests*.

$$result\_tests = \frac{testes\_correctos \times 20}{total\_testes} \qquad (1)$$

Posteriormente, o sistema extrai dados específicos baseados em métricas de engenharia de software da resposta submetida pelo aluno e da solução fornecida pelo docente juntamente com o enunciado do problema. Como referido na sub-secção anterior, o número de linhas de cada método, o número de métodos, a complexidade ciclomática e o número de variáveis globais de cada classe são alguns dos exemplos das métricas de engenharia de software que o módulo utiliza na avaliação automática de código-fonte Java. O sistema irá então comparar os dois conjuntos de métricas de engenharia de software (do aluno e do docente), como representado na expressão (2), ficando assim

calculado a nota para as métricas de engenharia de software definidas. Este resultado fica armazenado numa variável denominada *metricas*.

$$metricas = 20 - e^{\left(\frac{metrica\_algoritmo\_aluno - metrica\_solucao\_docente}{metrica\_solucao\_docente} - 1\right)} \quad (2)$$

A expressão (2) representa uma forma genérica para calcular cada métrica de engenharia de software de forma individual, ou seja, cada métrica do algoritmo produzido pelo estudante é calculada por comparação com a métrica da solução inserida no sistema pelo docente. Torna-se importante referir que nesta fase de avaliação das métricas de engenharia de software, caso existam vários valores para uma métrica de engenharia de software específica (i.e. cálculo da complexidade ciclomática dos vários métodos no cálculo da mesma métrica para uma classe), o resultado final da nota dessa métrica de engenharia de software é calculada pela média da avaliação de cada métrica analisada, como representa a expressão (3), e fica armazenada na variável *metrica_igual*.

$$metrica\_igual = \frac{\sum_{k=1}^{n} result\_metrics_k}{numero\_total\_metricas\_iguais} \quad (3)$$

Após o sistema analisar todas as notas das métricas de engenharia de software, calcula a nota global de todas as métricas combinadas, utilizando os dados baseados no cálculo das métricas da expressão (3) e um conjunto de percentagens fornecidas pelo docente, determinando a importância de cada elemento da avaliação.

Tendo em mente a relevância dos elementos específicos na avaliação do algoritmo submetido pelo aluno, o docente fornece uma percentagem para cada uma das métricas. A tabela 03 descreve todas as percentagens que são utilizadas no processo de avaliação baseado nas métricas de engenharia de software.

**Tabela 03:** Descrição das percentagens das variáveis

| Variável | Descrição |
|---|---|
| $P_{classe}$ | Probabilidade atribuída às métricas da classe |
| $P_{metodo}$ | Probabilidade atribuída às métricas do método |
| $P_{projecto}$ | Probabilidade atribuída às métricas do projecto |
| $pcc_{classe}$ | Probabilidade atribuída à complexidade ciclomática da classe |
| $pln_{classe}$ | Probabilidade atribuída ao número de linhas da classe |
| $pmn_{classe}$ | Probabilidade atribuída ao número de métodos da classe |
| $pan_{classe}$ | Probabilidade atribuída ao número de atributos da classe |
| $pcc_{metodo}$ | Probabilidade atribuída à complexidade ciclomática do método |
| $pln_{metodo}$ | Probabilidade atribuída ao número de linhas do método |
| $pcn_{projecto}$ | Probabilidade atribuída ao número de classes do projecto |
| $pmn_{projecto}$ | Probailidade atribuída ao número de métodos do projecto |
| $pcc_{projecto}$ | Probabilidade atribuída à complexidade ciclomática do projecto |
| $pln_{projecto}$ | Probabilidade atribuída ao número de linhas do projecto |
| $pan_{projecto}$ | Probabilidade atribuída ao número de atributos do projecto |

As expressões (4), (5) e (6) representam o cálculo efectuado pelo módulo de avaliação automática para atribuição das notas às métricas da classe, do método e do projecto respectivamente.

$$metricas_{classe} = pcc_{classe} \times complexidade\_ciclomatica + pln_{classe} \times numero\_linhas + pmn_{classe} \times numero\_metodos + pan_{classe} \times numero\_atributos \quad (4)$$

$$metricas_{metodo} = pcc_{metodo} \times complexidade\_ciclomatica + pln_{metodo} \times numero\_linhas \quad (5)$$

$$metricas_{projecto} = pcn_{projecto} \times numero\_classes + pmn_{projecto} \times numero\_metodos + pcc_{projecto} \times complexidade\_ciclomatica + pln_{projecto} \times numero\_linhas + pan_{projecto} \times numero\_atributos \quad (6)$$

A expressão (7) apresenta a fórmula utilizada para calcular a nota final do conjunto de todas as métricas utilizadas na fase da avaliação das métricas de engenharia de software, usando as percentagens descritas na tabela 03. O valor da nota final das métricas de engenharia de software está armazenado na variável *result_metrics*.

$$result\_metrics = P_{classe} \times metricas_{classe} + P_{metodo} \times metricas_{metodo} + P_{projecto} \times metricas_{projecto} \quad (7)$$

Finalmente, todos os dados do processo de avaliação são combinados, a fim de determinar a nota final atribuída pelo módulo de avaliação automática de código-fonte Java à resposta do aluno. A expressão (8) representa a fórmula utilizada para calcular a nota final.

$$R = \begin{cases} 0 & , \quad compilação = 0 \\ 0 & , \quad compilação = 1 \; e \; result\_tests = 0 \\ P_{r1} \times result\_tests + P_{r2} \times result\_metrics & , \quad compilação = 1 \; e \; result\_tests > 0 \end{cases} \quad (8)$$

Na expressão (8), $P_{r1}$ representa a percentagem associada à importância da fase de testes e $P_{r2}$ representa a percentagem associada à importância da fase da avaliação das métricas de engenharia de software. O docente fornece ambas as percentagens de acordo com o grau de dificuldade de cada exercício submetido no sistema a fim do módulo efectuar a melhor avaliação para o respectivo exercício. No final, o aluno recebe um relatório completo com a sua nota final, a quantidade de testes executados correctamente e todas as informações relativas às métricas de engenharia de software (i.e. nota final de cada métrica, comparação da métrica da resposta do aluno com a solução do docente, etc.).

# Capítulo IV - Implementação do Sistema

## *4.1 Introdução*

Depois de formulada a caracterização do sistema ACode (capítulo III) e a fim de criar um sistema que está em conformidade com os requisitos identificados (capítulo III, secção 3.2), torna-se importante desenvolver o sistema que faz a avaliação automática de código-fonte do aluno e o sistema Web que o põe em funcionamento. Sendo a linguagem de programação Java a mais utilizada, não só, no ISCTE - Instituto Universitário de Lisboa, mas também, noutras instituições de ensino, optou-se por desenvolver o módulo em Java, permitindo, mais tarde, que futuros alunos possam expandi-lo para facultar a avaliação de código-fonte um pouco mais complexo.

Para implementar o sistema Web recorre-se à linguagem de programação PHP. A linguagem de programação PHP tem como âmbito desenvolver aplicações vocacionadas para a Web e a sua característica principal é o desenvolvimento de sistemas mais rápidos, não esquecendo a simplicidade e a eficiência que lhe está subjacente.

Para que seja possível desenvolver uma ferramenta que permita avaliar o código-fonte do aluno de forma automática, aliviando algum trabalho associado à avaliação de exercícios individuais da parte do docente, é necessário separá-la em duas grandes partes: o sistema Web e o módulo que permite a avaliação automática de algoritmos. Com isto, o docente tem a possibilidade de introduzir mais exercícios para que os alunos pratiquem os seus conceitos e técnicas associadas à linguagem de programação Java.

Este capítulo apresenta em detalhe todos os mecanismos que fazem parte, não só, do sistema Web, mas também, do módulo de avaliação automática de algoritmos, quais as funções utilizadas e como está implementada a interação entre as duas ferramentas necessárias (sistema Web e módulo de avaliação automática) para ajudar o docente no processo de avaliação da unidade curricular de Introdução à Programação e permitir que o aluno efectue uma auto-avaliação dos seus conceitos.

Neste capítulo está especificada a arquitectura do modelo utilizado no sistema ACode (secção 4.2), sobre a qual foi detalhada e devidamente identificada no sistema ACode a camada de apresentação (sub-secção 4.2.1), a camada de processamento (sub-secção 4.2.2) e a camada de dados (sub-secção 4.2.3). É também apresentado neste capítulo uma especificação detalhada do módulo que permite a avaliação automática dos algoritmos submetidos através do sistema ACode (secção 4.3). Na secção 4.3 está apresentado todo o funcionamento necessário para que o módulo avalie o algoritmo, desde a compilação (sub-secção 4.3.1), passando pelos testes JUnit (sub-secção 4.3.2), pelas métricas de engenharia de software utilizadas (sub-secção 4.3.3) e finalizando na atribuição do *feedback* (sub-secção 4.3.4). Por último, na secção 4.4 está identificado o conteúdo necessário para que o servidor suporte este sistema e para que funcione da melhor forma.

## *4.2 Especificação da Arquitectura em Camadas*

Como foi identificado no capítulo III (secção 3.3, sub-secção 3.3.2) é usado o modelo de três camadas no desenvolvimento do sistema Web. Afim de conhecer a estrutura implementada do sistema utilizado nesta dissertação de mestrado, esta secção apresenta detalhadamente o que cada uma das camadas abrange.

Esta secção apresenta o modo de funcionamento de cada camada utilizada no sistema, nomeadamente a camada de apresentação (sub-secção 4.2.1), a camada de processamento (sub-secção 4.2.2) e a camada de dados (sub-secção 4.2.3).

### 4.2.1 Camada de Apresentação

**Figura 14:** Formulário de Login


Fonte: Marcos Pinto (2012)

O sistema que suporta o módulo de avaliação automática de código-fonte apresenta duas formas diferentes de visualização, dependendo do utilizador que efectua a entrada no sistema. No entanto, para todos os utilizadores a entrada no sistema é igual (ver figura 14). Quando o aluno entra no sistema é obrigado a inscrever-se nas unidades curriculares disponíveis (Anexo A, página 79) para que lhe seja atribuída a capacidade de resolver os exercícios armazenados no sistema. No entanto, esta inscrição por parte do aluno, só fica disponível após uma confirmação do docente (Anexo B, página 80). Esta acção permite um controlo sobre o registo dos alunos inscritos do presente ano curricular.

Quando um aluno entre no sistema ACode, para além de se poder registar nas disciplinas, tem também a possibilidade de submeter respostas a exercícios fornecidos pelo docente através de um ficheiro com extensão *.zip*[19] ou simplesmente através de texto (Anexo C, página 81). Depois de submetida uma resposta ao exercício, o sistema, também fornece a possibilidade ao aluno de submeter mais versões ao exercício respectivo. Com isto, o aluno pode submeter várias respostas, analisando em cada uma delas o resultado obtido (o sistema ACode tem a capacidade de armazenar cada resposta submetida) e efectuar a auto-avaliação das suas capacidades.

No cenário do docente, para além das características atribuídas ao aluno, tem também um conjunto de novas funcionalidades. A funcionalidade mais importante para o sistema ACode e atribuída ao docente é a gestão de exercícios, nomeadamente a inserção de novos exercícios no sistema (Anexo D, página 82) para que os alunos possam submeter as respectivas respostas. Para cada exercício submetido no sistema ACode, o docente tem que identificar todas as percentagens necessárias para o bom funcionamento do módulo. Com isto, o módulo pode efectuar a avaliação

[19] Formato de compressão de ficheiros muito utilizado na internet.

automática de código-fonte e atribuir ao aluno uma nota final de acordo com a especificação do processo de avaliação.

É também atribuído ao docente a capacidade de adicionar novas disciplinas (inserindo a quantidade de percentagens que o respectivo módulo utiliza) e bloquear um aluno, isto é, caso exista um novo ano curricular, o docente pode bloquear a entrada a um ou vários alunos. Enquanto o aluno apenas analisa as suas respostas submetidas, é permitido ao docente analisar as respostas de todos os alunos inscritos. Esta característica fornece ao docente a capacidade de explorar as respostas dos alunos e analisar qual ou quais os conceitos em que os alunos apresentam mais dificuldades.

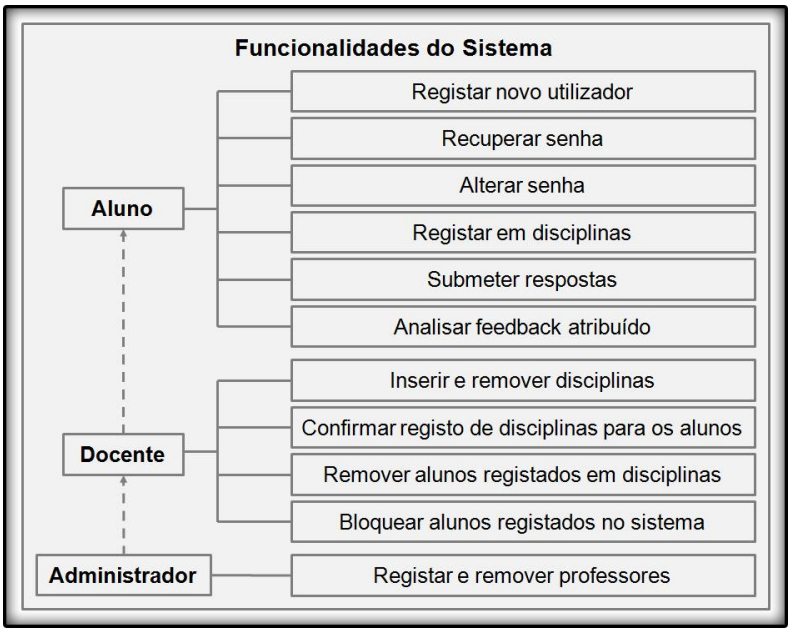
Para desenvolver esta camada foram utilizadas as linguagens de programação PHP e JavaScript, o HTML (*HyperText Markup Language*) e a linguagem de estilo CSS (*Cascading Style Sheets*) que permitiu obter uma apresentação final do sistema semelhante ao sistema Fénix já implementado na instituição.

### 4.2.2 Camada de Processamento

Como referido no capítulo III (secção 3.3, sub-secção 3.3.2), a camada de processamento é responsável por manipular todos os dados trocados entre o utilizador, o sistema de base de dados e o módulo que permite a avaliação automática de algoritmos simples. Cada acção desempenhada no sistema é feita através desta camada, permitindo uma maior segurança na troca de informações.

As funcionalidades desempenhadas através do sistema ACode estão descritas na figura 15.

**Figura 15:** Funcionalidades do Sistema ACode




Fonte: Marcos Pinto (2012)

As funcionalidades mais importantes no sistema, e que estão de acordo com os objectivos deste trabalho são:

**Inserir exercícios:** A funcionalidade da inserção de exercícios no sistema está vocacionada para o docente. Sempre que é inserido um novo exercício, o docente tem que identificar alguns atributos que são necessários para o aluno e para o módulo de avaliação automática de código-fonte. Da informação inserida pelo docente no sistema, fica disponível ao aluno o título do exercício, o enunciado, a data final de entrega e a disciplina a que está associado o exercício. A restante informação (percentagens, ficheiro de testes, exercício resolvido e data de disponibilidade) apenas fica armazenada no sistema para utilização posterior no módulo de avaliação automática, não sendo visível pelo aluno. As percentagens inseridas têm que estar separadas com o símbolo '#' pois o sistema utiliza-o para separar cada uma das percentagens e enviá-las para o módulo responsável pela avaliação, de acordo com a função *arranjaPercentagens*, definida na figura 16.

**Figura 16:** Função definida no sistema para interpretar cada percentagem

```
private function arranjaPercentagens($percentagens){
	$temp = explode("#", $percentagens);
	$temp2 = "";
	for($i = 0; $i < sizeof($temp); $i++){
		$temp2 .= "\"" . $temp[$i] . "\" ";
	}
	return $temp2;
}
```

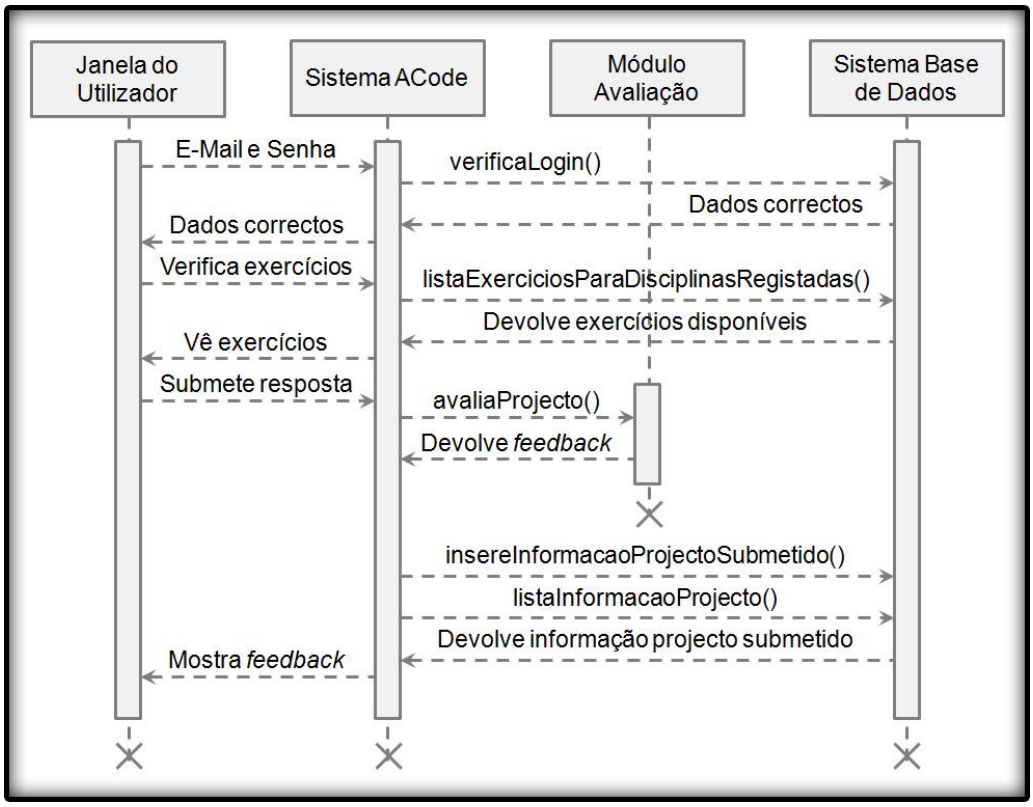
Fonte: Marcos Pinto (2012)

**Submeter respostas:** Depois dos exercícios inseridos, e caso o aluno esteja registado na respectiva disciplina, pode submeter respostas. O sistema atribui ao aluno a possibilidade de submeter dois tipos de resposta: um ficheiro com extensão *.zip* ou implementar a classe Java directamente no sistema. As respostas são armazenadas no sistema em duas pastas distintas, nomeadamente na pasta *projectos_avaliados* e *projectos_submetidos*, pois assim o aluno não tem acesso aos testes efectuados ao seu algoritmo, podendo efectuar o *download* da versão submetida. Após isto, e depois de todas as percentagens interpretadas pelo sistema, recorre-se à função *exec* do PHP (a função *exec* utiliza a linha de comandos para executar um comando específico passado no argumento), no caso particular do módulo de avaliação automática de código-fonte Java, a forma de utilização da função *exec* é: `exec(java -jar $modulo_avaliacao "$projecto_docente" "$caminho_projecto_aluno" $percentagens, $output)`, em que a variável `$modulo_avaliacao` representa o caminho relativo da localização do módulo, a variável `$projecto_docente` representa o caminho relativo da localização da solução submetida pelo docente quando inseriu o exercício, a variável

`$caminho_projecto_aluno` representa o caminho relativo da localização da resposta fornecida pelo aluno, a variável `$percentagens` representa as percentagens fornecidas pelo docente e arranjadas de acordo com a função representada na figura 16 e, por último, a variável `$output` é onde fica armazenado o resultado da execução do comando apresentado anteriormente.

**Analisar *feedback* atribuído:** Quando um aluno responde aos exercícios propostos pelo docente através do sistema ACode, é produzido um *feedback* automático. Este *feedback* tem origem no módulo de avaliação automática de algoritmos simples (secção 4.3, sub-secção 4.3.4) e engloba a nota final e um relatório completo analisando cada métrica de engenharia de software avaliada (Anexo E, página 83). Através dos dados produzidos pelo módulo e após serem analisados pelo aluno, este tem a capacidade de fazer a auto-avaliação das suas capacidades. O aluno, pode também optar por submeter várias versões da sua resposta até que o *feedback* atribuído seja o esperado. A figura 17 representa a interação entre o aluno, a comunicação do sistema Web e o módulo responsável pela avaliação automática do algoritmo submetido.

**Figura 17:** Diagrama de sequência - funcionamento geral do sistema




Fonte: Marcos Pinto (2012)

## 4.2.3 Camada de Dados

Como referido na secção 3.3, sub-secção 3.3.2, do capítulo III, a camada de dados representa os dados necessários para o funcionamento do sistema Web e o módulo de avaliação automática. O

sistema utiliza uma classe para efectuar a gestão dos dados na base de dados, nomeadamente a classe *DataBaseMysql.php*. As tabelas *utilizador*, *exercícios* e *disciplina* presentes na base de dados são responsáveis pelo funcionamento do módulo pois contêm informações necessárias para que o projecto submetido do aluno seja avaliado correctamente. Já as tabelas *projecto_user* e *disciplina_user* apresentam informações direcionadas para o aluno, permitindo que este tenha acesso a um conjunto de informações que possibilitam efectuar a sua auto-avaliação.

Para melhor identificar o que cada tabela representa no sistema, apresenta-se em seguida a descrição de cada uma das tabelas apresentadas anteriormente.

- **Tabela *utilizador*:** A tabela *utilizador* é responsável por armazenar todos os utilizadores registados. Esta tabela indica se um determinado utilizador pode ou não entrar no sistema, consoante o seu estado actual (um utilizador pode entrar no sistema se o seu estado é 1 e não pode entrar se o seu estado é 0). Apresenta também o e-mail do utilizador, permitindo que o sistema envie informações relevantes para o respectivo aluno.

- **Tabela *exercicios*:** Esta tabela representa todos os exercícios disponíveis no sistema. O título, o enunciado, o caminho do ficheiro de testes, o caminho da resposta do docente e as percentagens são alguns dos atributos necessários para que o aluno possa submeter as suas respostas e praticar os conceitos e técnicas que estão associados ao respectivo exercício. O atributo percentagens é fundamental estar bem definido pelo docente pois é através deste que o módulo vai produzir a nota final à resposta submetida.

- **Tabela *disciplina*:** A tabela *disciplina* é responsável por armazenar a localização de todos os módulos utilizados no sistema. Apresenta um atributo percentagens onde é armazenado um valor correspondente ao número total de percentagens utilizadas no módulo. Este valor é depois utilizado pelo sistema sempre que o docente insere um novo exercício para a respectiva disciplina, atribuindo ao sistema a característica de verificar se o docente insere o número de percentagens correctas.

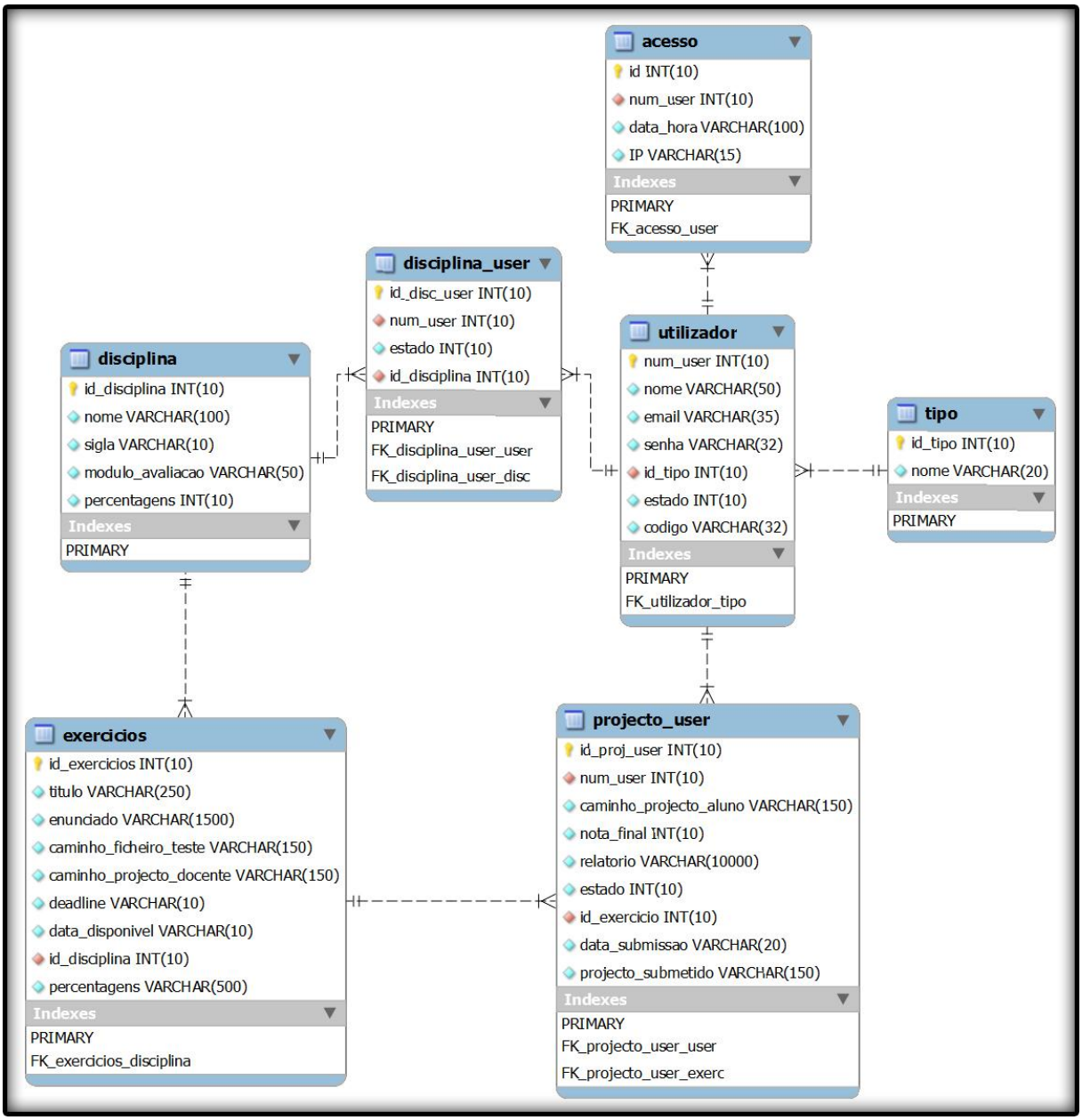
- **Tabela *projecto_user*:** Esta tabela, apesar de não ser útil para o funcionamento do módulo de avaliação automática, é fundamental para o aluno pois permite armazenar todos as respostas submetidas assim como o respectivo relatório, a nota final e o caminho relativo da resposta fornecida pelo aluno. Com isto, o aluno tem um registo completo de todas as respostas fornecidas, podendo analisar o *feedback* produzido pelo módulo a qualquer altura.

- **Tabela *disciplina_user*:** A tabela *disciplina_user* apenas identifica se um determinado aluno está ou não registado numa determinada disciplina. Assim, os alunos que não

tenham acesso a uma disciplina, não podem verificar os exercícios da disciplina respectiva nem submeter respostas para estes exercícios.

O diagrama de classes utilizado pelo sistema ACode está representado na figura 18. Este diagrama utiliza um conjunto de tabelas de forma a armazenar um conjunto de dados que permitam ao aluno analisar o *feedback* da avaliação automática sempre que pretender.

**Figura 18:** Diagrama de classes utilizado pelo sistema




Fonte: Marcos Pinto (2012)

## *4.3 Especificação do Módulo de Avaliação*

Como foi referido anteriormente, este projecto tem como objectivo principal a implementação de um módulo que efectue a avaliação automática de código-fonte submetido por alunos, sobre o qual produz um *feedback* automático para que o aluno tenha capacidade de fazer a auto-avaliação.

Este módulo está desenvolvido na linguagem de programação Java. A linguagem de programação Java para além de ser uma linguagem orientada a objectos, é relativamente simples, contudo contém um conjunto de pacotes de bibliotecas necessários para desenvolver programas úteis (Horstmann, 2002).

Para que o módulo permita ajudar os alunos a desenvolver os seus conceitos de programação e para ajudar no processo de avaliação da unidade curricular de Introdução à Programação, é fundamental que esteja definido uma forma justa para avaliar o código-fonte produzido. Como já foi referido no capítulo III (secção 3.4, sub-secção 3.4.1), o módulo que permite obter o *feedback* ao aluno (após uma avaliação automática do código-fonte) recorre a 4 fases: i) o código-fonte é compilado através da classe *Main* (sub-secção 4.3.1) e verifica se existe algum erro; ii) o código-fonte compilado é testado com um conjunto de testes JUnit fornecidos pelo docente (sub-secção 4.3.2); iii) é usado um conjunto de métricas de engenharia de software para comparar a solução do docente com a resposta do aluno, recorrendo à *framework JavaParser* (sub-secção 4.3.3); iv) e por último, com base nas fases anteriores, fornece um *feedback* (sub-secção 4.3.4) para que o aluno efectue a auto-avaliação e identifique qual ou quais as áreas que é necessário mais estudo.

Sempre que um exercício é fornecido ao módulo de avaliação, é também fornecido qual o caminho relativo do módulo da disciplina respectiva, qual o caminho relativo da solução do docente e quais as percentagens associadas ao exercício cuja resposta foi submetida. Estes dados são posteriormente passados para a função *iniciaExtracao* (presente na classe *ExtrairMetricas*) onde todo o processo de avaliação e atribuição de *feedback* é iniciado.

O método *iniciaExtracao* permite iniciar a análise de cada uma das métricas de engenharia de software definidas atribuindo, no final, uma análise individual de cada uma das métricas. Esta análise é feita de modo a que o aluno fique com a informação do valor da métrica da sua resposta com o valor da respectiva métrica da solução do docente. Esta informação irá permitir ao aluno alterar o código-fonte submetido em zonas específicas fornecendo a possibilidade de melhorar o seu desempenho.

Esta secção apresenta a forma como o módulo de avaliação automática compila o código-fonte submetido pelo aluno (sub-secção 4.3.1), como são formados os testes JUnit (sub-secção 4.3.2) pelo docente, como são avaliadas as métricas de engenharia de software (sub-secção 4.3.3) para que o

aluno tenha um resultado mais realista e, por último, como é atribuído o *feedback* ao aluno (sub-secção 4.3.4).

### 4.3.1 Compilação do Código-Fonte Submetido

Para que o módulo de avaliação automática atribua uma nota superior a 0 valores (zero valores), o código-fonte do aluno tem que compilar correctamente. É necessário que o código-fonte submetido pelo aluno não apresente erros nesta fase para que possa passar para a fase seguinte, fase em que é executada uma bateria de testes definidos pelo docente.

O módulo recorre à classe *Main* da biblioteca *com.sun.tools.javac* para analisar, não só, erros de gramática e sintaxe da linguagem de programação Java, como também, identificar se os métodos ou os objectos necessários foram criados pelo aluno.

A ferramenta *javac* lê cada classe Java e compila-a numa classe *bytecode* (a classe *bytecode* é interpretada pela máquina virtual que fará a execução do algoritmo). Para que o compilador da classe *Main* funcione correctamente, é necessário que o aluno submeta um ficheiro com a extensão .java. Na figura 19 está representada parte da função responsável pela compilação do código-fonte submetido pelo aluno.

**Figura 19:** Função para analisar a compilação do algoritmo

```
private String resultadoCompilacaoCodigoFonte(...){
//...
	Main javac = new Main();
	String[] opcoes = new String[] {"-classpath",
	System.getProperty("java.class.path") + ";" +
	caminho_projectos + "\\junit.jar"};
	String[] compilacao = new String[ficheiros.length +
	opcoes.length];
	compilacao[0] = opcoes[0];
	compilacao[1] = opcoes[1];
	for(int i = 0; i != ficheiros.length; i++){
		compilacao[i+2] = ficheiros[i];
	}
	int result = 0;
	try{
		PrintWriter erro = new PrintWriter(new
		File(projecto_aluno + "/erro_compilacao.txt"));
		result = javac.compile(compilacao, erro);
		erro.close();
	}catch (FileNotFoundException e){}
//...
}
```

Fonte: Marcos Pinto (2012)

A função *resultadoCompilacaoCodigoFonte* recebe como argumento uma lista com o(s) ficheiro(s) do aluno para ser(em) utilizado(s) pela função *compile* da classe *Main*. Sendo necessário a execução de vários testes ao algoritmo fornecido pelo aluno, torna-se necessário adicionar ao *classpath* (variável de ambiente que serve para compilar e executar algoritmos) a *framework* JUnit.

No final da compilação, caso exista algum erro, a função *resultadoCompilacaoCodigoFonte* retorna o resultado, ficando este armazenado numa variável denominada *resultado_compilacao* com algumas modificações na sua apresentação final. A figura 20 apresenta um exemplo de uma compilação, que não foi bem sucedida, para a classe *Factorial*.

**Figura 20:** Exemplo da compilação para a classe *Factorial*

**Classe Factorial**

```
public class Factorial {
	public int factorial(final int valor) {
		if (valores == 0)
			return 1;
		else
			return valor * factorial(valor - 1);
	}
}
```

**Resultado da Compilação da classe Factorial**

```
Classe: Classe: Factorial.java
Linha: 3 - O símbolo não foi encontrado.
Símbolo: variable valores
Localização: class Factorial
1 Erro
```

Fonte: Marcos Pinto (2012)

Através do resultado atribuído à compilação e fornecendo o sistema ACode a possibilidade de submeter várias respostas para o mesmo exercício, o aluno consegue melhorar o seu algoritmo até que não exista erros de compilação.

### 4.3.2 Testes JUnit

Depois do código-fonte compilado correctamente através do módulo de avaliação automática, a resposta do aluno é submetida a uma série de testes definidos pelo docente quando submetido o enunciado do exercício. Estes testes são formados através de código que recorre à *framework* JUnit, isto é, o docente tem que conhecer alguns dos métodos necessários para que possa definir os testes para cada exercício.

A *framework* JUnit permite executar uma série de testes automaticamente ao código-fonte desenvolvido em Java (Beck, et al.). Através da classe *TestCase* (que pode ser estendida para receber ambientes de teste específicos) desta *framework*, o módulo verifica se cada um dos métodos de uma classe funciona de acordo com o esperado. Para cada teste são fornecidos os parâmetros de entrada necessários para fazer uma chamada ao método que é testado pelo módulo. Seguidamente, o *framework* analisa o valor obtido através da chamada ao método do aluno e compara-o com um valor esperado definido pelo docente. O módulo de avaliação automática usa esta técnica para testar cada método do aluno com a solução fornecida pelo docente e produzir um relatório indicando o número de testes que foram bem sucedidos e quantos testes foram executados no algoritmo.

Estando a unidade curricular de Introdução à Programação mais orientada para o desenvolvimento de algoritmos simples, torna-se importante conhecer os métodos da *framework* JUnit fundamentais para definir os testes que permitem efectuar parte da avaliação do algoritmo do aluno. Na tabela 04 estão apresentados alguns dos métodos de teste disponíveis que são usados no módulo para avaliar o desenvolvimento de código-fonte simples.

**Tabela 04:** Métodos JUnit mais utilizados

| Métodos JUnit | Descrição |
|---|---|
| assertTrue | Método que verifica se o valor passado como parâmetro é verdadeiro. O método *Arrays.equals* pode ser utilizado para verificar uma igualdade entre dois vectores. |
| assertFalse | Método que verifica se o valor passado como parâmetro é falso. |
| assertEquals | Método que verifica se dois objectos são iguais. Recebe como parâmetro o valor esperado pelo docente e o valor obtido pelo método a ser testado. |

O método *assertEquals*, definido na tabela acima, pode ser usado de muitas formas diferentes. No entanto, a forma mais frequente de utilizar este método é através da comparação entre um valor esperado (definido pelo docente) e um valor que é atribuído pelo método implementado pelo aluno. Caso estes dois valores não sejam iguais, será lançada uma excepção do tipo *AssertionFailedError* que permite ao módulo passar para o teste seguinte e não incrementar a variável responsável pelo armazenamento da quantidade de testes correctos.

Para que o código-fonte, submetido pelo aluno, seja testado pelo módulo recorrendo a testes de *input/output*, o docente deve fornecer um ficheiro com os testes que devem ser executados ao algoritmo do aluno. A figura 21 mostra um exemplo de um ficheiro de teste utilizado para testar o método *factorial* da classe *Factorial*.

**Figura 21:** Exemplo de um ficheiro de teste

```
Factorial f = new Factorial ();
assertEquals(1, f.factorial(0));
##
Factorial f = new Factorial ();
assertEquals(1, f.factorial(1));
##
Factorial f = new Factorial ();
assertEquals(2, f.factorial(2));
##
Factorial f = new Factorial();
assertEquals(120, f.factorial(5));
##
```

Fonte: Marcos Pinto (2012)

Neste ficheiro de testes, o docente tem que escrever os testes para todos os métodos que devem ser testados na resposta submetida pelo aluno. A marca "##" presente no ficheiro, é apenas uma forma de separar os testes no mesmo ficheiro, permitindo ao módulo de avaliação automática contar, de forma independente, os testes que foram bem sucedidos.

Existem várias formas de executar testes de uma forma automática. A maneira descrita acima, e a mais comum, é através da utilização da *framework* JUnit. No entanto, também poderia ser utilizado o Java Reflection para executar os testes ao código-fonte submetido pelo aluno. De acordo com testes efectuados, esta segunda solução requer mais processamento e, por esta razão, torna-se num processo mais moroso. Como este módulo deve permitir a atribuição de *feedback* quase instantâneo ao aluno, torna-se mais rápido invocar métodos directamente recorrendo à *framework* JUnit, deixando a opção do Java Reflection de parte.

### 4.3.3 Métricas de Engenharia de Software

Após o módulo de avaliação automática executar todos os testes e caso exista pelo menos um teste correcto, passa automaticamente para a análise das métricas de engenharia de software. Para avaliar o código-fonte produzido pelo aluno, foram identificadas cinco métricas de engenharia de software fundamentais: complexidade ciclomática, número de linhas, número de métodos, número de atributos da classe e o número de classes.

A análise das cinco métricas de engenharia de software identificadas anteriormente é feita recorrendo à *framework JavaParser* (Project Google) que efectua um processamento do código-fonte submetido pelo aluno para produzir uma estrutura de dados correspondente. Esta estrutura de dados, permite ao módulo de avaliação automática recuperar alguns atributos úteis de cada classe e cada método presente na solução do aluno (i.e. nome, conteúdo, parâmetros, etc.).

Através da utilização da classe *VoidVisitorAdapter* (pode ser estendida para obter informações mais específicas de um algoritmo) o módulo de avaliação automática obtém os dados necessários para produzir a informação das métricas que serão processadas de modo a atribuir uma nota à solução do aluno através das expressões referidas no capítulo III (secção 3.4. sub-secção 3.4.2.3).

Foram implementadas quatro classes que permitem extrair a informação do código-fonte através da *framework JavaParser*: *MetricasMetodos*; *MetricasClasse*; *MetricasPackage* e *MetricasProjecto*. Esta forma de implementação possibilita uma maior organização do código, dividindo pelas classes respectivas a informação de cada métrica de engenharia de software a ser analisada. Em seguida estão apresentadas as classes utilizadas para analisar as métricas de engenharia de software do código-fonte submetido pelo aluno.

**Análise das métricas através da classe *MetricasMetodos*:**

As métricas de engenharia de software atribuídas aos métodos são as primeiras a serem analisadas pelo módulo de avaliação automática de código-fonte. As métricas utilizadas para avaliar os métodos são: complexidade ciclomática e o número de linhas.

Para que as métricas de engenharia de software presentes no método sejam calculadas, implementou-se uma classe privada (uma extensão da classe *VoidVisitorAdapter*) para obter alguns dados através dos métodos utilizados pelas classes *MethodDeclaration* e *ConstructorDeclaration*, definidas como parâmetro no método *visit* que a classe *VoidVisitorAdapter* disponibiliza. A tabela 05 apresenta os métodos utilizados para extrair as informações necessárias ao cálculo das métricas de engenharia de software apresentadas.

**Tabela 05:** Métodos utilizados para calcular as métricas dos métodos

| Métodos | Descrição |
|---|---|
| *getName()* | Devolve o nome do método ou construtor visitado pelo *JavaParser*. |
| *getType()* | Devolve o tipo do método visitado pelo *JavaParser* (i.e. *int*, *boolean*, *String*, etc.). |
| *getParameters()* | Devolve todos os parâmetros do método ou construtor visitado pelo *JavaParser*, incluindo o respectivo tipo. |
| *getBody()* / *getBlock()* | Devolve o conteúdo do método e do construtor respectivamente de modo a que possa ser analisado mais tarde. |

A complexidade ciclomática define a quantidade de caminhos que o algoritmo produzido tem. Esta métrica é calculada através da análise de cada palavra do conteúdo do método recorrendo à

informação atribuída pelo método *getBody()*/*getBlock()*. A figura 22 mostra a implementação do método responsável pela análise da métrica complexidade ciclomática.

**Figura 22:** Função para calcular a complexidade ciclomática de um método

```
private int calculaComplexidadeMcCabe(final String
codigo_fonte) {
	int complexidade_ciclomatica = 1;
	String[] palavras = {"if", "while", "case", "for", "&&",
		"||", "?", ":", "catch"};
	String palavra = "";
	StringTokenizer token = new
		StringTokenizer(codigo_fonte);
	while (token.hasMoreTokens()) {
		palavra = token.nextToken();
		for (int i = 0; i < palavras.length; i++) {
			if (palavras[i].equals(palavra)) {
				complexidade_ciclomatica++;
			}
		}
	}
	return complexidade_ciclomatica;
}
```

Fonte: Marcos Pinto (2012)

O número de linhas de um método é calculado também recorrendo ao método *getBody()*/*getBlock()*. O resultado final desta métrica não incluí linhas em branco nem linhas que sejam comentários ao código, apresentando apenas o número de linhas utilizado para o funcionamento efectivo do método.

**Análise das métricas através da classe *MetricasClasse*:**

À semelhança das métricas de engenharia de software atribuídas à análise de cada método, as classes também permitem a análise das métricas da complexidade ciclomática e do número de linhas. No entanto, acrescem a estas métricas, o número de métodos e o número de atributos.

Também na análise das métricas de engenharia de software presentes em cada classe optou-se por implementar uma classe privada, com extensão da classe *VoidVisitorAdapter*, para obter o número de atributos através do método utilizado pela classe *VariableDeclaratorId*, definida como parâmetro no método *visit* disponibilizado através da classe *VoidVisitorAdapter*.

A métrica complexidade ciclomática atribuída a cada classe é o resultado da soma da mesma métrica para cada método. O número de linhas de cada classe é calculado com base no código compilado através da classe *CompilationUnit* inserida na *framework JavaParser*, excluindo as linhas em branco e as linhas que estão em comentário. A métrica número de métodos é calculada através da soma dos métodos presentes na classe, tendo como base uma lista com todos os métodos

encontrados. Por último, a métrica número de atributos é calculada através da análise à classe recorrendo à informação atribuída pelo método *getName()* que devolve todos os nomes das variáveis visitadas pelo *JavaParser*.

**Análise das métricas através da classe *MetricasPackage*:**

As métricas de engenharia de software atribuídas aos pacotes não são contabilizadas para o processo de avaliação da unidade curricular de Introdução à Programação, pois esta matéria está mais vocacionada para alunos que tenham conhecimentos mais avançados da linguagem de programação Java. Contudo, estas métricas foram implementadas no módulo de avaliação automática fornecendo a capacidade ao docente de alterar pequenas partes do módulo caso pretenda utilizar os valores das métricas no processo de avaliação.

A avaliação de cada pacote é feita através das seguintes métricas de engenharia de software: número de classes, número de métodos, complexidade ciclomática, número de linhas e número de atributos. O cálculo de cada uma das métricas presentes no processo de avaliação de cada pacote transmite-se pela utilização de métodos responsáveis por fazerem a soma de cada uma das métricas, isto é, cada uma das cinco métricas é calculada através da soma das métricas correspondentes.

Apesar das métricas calculadas através da análise de cada pacote não influenciarem a nota final do aluno no processo de avaliação através de uma percentagem atribuída à análise de cada pacote, são utilizadas para calcular as métricas do projecto.

**Análise das métricas através da classe *MetricasProjecto*:**

O módulo de avaliação automática de código-fonte avalia as métricas de engenharia de software do projecto com base em métodos implementados na classe *MetricasProjecto*. As métricas utilizadas na avaliação do algoritmo e contabilizadas no processo de avaliação são: número de classes, número de métodos, complexidade ciclomática, número de linhas e o número de atributos.

A figura 23 apresenta a função utilizada para calcular a complexidade ciclomática de um projecto submetido pelo aluno.

**Figura 23:** Função para calcular a complexidade ciclomática de um projecto

```
private int calculaComplexidadeCiclomatica() {
        int temp = 0;
        for(int i = 0; i != analise_package.size(); i++){
                temp += analise_package.get(i).getComplexidadeCiclomatica();
        }
        return temp;
}
```

Fonte: Marcos Pinto (2012)

À semelhança da função apresentada na figura acima, todas as outras métricas de engenharia de software também utilizam funções que permitem acumular a soma da respectiva métrica através da análise feita em cada pacote do projecto. As métricas definidas para um projecto, ao englobarem as métricas definidas para as classes e para os métodos, podem ser apenas alvo do processo de avaliação da unidade curricular de Introdução à Programação caso o docente assim o deseje.

Depois de implementadas as classes responsáveis pela análise das métricas de engenharia de software, recorreu-se à ferramenta *Metrics* para verificar se as métricas estavam a ser bem calculadas pelo módulo através da comparação de valores. A ferramenta *Metrics* fornece um conjunto de métricas de engenharia de software (i.e. número de linhas, número de caracteres, número de comentários, etc.) e pode ser incorporada no Eclipse SDK.

### 4.3.4 Atribuição de *Feedback*

Depois do módulo de avaliação automática executar um conjunto de testes e analisar todas as métricas de engenharia de software, produz um *feedback* para que o aluno consiga fazer a auto-avaliação. Deste modo, o docente não avalia os exercícios de forma individual, guardando esses recursos para a inserção de novos exercício cuja teoria não esteja bem desenvolvida pelos alunos.

Para a atribuição do *feedback* ao aluno, o módulo de avaliação automática recorre a um conjunto de expressões (capítulo III - secção 3.4, sub-secção 3.4.1.3) que permitem calcular a nota final atribuída à resposta do exercício respectivo. O conteúdo do *feedback* é definido pelo docente através das percentagens que insere no sistema com o enunciado do exercício.

O *feedback* é atribuído através da função *devolveFeedbackCompleto* consoante o resultado da compilação do código-fonte do aluno, da nota da execução automática dos testes e das percentagens atribuídas às métricas de engenharia de software. Numa primeira fase o módulo analisa se existem erros de compilação. Caso não existam erros de compilação, o módulo verifica se a nota atribuída à análise dos testes é superior a zero valores e, caso exista uma nota superior a zero valores para o resultado dos testes, é atribuído o relatório do projecto, da classe e do método (caso cada uma das percentagens seja superior a 0% respectivamente) ao relatório completo produzido.

Através do sistema ACode, o aluno tem a possibilidade de analisar duas formas do *feedback* atribuído pelo módulo: *feedback* resumido e *feedback* completo. Na figura 24 está apresentado um *feedback* reduzido em que o aluno pode verificar qual o resultado da sua resposta.

**Figura 24:** Feedback reduzido visível no sistema ACode


Fonte: Marcos Pinto (2012)

Contudo, o aluno pode analisar um *feedback* mais completo (atribuído pelo módulo) da sua resposta. A figura 25 apresenta parte do relatório produzido pelo módulo de avaliação automática.

**Figura 25:** Relatório apresentado no sistema ACode


Fonte: Marcos Pinto (2012)

Através da análise da figura acima, verificamos que o aluno pode analisar o resultado final métrica-a-métrica dando a possibilidade ao aluno de alterar a sua resposta em zonas especificas e submeter uma nova versão para um determinado exercício.

## *4.4 Servidor que suporta o sistema ACode*

Para suportar o funcionamento do sistema ACode é necessário configurar um servidor que suporte algumas características. Numa primeira fase, o servidor tem que suportar a linguagem de programação PHP e a linguagem de base de dados MySQL. Através destas duas linguagens, o sistema funciona minimamente. No entanto não permite avaliar o algoritmo produzido pelo aluno.

Para que o algoritmo seja avaliado, para além de existir o módulo no servidor (através da inserção de uma nova disciplina), tem que estar instalado o JDK 6 ou superior do Java. Neste caso, foi utilizado o *Java(TM) SE Development Kit 6* para que o código-fonte submetido pelo aluno seja avaliado correctamente.

# Capítulo V - Avaliação do Sistema ACode

## 5.1 Introdução

Após a implementação do sistema ACode e do módulo que permite avaliar o código-fonte do aluno de forma automática, é importante que sejam efectuados alguns testes para analisar se estes dois mecanismos se comportam da forma esperada. Na avaliação do sistema/módulo é importante simular algumas situações limite para verificar se os requisitos são cumpridos.

Para testar o sistema é fundamental recorrer a testes de eficácia e de usabilidade, não só, do sistema ACode, como também, do módulo responsável pela avaliação automática dos algoritmos produzidos pelos alunos. Os testes de eficácia são, na sua maioria, através da utilização do sistema e os testes de usabilidade são feitos através de inquéritos aos utilizadores.

É importante também que um sistema que suporte a avaliação dos alunos forneça algum desempenho. Como foi referido no capítulo III (secção 3.2, sub-secção 3.2.2) o sistema deve ser capaz de receber ficheiros de alunos, mesmo que o sistema esteja com muitos processos activos (i.e. o sistema está em sobrecarga). Normalmente, o sistema está em sobrecarga quando um conjunto de alunos submete ficheiros (resposta aos exercícios) no sistema no mesmo dia. Para testar o desempenho do sistema foi utilizado o ApacheBench com o intuito de verificar o tempo (em milissegundos) que cada pedido ao sistema demora.

Foi também alvo de estudo, a eficácia e a usabilidade do sistema e do módulo que permite a avaliação automática de código-fonte. Para tornar este estudo possível, recorreu-se às respostas dos exercícios presentes no sistema para teste e a um questionário entregue aos alunos depois do sistema testado.

Neste capítulo são apresentadas as características do servidor utilizado para o estudo (secção 5.2), um estudo sobre o desempenho do sistema (secção 5.3) e a avaliação do sistema (secção 5.4) que reúne os testes de eficácia (sub-secção 5.4.1) e os testes de usabilidade (sub-secção 5.4.2).

## 5.2 Características do Servidor

Para tornar o sistema ACode acessível para os utilizadores, foi necessário criar um servidor a partir do sistema operativo Microsoft Windows XP Professional com o Service Pack 3. O computador que suporta este sistema operativo apresenta um processador Intel(R) Pentium(R) 4 de 2,40 GHz e uma memória Ram de 504 MB.

O sistema ACode ficou disponível aos utilizadores através do WampServer (versão 2.2) cuja versão do apache é a 2.2.22. Uma vez que o sistema foi implementado na linguagem de programação PHP e na linguagem MySQL, instalou-se também no sistema operativo referido a versão 5.3.13 do PHP e a versão 5.5.24 do MySQL.

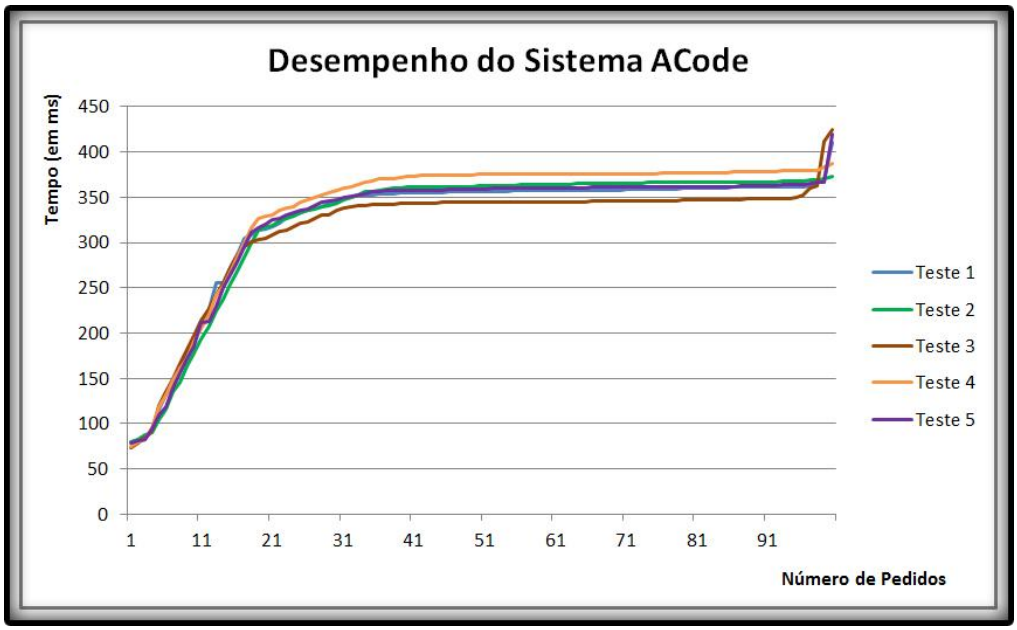
Para que o módulo de avaliação automática de algoritmos Java funcione correctamente é necessário instalar também o Java(TM) SE Development Kit na versão 6 ou superior.

## *5.3 Desempenho do Sistema*

Como foi referido no início deste capítulo, para avaliar o sistema ACode quanto ao seu desempenho, utilizou-se o ApacheBench. O ApacheBench é uma ferramenta que devolve os pedidos por segundo que o servidor, onde está hospedado um determinado domínio, suporta. Para testar o desempenho do sistema ACode utilizou-se cinco testes para obter resultados mais realista do desempenho do servidor.

Através do comando `ab.exe -n 500 -c 100 -e ficheiro.csv http://acode.no-ip.org/` executado na linha de comandos cinco vezes conseguimos obter o resultado apresentado na figura 26 (após extrair os dados do ficheiro ficheiro.csv). Através do comando apresentado conseguimos extrair a informação do número de pedidos efectuados ao site (neste caso concreto foram feitos 500 pedidos) e o número de pedidos paralelos a serem executados (neste caso concreto foram feitos 100 pedidos paralelos).

**Figura 26:** Desempenho do sistema ACode




Fonte: Marcos Pinto (2012)

Através da análise do gráfico da figura anterior podemos concluir que o desempenho do sistema é bom, considerando que, por cada 100 pedidos, o sistema apresenta um tempo de resposta de 425 milissegundos aproximadamente (0,425 segundos). Podemos verificar também que ao longo dos cinco testes efectuados ao sistema a curva do desempenho mantém-se praticamente igual, o que nos permite comprovar a eficiência do servidor.

## *5.4 Avaliação do Sistema*

Com a finalidade de testar o sistema ACode e o módulo que permite avaliar o algoritmo de forma automática, foram seleccionados um conjunto de exercícios ligados à unidade curricular de Introdução à Programação, cuja linguagem é Java, do ISCTE - Instituto Universitário de Lisboa. Desta forma, procurou-se manter o grau de dificuldade dos exercícios atribuídos aos alunos nas aulas práticas. O período de utilização do sistema foi oito dias.

Durante a avaliação do sistema, foram disponibilizados aos alunos um total de quatro exercícios, estando cada exercício elaborado de forma a especificar e a abranger um determinado conteúdo da unidade curricular de Introdução à Programação. Os exercícios utilizados na avaliação deste sistema estão apresentados no anexo F deste trabalho (exercícios 1, 2.1, 2.3 e 4).

A fim de poder testar este sistema e poder contar com a participação de alguns alunos da unidade curricular de Introdução à Programação, o sistema ACode foi disponibilizado via internet. Procedeu-se a este mecanismo uma vez que a fase de testes do sistema ocorreu fora do período lectivo. Com isto, os alunos foram contactados e mostraram-se receptivos, não só, a testar o sistema ACode, como também, no preenchimento de um questionário a fim de avaliar alguns aspectos de usabilidade do sistema (i.e. analisar se o sistema é simples e intuitivo).

Nesta secção, e em conformidade com os objectivos propostos deste trabalho, estão apresentados os testes de eficácia (sub-secção 5.4.1) e os testes de usabilidades (sub-secção 5.4.2) que foram alvo de estudo após a submissão das respostas dos alunos e do preenchimento de um questionário (referido anteriormente).

### 5.4.1 Testes de Eficácia

Para verificar que o sistema ACode satisfaz todos os requisitos apresentados, foram seleccionados quatro exercícios para serem respondidos pelos alunos. Estes exercícios estão directamente ligados à prática da programação Java leccionada na unidade curricular de Introdução à Programação no ISCTE - Instituto Universitário de Lisboa.

São diversas as matérias que os exercícios seleccionados apresentam para que o módulo que permite a avaliação do código-fonte também seja testado quanto à sua eficácia na atribuição do *feedback* do aluno. A recursividade, a declaração de vectores, condições e ciclos são algumas matérias presentes nos exercícios disponibilizados através do sistema ACode. Para melhor analisar a eficácia destas ferramentas, está apresentado um exemplo que recorre a um exercício (exercício 2.1 do anexo F, página 84) e cinco respostas de alunos.

**Exemplo para um exercício e respostas de cinco Aluno**

Para testar a eficácia do módulo responsável pela a avaliação dos algoritmos submetidos pelo aluno, seleccionou-se um exercício e cinco respostas. O exercício utilizado para este estudo engloba parte da matéria leccionada na unidade curricular, nomeadamente matéria cuja utilização dos vectores está associada.

Nas tabelas 6, 7, 8, 9 e 10 estão apresentados os resultados obtidos por cinco dos alunos que testaram o sistema. Para cada aluno está apresentada a nota atribuída para cada submissão efectuada. Foram escolhidas as cinco primeiras respostas (fornecidas ao sistema pelos alunos) a fim de verificar a evolução da resposta de cada aluno.

**Tabela 06:** Submissão da resposta do Aluno 1

| Submissões | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Nota Atribuída | 9 | 13 | 16 | 19 | 20 |

**Tabela 07:** Submissão da resposta do Aluno 2

| Submissões | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Nota Atribuída | 6 | 12 | 15 | 17 | 20 |

**Tabela 08:** Submissão da resposta do Aluno 3

| Submissões | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Nota Atribuída | 12 | 15 | 12 | 16 | 19 |

**Tabela 09:** Submissão da resposta do Aluno 3

| Submissões | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Nota Atribuída | 8 | 14 | 17 | 18 | 20 |

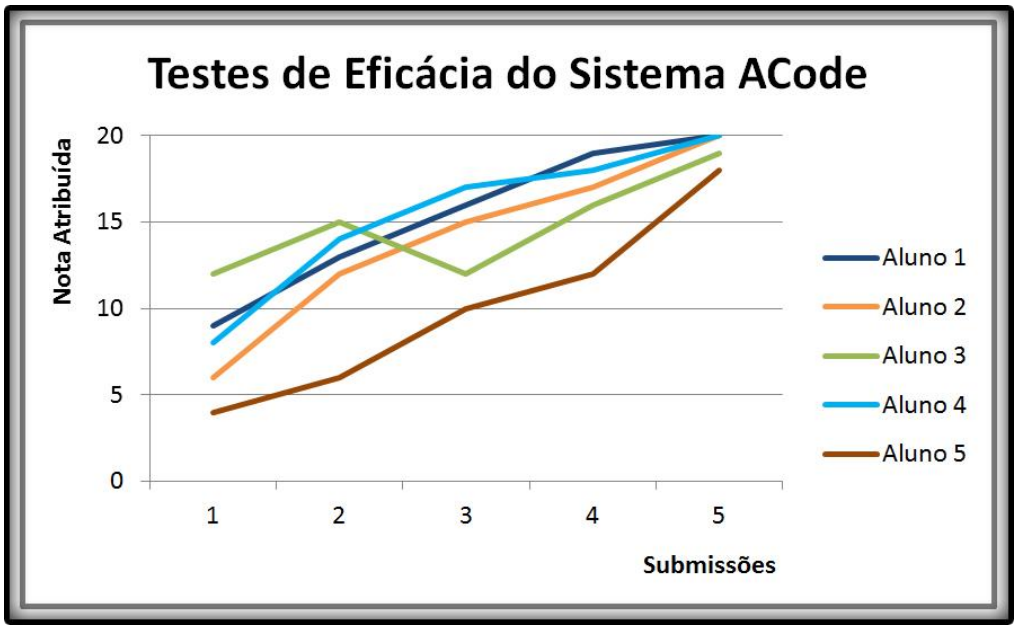
**Tabela 10:** Submissão da resposta do Aluno 4

| Submissões | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Nota Atribuída | 4 | 6 | 10 | 12 | 18 |

Para analisar melhor as tabelas acima, organizou-se os dados num gráfico a fim de obter uma informação relevante sobre a eficácia do sistema ACode e do módulo que permite a avaliação automática de código-fonte Java submetido pelos alunos.

A figura 27 apresenta os dados extraídos das tabelas e compreende as respostas fornecidas por cinco alunos a um exercício utilizado durante a unidade curricular de Introdução à Programação.

**Figura 27:** Submissão de respostas para 5 Alunos




Fonte: Marcos Pinto (2012)

Através da análise do gráfico apresentado na figura 27 verificamos que os alunos evoluem com as respostas submetidas no sistema ACode. Esta evolução deve-se ao facto do sistema permitir a entrega de respostas num período de tempo, sendo cada uma delas analisada e avaliada pelo sistema automaticamente. Com isto, o aluno consegue obter *feedback* das suas respostas e, caso o *feedback* não seja o esperado pelo aluno, pode voltar a submeter respostas até que o *feedback* seja o esperado.

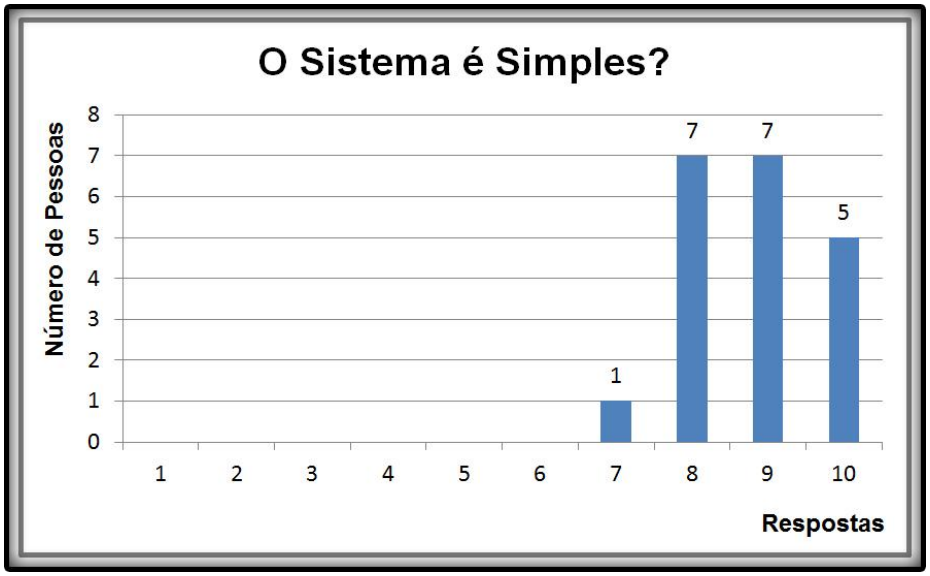
## *5.4.2 Testes da Usabilidade*

Os testes de usabilidade são importantes para validar uma interface ou identificar problemas existentes no sistema através de um grupo de utilizadores. Os testes de usabilidade apresentam alguns benefícios e vantagens em relação a outro tipo de testes: o comportamento dos utilizadores pode ser observado e comparado com outros utilizadores quando realizam a mesma tarefa no sistema, a compreensão das dificuldades sentidas pelos utilizadores podem ser comunicadas directamente a quem faz o teste, etc. No entanto, como os testes de usabilidade foram feitos através da internet, a análise é feita através de um questionário efectuado a 20 utilizadores do sistema.

O questionário foi aplicado com o intuito de analisar a avaliação atribuída pelo aluno aos recursos visíveis no sistema, ou seja, analisar se o sistema é simples e intuitivo. As questões presentes no formulário foram objectivas e as respostas foram organizadas através de uma escala numerada de 1 a 10 e através de respostas cuja opção era *sim* ou *não*. No final do questionário foi atribuído um campo para que o aluno descreve-se alguma opinião/sugestão sobre o sistema utilizado.

Depois de 20 alunos responderem ao questionário, organizou-se um gráfico para que a informação seja visível facilmente. Na figura 28 e 29 estão apresentados dois gráficos onde as respostas indicam se o sistema é simples ou intuitivo, respectivamente.

**Figura 28:** Resposta ao questionário: simplicidade do sistema




Fonte: Marcos Pinto (2012)

**Figura 29:** Resposta ao questionário: sistema intuitivo

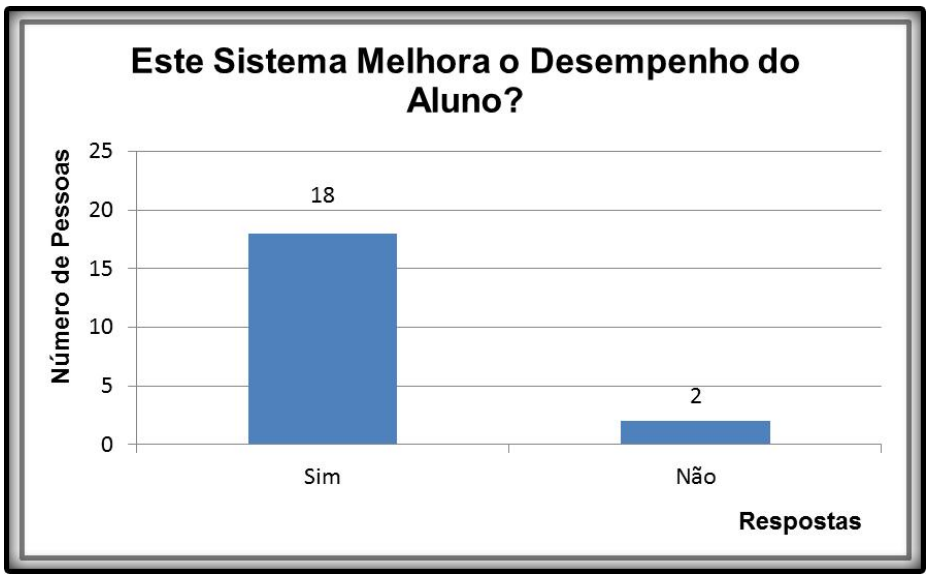

Fonte: Marcos Pinto (2012)

De acordo com os gráficos apresentados, podemos concluir que o sistema ACode é simples pois apresenta resultados nas três notas mais altas (mais respostas na escala 8, 9 e 10). Através disto, conseguiu-se implementar um sistema que ajuda o aluno na avaliação do código-fonte desenvolvido, e criar uma forma simples de obter o resultado dessa avaliação, permitindo uma concentração superior no desenvolvimento da resposta ao enunciado do exercício.

No gráfico apresentado na figura 29, os alunos responderam a uma pergunta onde podemos analisar se o sistema é ou não intuitivo. Da informação que conseguimos extrair do gráfico apresentado, podemos concluir que o sistema ACode poderia ser mais intuitivo. As respostas atribuídas pelos alunos variam muito entre a escala 6 e 10 (de uma escala de 1 a 10 em que à escala 1 é atribuído o texto nada intuitivo e à escala 10 muito intuitivo). Apesar disto, e através de alguma exploração do sistema, os alunos conseguiram submeter as suas respostas e analisar o *feedback* atribuído pelo sistema.

Foi também alvo de estudo, através do questionário, se o sistema ACode melhora o desempenho do aluno na unidade curricular de Introdução à Programação. Na figura 30 está apresentado um gráfico com as respostas fornecidas pelos alunos relativas a esta questão.

**Figura 30:** Resposta ao questionário: sistema melhora desempenho do aluno




Fonte: Marcos Pinto (2012)

Como podemos verificar através do gráfico, 90% dos alunos referem que este tipo de sistema melhora o seu desempenho na unidade curricular de Introdução à Programação. Em estudos feitos anteriormente e apresentados no capítulo II (secção 2.2), os sistemas que aliviam o trabalho do docente através da avaliação automática de algoritmos ajuda no bom aproveitamento dos alunos.

No questionário esteve também presente uma questão referente ao uso do sistema para a avaliação do aluno na unidade curricular. A figura 31 apresenta os resultados obtidos através do questionário.

**Figura 31:** Resposta ao questionário: sistema melhora a avaliação


Fonte: Marcos Pinto (2012)

Através da análise do gráfico verificamos que 95% dos alunos usariam este sistema no processo de avaliação da unidade curricular. Este estudo torna-se de algum forma positivo pois, através de sistemas que têm como base a avaliação automática de algoritmos, é possível melhorar o desempenho do aluno como apresentado anteriormente (sub-secção 5.4.1 deste capítulo).

## *5.5 Análise de Resultados*

Através da análise dos dados fornecidos pelo sistema e dos dados fornecidos pelos alunos através do preenchimento de um questionário, pode-se concluir que a ferramenta obteve sucesso. Os alunos que obtiveram nota máxima através da utilização do sistema ACode foi quase 100% o que permite comprovar que este tipo de sistemas melhoram significativamente o desempenho dos alunos na prática da programação Java.

Ao número reduzido de respostas erradas (fornecidas pelos alunos) podem ser atribuídos dois factores. O primeiro factor centra-se na formulação do enunciado, ou seja, o enunciado pode não estar devidamente especificado e/ou completo. O segundo factor centra-se nas capacidades dos alunos no acto da implementação de um novo algoritmo, isto é, estando o sistema ACode vocacionado para alunos de Introdução à Programação, muitos são aqueles que cometem alguns erros comuns na lógica de programação.

Através do estudo efectuado com os alunos, pode ser acrescentado que o sistema ACode está apto para dar apoio nas aulas práticas de Introdução à Programação e a trabalhos que incorporem o processo de avaliação da unidade curricular de Introdução à Programação. Neste último caso, o enunciado ao exercício deve estar bem definido e sem contradições para que seja perceptível ao aluno o que é pedido.

# Capítulo VI - Conclusões e Trabalho Futuro

## *6.1 Conclusão*

A presente dissertação baseou-se principalmente na implementação de um sistema Web e um módulo que permite avaliar o código-fonte do aluno para poder dar resposta à questão formulada inicialmente:

> *"Como analisar e avaliar o algoritmo produzido pelo estudante de forma a que este receba o respectivo feedback em tempo útil?"*

A utilização dos sistemas, que permitem avaliar o código-fonte submetido pelo aluno de forma automática, é útil para aliviar o trabalho do docente na avaliação individual do aluno. Este tipo de sistemas primam pela velocidade na atribuição do *feedback* para cada resposta fornecida pelo aluno ao sistema. Contudo, muitos destes sistemas não cumprem todos os requisitos, considerados importantes, para a utilização em unidades curriculares de Introdução à Programação.

Foram identificados alguns requisitos neste trabalho (capítulo III, secção 3.2) que sustentam a forma como o sistema efectua a avaliação automática utilizada em cursos que requerem a prática de linguagens de programação (neste caso concreto a avaliação automática de código-fonte Java). A submissão de ficheiros, a definição de testes e de métricas de engenharia de software, a disponibilidade e a segurança são alguns dos requisitos identificados ao longo deste trabalho.

A partir desta estruturação de ideias, implementou-se um mecanismo que dê resposta às necessidades dos alunos e dos docentes afim de melhorar a atribuição de *feedback* e o desempenho do aluno.

Uma vez que este sistema não foi utilizado nas aulas práticas da unidade curricular de Introdução à Programação, não é possível analisar correctamente a sua eficiência no que diz respeito à melhoria do desempenho de cada aluno. No entanto os resultados apresentados foram obtidos através de alunos que frequentaram esta unidade curricular a fim de testar alguns pressupostos (submissão de ficheiros, produção de um *feedback* consoante a resposta fornecida ai sistema, etc.).

Através da análise do capítulo anterior verifica-se que o sistema ACode, apesar de englobar todos os requisitos formulados ao longo do trabalho, apresenta também um desempenho satisfatório quando existem vários utilizadores ligados simultaneamente ao sistema. Além disto, através da utilização do sistema, podemos concluir que este tipo de mecanismos conferem ao estudante a capacidade, não só, de fazer a auto-avaliação, como também, de melhorarem o seu desempenho na unidade curricular de Introdução à Programação. A partir do questionário entregue aos alunos, é

possível verificar que o sistema ACode é simples e intuitivo possibilitando ao aluno uma concentração superior na formulação da resposta aos exercícios fornecidos pelos docentes.

Em suma, o sistema desenvolvido aceita as respostas dos alunos através de um ficheiro (com o formato *.zip*) ou de uma resposta fornecida através de texto (algoritmo com uma resposta ao respectivo problema). Depois disto, a resposta do aluno é enviada para o módulo responsável pela avaliação automática que devolve um relatório completo, indicando a nota final atribuída. Este relatório fica disponível ao aluno sempre que analisar a respectiva resposta submetida no sistema. Caso o aluno não fique contente com o resultado da sua resposta, pode voltar a responder ao exercício desde que o prazo de entrega deste não tenha excedido.

Esta dissertação permitiu alargar o espectro de conhecimento sobre a linguagem de programação PHP. Este trabalho pode servir de base para futuras implementações de novos módulos ou para melhor o sistema Web existente,

## *6.2 Contribuições*

Actualmente existem alguns sistemas que efectuam a avaliação automática de código-fonte desenvolvido pelo aluno. Contudo, esta correcção baseia-se fundamentalmente na comparação entre os dados produzidos pelo algoritmo do aluno e os dados inseridos no sistema pelo docente.

Esta dissertação de mestrado visa melhorar o conceito do mecanismo de avaliação que os sistemas actuais apresentam, através da implementação de um novo sistema capaz de avaliar os algoritmos, não só, através da comparação dos dados produzidos pelo código-fonte do aluno com os dados inseridos no sistema pelo docente, mas também, através de algumas métricas de engenharia de software.

Este sistema, tal como os outros, apresenta melhorias no desempenho dos alunos da unidade curricular de Introdução à Programação e, acresce à capacidade do aluno, efectuar a auto-avaliação dos seus conhecimentos de programação. Contudo, este sistema, apresenta resultados mais realistas tanto para o docente, como para o aluno, uma vez que o código-fonte submetido não é analisado simplesmente através da execução de testes. Com isto, este sistema contribui para o avanço da literatura relacionada com o tema e para aliviar, de certa forma, as tarefas do docente na atribuição de notas individuais dos exercícios propostos.

## *6.3 Limitações*

Durante a realização desta dissertação de mestrado foram identificadas algumas limitações, não só, inerentes ao funcionamento do sistema, como também, a recursos humanos essenciais para a

análise critica do sistema. A principal limitação prendeu-se com a baixa adesão dos alunos em testar este sistema de avaliação automática. Esta limitação foi encontrada pois o sistema esteve disponível para testes num período de férias. Com base nisto, foram poucos os alunos que testaram o sistema e preencheram o questionário referente à avaliação qualitativa do sistema. Outra limitação encontrada referente aos recursos humanos, foi a não disponibilização, em tempo útil, de um servidor que permitisse o funcionamento correcto do sistema. Esta limitação foi ultrapassada através da configuração de um servidor pessoal.

Foram encontradas algumas limitações inerentes ao funcionamento do sistema como o cálculo do tempo de execução de cada teste, a não avaliação de código replicado e a presença de um sistema básico. Existem soluções para encontrar código-fonte replicado (i.e. PMD - analisa código fonte), pelo que esta funcionalidade não foi um objectivo inicial desta dissertação de mestrado. Relativamente ao sistema, apesar de ser um protótipo que suporte várias unidades curriculares, a submissão das respostas apenas pode ser utilizada para inserir código-fonte Java.

## *6.4 Trabalho Futuro*

Com base nas limitações encontradas, derivam-se algumas recomendações para trabalhos futuros no sistema ACode.

As primeiras recomendações centram-se nas limitações encontradas. A principal recomendação transmite-se por arranjar um servidor interno do ISCTE - Instituto Universitário de Lisboa, para que o sistema ACode possa ser utilizado nas aulas práticas da unidade curricular de Introdução à Programação. É importante também que seja melhorado o sistema para que seja possível aceitar registo de grupos. Com isto, o sistema ACode pode também ser utilizado para avaliar os trabalhos em grupo da unidade curricular de Introdução à Programação.

A implementação de um editor de texto semelhante ao eclipse, isto é, um editor de texto que permita destacar as palavras reservadas do Java e os erros encontradas no algoritmo em tempo real, também seria um mecanismo interessante a ser inserido no sistema ACode. Uma vez que este sistema foi pensado para várias unidades curriculares, seria também interessante implementar um módulo que permita o reconhecimento de diagramas UML para poder avaliar exercícios da unidade curricular Fundamentos de Base de Dados.

O módulo implementado e que fornece o *feedback* ao aluno indica mensagens de erro da compilação do código-fonte submetido. Estas mensagens de erro também requerem uma melhoria pois são pouco esclarecedoras para o aluno. Apenas indicam qual a classe onde está o erro, qual o símbolo errado e a linha que deve ser alterada. Seria interessante mostrar o erro completo. Esta abordagem foi pensada ao longo do desenvolvimento do módulo, contudo, não foi implementada

pois, com isto, a visualização dos casos de teste JUnit seriam visualizados pelo aluno caso existisse algum erro na classe e/ou método a ser analisado.

Seria importante também que o sistema apresentasse uma funcionalidade de converter as notas dos alunos para um ficheiro Excel. Com isto, o docente facilmente utilizava as notas atribuídas pelo sistema ao processo de avaliação da unidade curricular.

Por último, uma recomendação a ser analisada é a tentativa de poder ser alterada a solução inserida no sistema pelo docente por outra, desenvolvida pelo aluno, que apresentasse melhores resultados ao nível das métricas de engenharia de software.

# Referências Bibliográficas

**Almeida, Eliana S. de, et al. (2011). "**AMBAP: Um Ambiente de Apoio ao Aprendizado de Programação". Online. Consultado em 26 de Janeiro de 2012. Disponível no link: http://www.ic.unicamp.br/~atanasio/PUB/artigos/2002SbcAmbap.pdf.

**Ashoo, Samir E, Boudreau, Troy e Lane, Douglas A. (2011). "**$PC^2$ Documentation - Version 9.1 What's New Document". Online. Consultado em 26 de Janeiro de 2012. Disponível no link: http://www.ecs.csus.edu/pc2/doc/v9/WhatsNewInV9.1.pdf.

**Bass, Len, Clements, Paul, Rick Kazman (2003).** "Software Architecture in Practice". 2ª Edição. Addison-Wesley Professional, 2003 - 528 páginas. ISBN: 978-0321154958.

**Beck, Kent, Gamma, Erich.** JUnit. Consultado em 22 de Fevereiro de 2012. Disponível no Link: http://www.junit.org/.

**Bravo, Crescencio, Marcelino, Maria Jose, Gomes, Anabela, Esteves, Micaela, Mendes, Antonio Jose (2005).** "Integrating Educational Tools for Collaborative Computer Programming Learning" em "Journal of Universal Computer Science". Vol. 11, no. 9. ISSN 0948-6968. pp. 1505-1517.

**Cardoso, Jorge (2006).** "Process control-flow complexity metric: An empirical validation" em "IEEE International COnference on Services Computing". Setembro de 2006. ISBN: 0-7695-2670-5, pp. 167-173.

**Company, The New York Times. (2012).** "About.com Linux". Online. Consultado em 27 de Janeiro de 2012. Disponível no link: http://linux.about.com/library/cmd/blcmdl1_diff.htm.

**Dalziel, James. (2003).** "Implementing Learning Design: The Learning Activity Management System (LAMS)". Online. Consultado em 26 de Janeiro de 2012. Disponível no link: http://www.lamsinternational.com/CD/html/resources/whitepapers/ASCILITE2003%20Dalzie%20Final.pdf.

**Fischer, Alice E., Grodzinsky, Frances Schlamowitz (1993).** "The Anatomy of Programming Languages". Prentice Hall, 1993, 557 páginas. ISBN 0-13-035155-5, p. 4.

**García-Mateos, Ginés e Fernández-Alemán, José Luis. (2009).** "A Course on Algorithms and Data Structures Using On-line Judging" em "ITiCSE '09 Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education". Vol. ITiCSE '09, 6-9 de Julho de 2009. ACM New York, 2009. ISBN: 978-1-60558-381-5. pp. 45-49.

**Gomes, Anabela, Mendes, António José e Marcelino, Maria José. (2004).** "Avaliação e Evolução de um Ambiente de Suporte à Aprendizagem da Programação" em "Congresso Ibero-Americano de Informática Educativa". Vol. VII. 2004. pp. 187-196.

**Hearst, Marti A. (2002).** "The debate on automated essay grading" em "Intelligent Systems and their Applications". Vol. 5. IEEE Computer Society, 2002. ISSN: 1094-7167. pp. 22-37.

**Horstmann, Cay (2002).** "Big Java". Porto Alegre: Bookman, 2004. 1125 páginas. ISBN: 85-363-0345-X.

**Leal, José Paulo e Queirós, Ricardo. (2009).** "Designing a User Interface for Repositories of Learning Objects" em "Proceedings of e-Learning 2009". Vol. 17-20 de Junho de 2009, Algarve, Portugal. IADIS - Internacional Association for Development of the Information Society, 2009. ISBN: 978-972-8924-83-6. pp. 137-140.

**Leal, José Paulo e Silva, Fernando. (2002).** "Managing programming contests with Mooshak" em "Eunis 2002 - The 8th International Conference of European University Information Systems". Data: 19-22 de Julho de 2002.

**Leal, José Paulo e Silva, Fernando. (2003).** "Mooshak: a Web-based multi-site programming contest system" em "Software: Practice and Experience". Vol. 33. Wiley Interscience, 2003. ISSN: 0038-0644. pp. 567-581.

**Manso, António, Oliveira, Luís e Marques, Célio Gonçalo. (2011).** "Ambiente de Aprendizagem de Algoritmos - Portugol IDE". Online. Consultado em 26 de Janeiro de 2012. Disponível no link: http://orion.ipt.pt/~manso/papers/2009/Portugol%20Challenges2009.pdf.

**McCabe, Thomas J. (1976).** "A Complexity Measure" em "IEEE Transactions on Software Engineering". Volume SE-2, N.º 4. Dezembro de 1976. ISSN: 0098-5589, pp. 308-320.

**Mendes, António José e Gomes, Anabela Jesus.** "Suporte à aprendizagem da programação com o ambiente SICAS". Online. Consultado em 26 de Janeiro de 2012. Disponível no link: http://lsm.dei.uc.pt/ribie/docfiles/txt20037292359Suporte%20%C3%A0%20aprendizagem.pdf.

**Moreira, Mirelle Pinheiro e Favero, Eloi Luiz. (2009).** "Um Ambiente para Ensino de Programação com Feedback Automático de Exercícios" em "WEI - Anais do Workshop sobre Educação em Informática". Vol. XVII. SBC – Sociedade Brasileira da Computação, 20-22 de Julho de 2009. ISSN: 2175-2761. pp. 429-438.

**Mota, Marcelle Pereira, Pereira, Lis W. Kanashiro e Favero, Eloi Luiz. (2008).** "JavaTool: Uma Ferramenta para Ensino de Programação" em "WEI – Workshop sobre Educação em Computação". Vol. XXVIII. SBC – Sociedade Brasileira da Computação, 12 a 18 de Julho de 2008. pp. 127-136.

**Nobre, Isaura A. M. e Menezes, Crediné S. de. (2002).** "Suporte à Cooperação em um Ambiente de Aprendizagem para Programação (SAmbA)" em "Anais do Simpósio Brasileiro de Informática na Educação". Vol. XIII. SBIE - Simpósio Brasileiro de Informática na Educação, 2002. ISSN: 2176-4301. pp. 337-347.

**Perego, Cássia Alves. (2002).** "JEduc: Reflexão sobre a Linguagem Java na Educação". Universidade Federal do Rio Grande do Sul - Porto Alegre, 2002. Dissertação de Mestrado em Ciência da Computação.

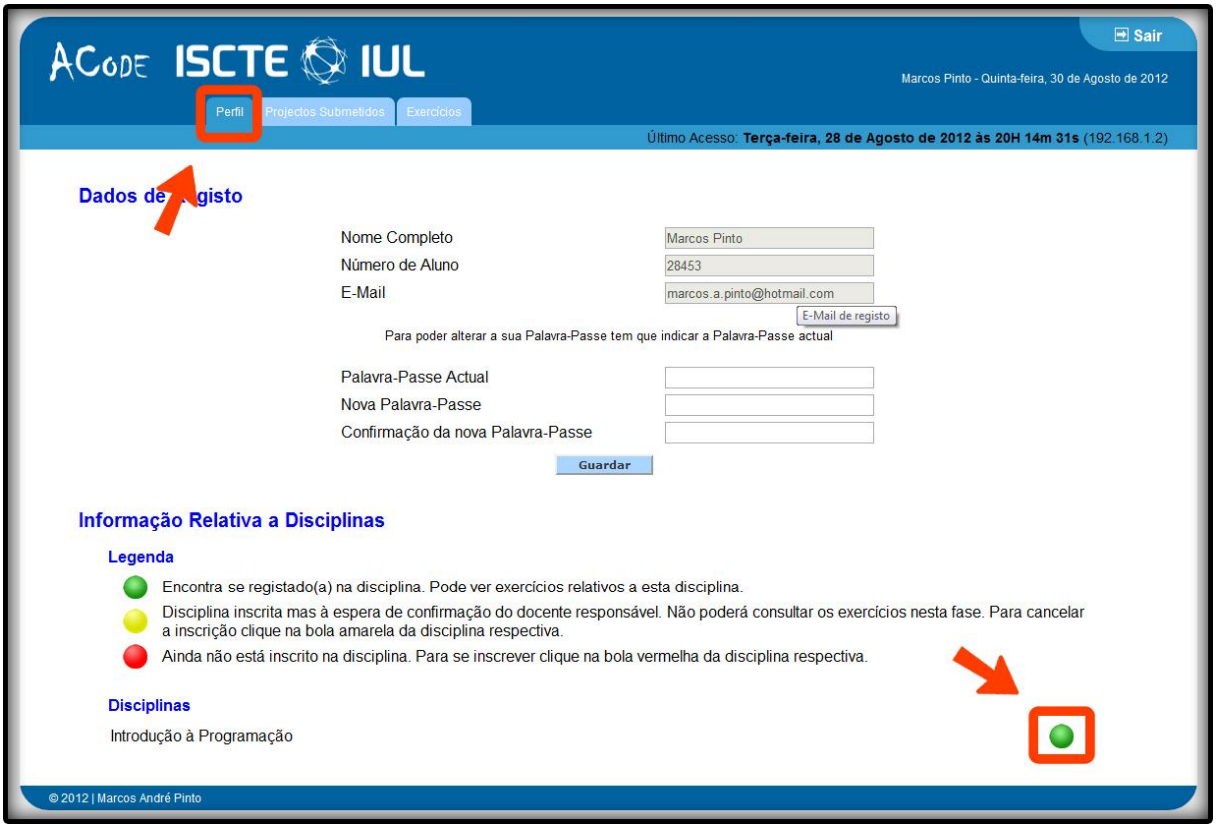
**Project Google**. Java 1.5 Parser and AST. Consultado em 11 de Janeiro de 2012. Disponível no Link: http://code.google.com/p/javaparser/.

**Revilla, Miguel A., Manzoor, Shahriar e Liu, Rujia. (2008).** "Competitive Learning in Informatics: The UVa Online Judge Experience" em "Olympiads in Informatics". Vol. 2. Institute of Mathematics and Informatics, Lithuania, 2008. ISSN: 1822-7732. pp. 131-148.

**Software, Edgewall. (2012).** "Fénix Framework". Online. Consultado em 27 de Janeiro de 2012. Disponível no link: https://fenix-ashes.ist.utl.pt/trac/fenix-framework.

# Anexo A: Inscrição na Unidade Curricular

**Figura 32:** Inscrição na Unidade Curricular




Fonte: Marcos Pinto (2012)

# Anexo B: Confirmação da Inscrição na Unidade Curricular

**Figura 33:** Confirmação da Inscrição na Unidade Curricular


Fonte: Marcos Pinto (2012)

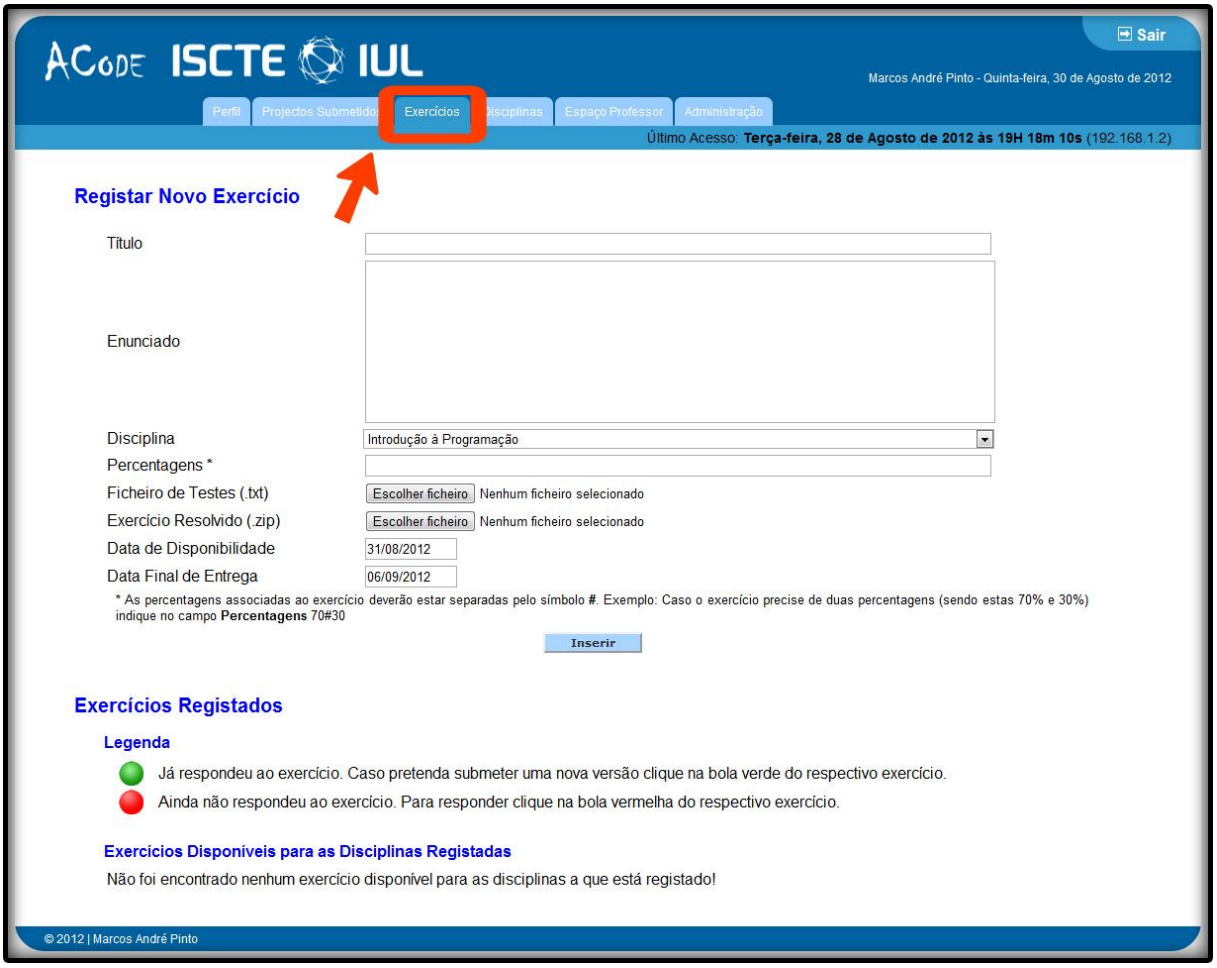
# Anexo C: Submeter Respostas a Exercícios

**Figura 34:** Submeter Respostas a Exercícios


Fonte: Marcos Pinto (2012)

# Anexo D: Inserir Exercícios no Sistema

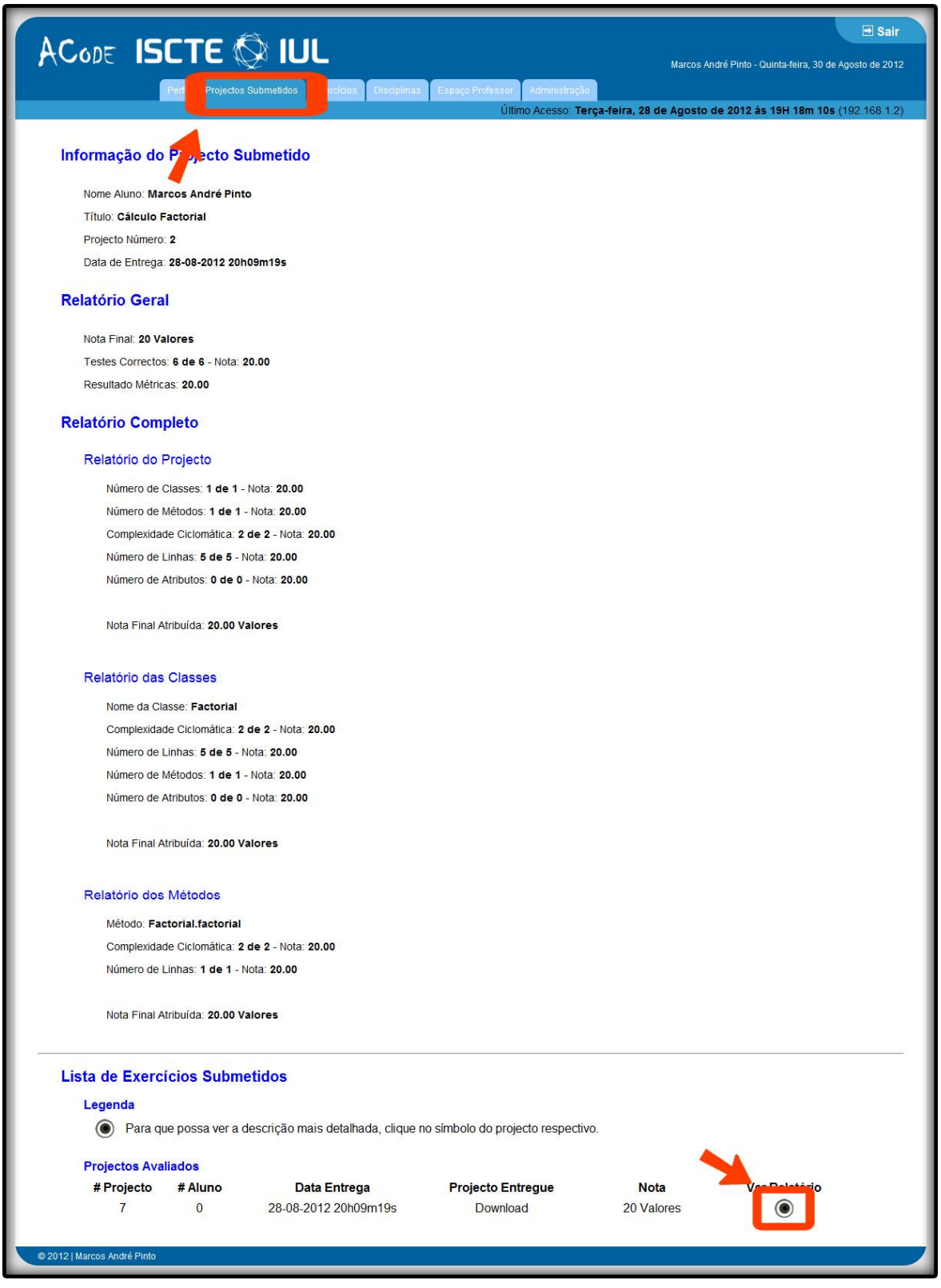
**Figura 35:** Inserir Exercícios no Sistema




Fonte: Marcos Pinto (2012)

# Anexo E: Análise do Relatório Completo

**Figura 36:** Análise do Relatório Completo




Fonte: Marcos Pinto (2012)

# Anexo F: Exercícios para Testar o Sistema

| ISCTE IUL Instituto Universitário de Lisboa | **Introdução à Programação**<br>**Ano Lectivo:** 2011/2012<br>Exercícios de Aprendizagem |
|---|---|

Está neste momento em curso um projecto - ACode - para avaliar, de forma automática, os algoritmos produzidos pelos alunos. Este projecto está associado a uma dissertação de Mestrado em Engenharia Informática (ramo Sistemas de Informação e Gestão do Conhecimento) e é fundamental a elaboração de testes, tanto de desempenho como da própria avaliação do código-fonte produzido pelo aluno.

Foram seleccionados alguns exercícios para que possam ser avaliados através do sistema. É fundamental resolver todos os exercícios presentes não só para a fase de testes deste projecto como também para praticar os conceitos e técnicas adquiridas ao longo do semestre de forma a preparar-se da melhor forma para a frequência/exame.

Todos os exercícios deverão ser organizados numa pasta de acordo com a seguinte estrutura:

```
– ACode_XX
    └ Ex01_XX
        └ Factorial.java
    └ Ex02_XX
        └ Vectores.java
    └ Ex03_XX
        └ Matriz.java
    └ Ex04_XX
        └ Aluno.java
        └ Disciplina.java
```

em que XX representa o número atribuído à bancada.

**Exercício 1**

Crie a classe `Factorial` e defina uma função recursiva para calcular o factorial de um número. Esta função deverá ter o nome `factorial` e irá receber um valor no argumento.

```
public int factorial(final int valor){
      //A completar...
}
```

**Exercício 2**

Crie a classe `Vectores` e implemente as seguintes funções:

**2.1** Defina uma função que permita obter a primeira metade de um vector `v` de números inteiros, incluindo o elemento central (caso o comprimento do vector seja ímpar). Esta função deverá ter o nome `primeiraMetadeVector` e irá receber um vector no argumento.

```
public int[] primeiraMetadeVector(final int[] v){
      //A completar...
}
```

**2.2** Defina uma função que produza o mesmo vector `v` de números inteiros sem o elemento central (caso o comprimento seja ímpar). Esta função deverá ter o nome `copiarVectorSemElementoCentral` e irá receber um vector no argumento.

```
public int[] copiarVectorSemElementoCentral(final int[] v){
      //A completar...
}
```

**2.3** Defina uma função que produza a média dos valores inteiros do vector `v`. Esta função deverá ter o nome `mediaValoresVector` e irá receber um vector no argumento.

```
public double mediaValoresVector(final int[] v){
      //A completar...
}
```

**2.4** Defina uma função que produza a quantidade de números primos existentes no vector `v`. Esta função deverá ter o nome `quantidadeNumerosPrimos` e irá receber um vector no argumento.

```
public int quantidadeNumerosPrimos(final int[] v){
      //A completar...
}
```

**Exercício 3**

Crie a classe `Matriz` e implemente as seguintes funções:

**3.1** Defina uma função que recebe como argumento uma matriz `m` e devolve a sua transposta. Esta função deverá ter o nome `calculaMatrizTransposta` e irá receber uma matriz no argumento.

*Nota:* A matriz transposta traduz-se na troca das linhas pelas colunas. Exemplo:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```
public int[][] calculaMatrizTransposta(final int[][] m){
      //A completar...
}
```

**3.2** Defina uma função que recebe como argumento uma matriz `m` e devolve a multiplicação de `m` com a sua transposta. Esta função deverá ter o nome `multiplicaMatrizComTransposta` e irá receber uma matriz no argumento.

```
public int[][] multiplicaMatrizComTransposta(final int[][] m){
      //A completar...
}
```

**Exercício 4**

**4.1** Crie a classe `Aluno` onde o `nome`, o `numero` e a `nota` (assuma que a `nota` apenas representa valores inteiros) são representadas como variáveis instância da classe. Crie uma instância da classe `Aluno` com o `nome`, o `numero` e a `nota` passados no argumento. Crie também os inspectores e os modificadores seguintes:

```
public String getNome(){
      //A completar...
}
public int getNumero(){
      //A completar...
}
public int getNota(){
      //A completar...
}
public void setNota(final int nota){
      //A completar...
}
```

**4.2** Crie a classe `Disciplina` que é composta por um vector de alunos (`alunos_disciplina`) e um nome da disciplina (`nome_disciplina`). O vector de alunos e o nome da disciplina são variáveis de classe.

**4.2.1** Defina um procedimento na classe `Disciplina` para adicionar um aluno `a` aos alunos da disciplina. Esta função deverá ter o nome `adicionaAluno` e irá receber um aluno no argumento. O aluno só será inserido caso ainda não esteja nos `alunos_disciplina`.

```
public void adicionaAluno(final Aluno a){
      //A completar...
}
```

**4.2.2** Defina um procedimento na classe `Disciplina` para remover um aluno `a` aos alunos da disciplinas. Esta função deverá ter o nome `removeAluno` e irá receber um aluno a no argumento.

```
public void removeAluno(final Aluno a){
      //A completar...
}
```

**4.2.3** Defina um procedimento na classe `Disciplina` para alterar a nota de um aluno `a`. Este procedimento deverá ter o nome `alteraNotaAluno` e irá ter como argumentos o `numero` do aluno e a `nota` actualizada.

```
public void alteraNotaAluno(final int numero, final int nota){
      //A completar...
}
```

**4.2.4** Defina uma função na classe `Disciplina` que devolva um vector com os 2 melhores alunos (se o vector `alunos_disciplina` estiver vazio, deverá devolver um vector vazio. Esta função deverá ter o nome `doisMelhoresAlunos`.

```
public Alunos[] doisMelhoresAlunos(){
      //A completar...
}
```

**4.2.5** Defina uma função na classe `Disciplina` que devolva o vector `alunos_disciplina` ordenado pela nota (se o vector `alunos_disciplina` estiver vazio, deverá devolver um vector vazio). Esta função deverá ter o nome `ordenaPelaNota`.

```
public Alunos[] ordenaPelaNota(){
      //A completar...
}
```

# Anexo G: Artigo Publicado

## ACode: Web-based System for Automatic Evaluation of Java Code

Marcos André Pinto[1], António Luis Lopes[2]

Instituto das Telecomunicações - Instituto Universitário de Lisboa, Lisboa, Portugal
[1] a28453@iscte.pt; [2] alsl@iscte.pt

**Abstract.** Most current approaches for automatic evaluation of source code use input/output testing to validate student-submitted solutions. However, very few use software engineering metrics to analyze source code. With the limitations of related work in mind, we present, in this paper, our 4-stage approach for automatic evaluation of source code: i) the code is first compiled and checked for any errors; ii) the compiled code is then tested against a set of JUnit tests provided by the teacher; iii) a set of software engineering metrics is used to compare the student's solution against the teacher's solution; iv) and finally, based on the previous stages, feedback is provided to the students so they can self-evaluate and identify the areas in which they need further study.

**Keywords:** Student feedback, automatic evaluation, web application, java code

### 1 Introduction

The success of first year students in learning programming languages and concepts is usually associated with hard work and frequent testing of their programming skills so as to assess their current knowledge. During evaluation periods, teachers provide programming exercises for which the students have to present solutions. Afterwards, the teacher grades these tests individually and it is through this grading feedback that students can self-evaluate and determine the concepts that need further study. Several systems exist that provide an automatic process to evaluate students' solutions, with Mooshak [1], a web-based system designed to support programming contests, being one of the most used. Other systems like PC2 (desktop application [2]), UVa Judge and EduJudge [3] also provide similar functionalities.
These approaches are capable of evaluating source code by using simple methods that merely test the inputs and outputs of the compiled programs but fail to use more comprehensive methods that would enable the evaluation module to produce a more accurate grade for the student's solution. Our goal is to overcome these limitations and extend the functionality of typical evaluation mechanisms and thus provide an accurate evaluation of the entire program. We do so by introducing the use of software engineering metrics while comparing the student's solution with the teacher's solution. This data, combined with the input/output testing data, provides a complete view of the quality of the student's solution.

### 2 ACode System

The ACode system's goal is to provide a way for students to autonomously practice their programming skills and solve problems and assignments given by their teachers and have an automatic and immediate feedback on their performance. Through the web-based system, students submit their solution for the given problem, which is then analyzed in a four-stage process (see Fig. 1):

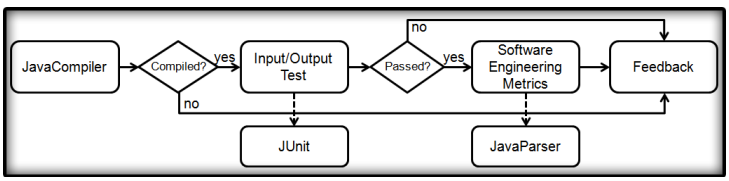



**Fig. 1.** ACode system's flowchart

1. The source code sent by the student is first compiled by the system to make sure that no compilation errors exist. If errors exist, then the system returns a final grade of 0 (zero). The student can then re-submit a solution.
2. If the code compiles without any errors, the second stage will run a set of JUnit tests [4] (allows a programmer to execute automatic tests on java code) provided by the teaching staff. If all tests fail, then the system returns a final grade of 0 (zero), otherwise, continues.
3. After the testing stage, the system will retrieve important data from the compiled source code (through the JavaParser framework [5]) and from the teacher's solution and compare both sets of data using simple software engineering metrics (see section 2.1).
4. Finally, the system gathers all information from previous stages and using a set of formulas (see section 2.1), calculates the final grade of the student's solution. This final grade, along with a detailed report of the evaluation is presented to the student so that he or she can assess his or her performance.

## 2.1 Evaluation Process

The second stage, as described in the previous section, performs a set of tests through the ACode system so as to obtain the number of successful tests in the student's solution. Expression (1) depicts the way the system assigns a grade to the test-stage of the evaluation process.

$$tests = \frac{correct\_tests \times 20}{total\_tests} \quad (1)$$

Afterwards, the system extracts specific metric-based data from both the student's solution and the teacher's solution. The number of lines of each method, the number of methods, the cyclomatic complexity [6] (measurement and control of the number of paths in a program) and the number of global variables in the class are some of the software engineering metrics that are used in the ACode's automatic evaluation module for Java source code. We have chosen this set of metrics because they are adequate to evaluate simple algorithms, which are often the ones used by first year students. The system will compare both sets of metric-based data (the student's and the teacher's), as depicted in expression (2), to calculate the grade for the metrics evaluation stage (stored in the individual_metric variable).

$$individual\_\text{metric} = 20 - \mathrm{e}^{\left(\frac{\text{metric}_{\text{student}} - \text{metric}_{\text{professor}}}{\text{metric}_{\text{professor}}} - 1\right)} \quad (2)$$

The teacher will provide a percentage for each of the evaluation elements, which marks the relevance of that specific element within the evaluation. Table 1 describes all of the percentages used in this metrics-based evaluation.

**Table 1.** Description of the percentage variables

| Variable | Description |
|---|---|
| $P_{class}$ | Percentage assigned to the metrics of the class |
| $P_{method}$ | Percentage assigned to the metrics of a method |
| $P_{project}$ | Percentage assigned to the metrics of the project |
| $pcc_{class}$ | Percentage assigned to the cyclomatic complexity of the class |
| $pln_{class}$ | Percentage assigned to the number of lines in the class |
| $pmn_{class}$ | Percentage assigned to the number of methods in the class |
| $pan_{class}$ | Percentage assigned to the number of attributes in the class |
| $pcc_{method}$ | Percentage assigned to the cyclomatic complexity of a method |

| | |
|---|---|
| $pln_{method}$ | Percentage assigned to the number of lines in a method |
| $pcn_{project}$ | Percentage assigned to the number of classes in the project |
| $pmn_{project}$ | Percentage assigned to the number of methods in the project |
| $pcc_{project}$ | Percentage assigned to the cyclomatic complexity of the project |
| $pln_{project}$ | Percentage assigned to the number of lines in the project |
| $pan_{project}$ | Percentage assigned to the number of attributes in the project |

$$metrics\_class = pcc_{class} \times cyclomatic\_complexity + pln_{class} \times lines\_number + pmn_{class} \times methods\_number + pan_{class} \times attributes\_number \quad (3)$$

$$metrcis\_method = pcc_{method} \times cyclomatic\_complexity + pln_{method} \times lines\_number \quad (4)$$

$$metrcis\_project = pcn_{project} \times classes\_number + pmn_{project} \times methods\_number + pcc_{project} \times cyclomatic\_complexity + pln_{project} \times lines\_number + pan_{project} \times attributes\_number \quad (5)$$

Expression (6) presents the formula used to calculate the grade associated with the metrics evaluation stage (stored in variable metrics) that uses the percentages described in Table 1.

$$metrics = P_{class} \times metrics\_class + P_{method} \times metrics\_method + P_{project} \times metrics\_project \quad (6)$$

Finally, all this evaluation data is combined in order to determine the final grade assigned to the student's solution. Expression (7) represents the formula used to calculate the final grade:

$$R = \begin{cases} 0 & , compilation = 0 \\ 0 & , compilation = 1 \; and \; tests = 0 \\ P_{r1} \times tests + P_{r2} \times metrics & , compilation = 1 \; and \; tests > 0 \end{cases} \quad (7)$$

In this expression, $P_{r1}$ represents the percentage associated with the importance of the testing stage and $P_{r2}$ represents the percentage associated with the importance of the metrics evaluation stage. The teacher provides both percentages as he (or she) sees fit. In the end, the student will get a complete report stating his or her final grade and the evaluation information that shows the tests that were successful and the ones that were unsuccessful.

## Acknowledgements

This work was partly supported by Fundation of Science and Technology (FCT) under the grants PTDC/EEACRO/104658/2008, SFRH/BD/76438/2011 and PEst-OE/EEI/LA0008/2011.

## References

1. Leal, José Paulo e Silva, Fernando: Mooshak: a Web-based multi-site programming contest system. In Software: Practice and Experience, Wiley Interscience, vol. 33, pp. 567-581, ISSN: 0038-0644 (2003)
2. Ashoo, Samir E, Boudreau, Troy e Lane, Douglas A.: PC2 Documentation - Version 9.1 What's New Document. Link: http://www.ecs.csus.edu/pc2/doc/v9/WhatsNewInV9.1.pdf (Accessed in 26 January 2012)
3. Revilla, Miguel A., Manzoor, Shahriar e Liu, Rujia: Competitive Learning in Informatics: The UVa Online Judge Experience. In Olympiads in Informatics, Institute of Mathematics and Informatics, Lithuania, vol. 2, pp. 131-148, ISSN: 1822-7732 (2008)

4. Beck, Kent; Gamma, Erich: JUnit. Link: http://www.junit.org/ (Accessed in 22 February 2012)
5. Project Google: Java 1.5 Parser and AST. Link: http://code.google.com/p/javaparser/ (Accessed in 11 January 2012)
6. McCabe, Thomas J.: A Complexity Measure. In IEEE Transactions on Software Engineering, Institute of Electrical and Electronics Engineers, New York, vol. SE-2, NO. 4, pp. 308-320 (1976)