FRANCIELE BEAL

# UM MÉTODO PARA A CONSTRUÇÃO DO PERFIL DINÂMICO DO DESENVOLVEDOR NO DESENVOLVIMENTO COLABORATIVO DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção de título de Mestre em Informática.

CURITIBA

2014

FRANCIELE BEAL

# UM MÉTODO PARA A CONSTRUÇÃO DO PERFIL DINÂMICO DO DESENVOLVEDOR NO DESENVOLVIMENTO COLABORATIVO DE SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para a obtenção de título de Mestre em Informática.

Área de Concentração: *Ciência da Computação*
Linha de Pesquisa: *Descoberta do Conhecimento e Aprendizagem de Máquina*

Orientador: Prof. Dr. Emerson Cabrera Paraiso



CURITIBA

2014

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

B366m 2014

Beal, Franciele
Um método para a construção do perfil dinâmico do desenvolvedor no desenvolvimento colaborativo de software / Franciele Beal ; orientador, Emerson Cabrera Paraiso. – 2014.
59, [43] f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2014
Inclui bibliografias

1. Informática. 2. Software - Desenvolvimento. 3. Software – Manutenção. 4. Software – Controle de qualidade. 5. Programação (Computadores). 6. Engenharia de software. I Paraiso, Emerson Cabrera. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 004.068

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

PUCPR
GRUPO MARISTA

## ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

### DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 02/2014

Aos 30 dias do mês de Abril de 2014 realizou-se a sessão pública de Defesa da Dissertação **"Um Método para a Construção do Perfil Dinâmico do Desenvolvedor no Desenvolvimento Colaborativo de Software"** apresentado pela aluna **Franciele Beal**, como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

| | | |
|---|---|---|
| Prof. Dr. Emerson Cabrera Paraiso<br>PUCPR (Orientador) | (assinatura) | APROV.<br>(Aprov/Reprov) |
| Prof. Dr. Julio Cesar Nievola<br>PUCPR | (assinatura) | APROVADO<br>(Aprov/Reprov) |
| Profª. Drª. Deborah Ribeiro Carvalho<br>PUCPR/PPGTS | (assinatura) | Aprov<br>(Aprov/Reprov) |
| Profª. Drª. Carmem Hara<br>UFPR | (assinatura) | APROV<br>(Aprov/Reprov) |

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado Aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Mauro Sério Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática

Rua Imaculada Conceição, 1155 Prado Velho CEP 80215-901 Curitiba Paraná Brasil
Tel.: (41) 3271 1689 Fax.: (41) 3271 2121 www.ppgia.pucpr.br

*Dedico essa dissertação ao meu marido* Celso *e aos meus pais* Honorino *e* Julcemir.

*Livros não mudam o mundo,*
*quem muda o mundo são as pessoas.*
*Os livros só mudam as pessoas.*
*(Mario Quintana)*

# Agradecimentos

Gostaria de agradecer primeiramente a Deus por iluminar meus passos durante esta longa caminhada. Por ter me dado força e saúde para superar os desafios e as dificuldades.

Agradeço ao Professor Emerson pela confiança, dedicação, ensinamentos e paciência nas orientações.

Aos meus colegas e amigos Andréia, Edenilson, Irapuru, Mariza, Cheila, Rosana, Denise, Priscila, Gregory, Lohann e Daniel. Obrigada pelas contribuições e pelo apoio constantes, mas principalmente pela amizade.

Ao meu esposo Celso pelo incentivo, carinho e compreensão. Eu não teria conseguido chegar onde cheguei sem seu apoio.

À minha irmã Alexandra e minha colega Andréia por terem me acolhido generosamente em suas residências durante o período do mestrado.

À Fundação Araucária, CAPES e PUC-PR pela bolsa de estudos.

Enfim, a todos que de alguma forma contribuíram para a conclusão deste mestrado.

# Sumário


**CAPÍTULO 1** ..... **1**

**INTRODUÇÃO** ..... **1**

Motivação ..... 3

Objetivos ..... 3

Hipóteses de Trabalho ..... 4

Contribuição Científica ..... 4

Escopo ..... 4

Organização do Documento ..... 4

**CAPÍTULO 2** ..... **6**

**FUNDAMENTAÇÃO TEÓRICA** ..... **6**

2.1. A Modelagem do Usuário ..... 6

2.1.1. Composição do modelo ..... 7

2.1.2. Representação do modelo ..... 7

2.1.3. Aquisição do Modelo ..... 9

2.1.4. Aprendizado do Modelo ..... 10

2.1.5. Manutenção do Modelo ..... 10

2.2. Coleta de Dados para Aquisição do Modelo ..... 11

2.2.1 Framework HACKYSTAT ..... 11

2.2.2 Plugin METRICS ..... 13

2.3. Aprendizado do Modelo: Classificação ..... 14

2.3.1. Conjunto de Treinamento ..... 15

2.3.2. Aprender o Modelo ..... 16

2.3.3. Modelo ..... 18

2.3.3. Conjunto de Teste ..... 20

2.4. Métricas de Código-fonte Orientado a Objetos ..... 21

2.5. Considerações Finais ..... 25

**CAPÍTULO 3** ..... **26**

**TRABALHOS RELACIONADOS** ..... **26**

3.1. Modelagem de Usuário na Engenharia de Software ..... 26

3.2. Sensores para Coleta de Dados sobre os Desenvolvedores de Software ..... 28

3.3. Ferramentas para a Obtenção de Métricas de Código-Fonte ..... 28

3.4. Aprendizagem de Máquina na Engenharia de Software ..... 28

3.5. Considerações Finais ..... 30

CAPÍTULO 4 ..... 31
MÉTODO PROPOSTO ..... 31
4.1. Aquisição dos Perfis ..... 35
4.2. Modelo de Classificação ..... 40
4.3. Utilização do Modelo ..... 41
4.4. Manutenção do Modelo ..... 43
4.5. Considerações Finais ..... 43
CAPÍTULO 5 ..... 44
PROTÓTIPO COMPUTACIONAL: DEVMODEL ..... 44
5.1. Considerações Finais ..... 47
CAPÍTULO 6 ..... 48
EXPERIMENTOS REALIZADOS E ANÁLISE DOS RESULTADOS ..... 48
6.1. Formação da Base Histórica ..... 48
6.2. Formação do Conjunto de Dados para Treinamento dos Classificadores ..... 49
6.3. Treinamento dos Classificadores e Avaliação do Desempenho ..... 50
6.4. Avaliação da Viabilidade da Aplicação de Novos Cenários ..... 52
6.5. Considerações Finais ..... 55
CONCLUSÃO ..... 56
Trabalhos Futuros ..... 57
Dificuldades Encontradas Durante a Pesquisa ..... 58
REFERÊNCIAS BIBLIOGRÁFICAS ..... 60
APÊNDICE A ..... 67
EXEMPLOS DE EVENTOS COLETADOS PELO SENSOR HACKYSTAT ..... 67
APÊNDICE B ..... 70
EXEMPLOS DE CÁLCULOS DAS MÉTRICAS DE COMPLEXIDADE ..... 70
A.1. Exemplo de Cálculo da CCM: ..... 70
A.2. Exemplo de Cálculo da WMC: ..... 71
A.3. Exemplo de Cálculo da LCOM*: ..... 71
A.4. Exemplo de Cálculo da NBD: ..... 74
APÊNDICE C ..... 75
GRUPOS DE ERROS ..... 75
APÊNDICE D ..... 76
CENÁRIO : MYBMI ..... 76
APÊNDICE E ..... 81
DOCUMENTOS COMITÊ DE ÉTICA DA PUCPR ..... 81

# Lista de Figuras

**Figura 1**- Exemplo de Modelo de Usuário utilizando Árvores de Decisão (Adaptado de: Barth, 2010, v.6, p.62 [BAR10])..... 9
**Figura 2** - Processo de coleta de dados utilizando o HACKYSTAT SENSOR ECLIPSE (Fonte: Autor)..... 11
**Figura 3**- Guia de visualização das métricas calculadas pelo METRICS (Fonte: Autor)..... 13
**Figura 4**- Abordagem geral para a tarefa de classificação (Fonte: Adaptado de [TAN09]). ..... 15
**Figura 5** - Matriz de Confusão Clássica para 2 classes (Fonte: adaptado de [TAN09])..... 19
**Figura 6**- Matriz de Confusão com definição de VP, FP, FN e VN (Fonte: adaptado de [TAN09]..20
**Figura 7**- Gráfico Abstração X Instabilidade (Fonte: adaptado de [MAR94]). ..... 24
**Figura 8**- Etapas do Método Proposto (Fonte: Autor). ..... 35
**Figura 9**- Processo de Coleta de Dados por Sensores (Fonte: Autor)..... 37
**Figura 10**- Rotulação dos Dados por um Especialista (Fonte: Autor). ..... 38
**Figura 11**- Processo de Treinamento e Avaliação de Desempenho dos Modelos (Fonte: Autor). ....41
**Figura 12** - Fluxo da Utilização do Perfil Dinâmico (Fonte: Autor)..... 42
**Figura 13**- Fluxo do Protótipo para a Aquisição de Perfis (Fonte: Autor)..... 46
**Figura 14**- Fluxo do Protótipo para o Modelo de Classificação (Fonte: Autor). ..... 46
**Figura 15**- Exemplo de Aplicação da Métrica CCM (Fonte: Autor). ..... 70
**Figura 16**- Exemplo de cálculo da WMC utilizando CCM (Fonte: Autor). ..... 71
**Figura 17**- Exemplo de classes com o mesmo valor de LCOM (Adaptado de [HEN96] (p. 145))..73
**Figura 18** - Exemplo de método com NBD igual a 3 (Fonte: Autor). ..... 74
**Figura 19**- Protótipos de Telas para Lançamento de Dados de Usuário (Fonte: Autor). ..... 78
**Figura 20**- Telas de Visualização do IMC (Fonte: Autor)..... 78
**Figura 21**- Tela de visualização do GEB (Fonte: Autor). ..... 79
**Figura 22**- Tela de lançamento de calorias (Fonte: Autor)..... 79
**Figura 23**- Tela de informações de Acompanhamento (Fonte: Autor)..... 79

# Lista de Tabelas

**Tabela 1**- Campos Obrigatórios de uma Instância SensorData ([HAC12b]). ..... 12
**Tabela 2** - Principais tipo de dados coletados (Fonte: Adaptado de [USE13]). ..... 12
**Tabela 3**- Definições e Fórmulas de Métricas de Complexidade (Fonte: Autor). ..... 21
**Tabela 4**- Definições e Fórmulas de Métricas de Herança (Fonte: Autor). ..... 22
**Tabela 5**- Definições e Fórmulas de Métricas de Tamanho (Fonte: Autor). ..... 23
**Tabela 6**- Definições e Fórmulas de Métricas de Acoplamento (Fonte: Autor). ..... 23
**Tabela 7** - Trabalhos Relacionados mais importantes(Fonte: Autor). ..... 30
**Tabela 8**- Lista de Características do Desenvolvedor (Fonte: Autor). ..... 32
**Tabela 9**- Campos de uma Instância de Dados da Base Histórica (Fonte: Autor). ..... 36
**Tabela 10**- Matriz de observações(Fonte: adaptado de [FON07]). ..... 39
**Tabela 11**- Valores de concordância Kappa propostos por [FLE81] (Fonte: adaptado de [FON07]). ..... 40
**Tabela 12**- Distribuição dos Registros por Classe - Conjunto de Treinamento (Fonte: Autor). ..... 50
**Tabela 13**- Acurácia dos Modelos (Fonte: Autor). ..... 51
**Tabela 14**- F-Measure dos Modelos (Fonte: Autor). ..... 51
**Tabela 15**- Matrizes de Confusão - J48, MLP e IBK (Fonte: Autor). ..... 52
**Tabela 16**- Acurácia Subconjunto Parte 1 (Fonte: Autor). ..... 53
**Tabela 17**- Acurácia Subconjunto Parte 2 (Fonte: Autor). ..... 53
**Tabela 18**- Acurácia Subconjunto Parte 3 (Fonte: Autor). ..... 53
**Tabela 19**- F-Measure Subconjunto Parte 1 (Fonte: Autor). ..... 53
**Tabela 20**- F-Measure Subconjunto Parte 2 (Fonte: Autor). ..... 54
**Tabela 21**- F-Measure Subconjunto Parte 3 (Fonte: Autor). ..... 54
**Tabela 22**- Exemplos de Eventos Coletados pelo HACKYSTAT SENSOR ECLIPSE (Adaptado de [USE13]). ..... 67
**Tabela 23**- Grupos de Erros (Fonte: Autor). ..... 75
**Tabela 24**- Histórias de Usuário (Fonte: Autor). ..... 77

# Lista de Listagem

**Listagem 1-** Um Exemplo Simples de Modelo de Usuário (Fonte: Autor)..... 8
**Listagem 2-** Representação de um Perfil em Formato XML (Fonte: Autor). ..... 34
**Listagem 3-** Intersecção de pares de métodos (Fonte: Autor)..... 72

# Lista de Gráficos

**Gráfico 1-** Resultado da Acurácia do J48 - Partes 1, 2 e 3 (Fonte: Autor)..... 54
**Gráfico 2-** Resultado da F-Measure do J48 - Partes 1, 2 e 3 (Fonte: Autor)..... 55

# Lista de Abreviaturas

| | |
|---|---|
| A | *Abstractness* |
| ACM | *Association for Computing Machinery* |
| AM | *Machine Learning* |
| Ca | *Afferent Coupling* |
| CCM | Complexidade Ciblomática de *McCabe* |
| Ce | *Efferent Coupling* |
| CK | Chimdaber and Kemerer |
| CSCW | *Computer Supported Cooperative Work* |
| CSCW-SD | *Computer Supported Cooperative Work in Software Development* |
| DIT | *Depth of Inheritance Tree* |
| Dn | *Normalized Distance* |
| EPL | *Eclipse Public License* |
| I | *Instability* |
| IDE | *Integrated Development Environment* |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| k-NN | *k-Nearest Neighbor* |
| LCOM | *Lack of Cohesion in Methods* |
| MLOC | Linhas de Código por Método |
| MAS | *Multi-Agent System* |
| MLP | *Multilayer Perceptron* |
| MODUS-SD | *User Modelling in Cooperative Software Development* |
| NAC | *Number of Attributes per Class* |
| NBD | *Nested Block Depth* |
| NOC | *Number of Children* |
| NOI | *Number of Interfaces* |
| NORM | *Number of Overridden Methods* |
| NOP | *Number of Packages* |
| NSF | *Number of Static Attributes* |

| | |
|---|---|
| NSM | *Number of Static Methods* |
| PA | *Personal Agent* |
| PAR | *Number of Parameters* |
| PROG.II | Turma de Programação II |
| PROG.IV | Turma de Programação IV |
| PROG.VI | Turma de Programação VI |
| PROM | PRO *Metrics* |
| PUC-PR | Pontifícia Universidade Católica do Paraná |
| RNA | Redes Neurais Artificiais |
| SIX | *Specialization Index* |
| SOM | *Self-Organizing Map* |
| SVM | *Support Vector Machines* |
| WMC | *Weighted Methods per Class* |
| XML | *eXtensible Markup Language* |
| XP | *Extreme Programming* |

# Resumo

O trabalho realizado por desenvolvedores de software (i.e. a programação), interfere diretamente na complexidade e na manutenibilidade do software. A experiência e a capacidade de cada desenvolvedor, por sua vez, podem interferir na qualidade do código-fonte produzido. A complexidade está relacionada com estruturas de códigos-fonte difíceis de serem mantidas e testadas. A manutenibilidade está relacionada com a facilidade de compreender, modificar, adicionar novas funcionalidades, corrigir falhas e manter um código-fonte. Os desenvolvedores possuem diferentes características que determinam a qualidade dos códigos-fonte construídos por eles. Os modelos de usuários podem ser utilizados para representar as características dos desenvolvedores. Estas características podem ser estáticas ou dinâmicas. As características estáticas normalmente não se alteram, ou se alteram raramente. As dinâmicas se modificam com o passar do tempo e com o ganho de experiência. As características dinâmicas podem ser obtidas a partir do código-fonte. Nesta pesquisa é proposto um método para a construção do perfil dinâmico do participante desenvolvedor no desenvolvimento colaborativo de software. O método é capaz também de classificar os desenvolvedores em diferentes perfis. Técnicas para criação, representação, aquisição, aprendizado e manutenção do perfil são apresentadas. Uma vez tendo o perfil dinâmico, os desenvolvedores podem ser classificados em diferentes tipos ou classes. Esta classificação pode, por exemplo, ser utilizada para orientar o processo de formação de equipes ou duplas de programação e orientar na distribuição de atividades. Pode também orientar o direcionamento de suporte e treinamento aos desenvolvedores para que eles possam melhorar e aperfeiçoar suas características. Para a avaliação do método, informações do código-fonte produzidos pelos desenvolvedores foram coletados, formando uma base com dados históricos. Um grupo de classificadores candidatos (RNA, *k*-NN, SVM, Naive Bayes e Árvore de Decisão) foi treinado com os dados desta base. O classificador com melhor desempenho foi uma Árvore de Decisão com uma Acurácia de 90,96%, seguida pelos classificadores MLP e *k*-NN, com Acurácias 86,44% e 85,08%, respectivamente. Os classificadores SVM e Naive Bayes tiveram desempenhos inferiores.

**Palavras-Chave**: Modelagem do Usuário, Perfil do Desenvolvedor, Desenvolvimento de Software, Classificação de Desenvolvedores.

# Abstract

The experience and ability of developers interferes in the complexity and maintainability of the source code produced by them. The complexity is related to complex source code structures, difficult to maintain and test. The maintainability is linked to easy understanding, modifying, adding new features and fixing bugs. The developers have different features that determine the quality of the source code. The user models can be used to represent the features of the developers, which may be static or dynamic. The static features are unchanged over time. The dynamic part of the model changes with the course of time. The dynamic features can be determined from the source code. In this research is described a method capable of building the dynamic profile of the developer participant of a collaborative software development. This method can classify developers in different profiles. Techniques to structure, to represent, to acquire, to learn and to maintain the model are presented. Developers can be classified into different types or classes with the dynamic profile. This classification can, for example, be used to guide the process of forming teams or pairs of programming and distribution activities. It can also guide the targeting of support and training to developers so they can improve and enhance its features. For evaluation of the method, information from the source code produced by developers were collected, forming a base with historical data. A group of candidates (ANN, *k*-NN, SVM, Naive Bayes and Decision Tree) classifier was trained with this data base. The classifier performance was better with the Decision Tree with an accuracy of 90.96%, followed by the classifiers RNA and *k*-NN with accuracies 86.44% and 85.08%, respectively. The SVM and Naive Bayes classifier had underperformed.

**Keywords:** User Modeling, Developer Profile, Software Development, Developer Classifying.

# Capítulo 1

## Introdução

Em ambientes de desenvolvimento que seguem os princípios do trabalho colaborativo apoiado por computador (CSCW) [GRU94] os participantes constroem os artefatos, como código-fonte, diagramas, documentos de requisitos, colaborativamente.

Os artefatos produzidos devem ser avaliados para garantir a qualidade do software. Entre as estratégias de avaliação, as métricas de software pode ser utilizadas para quantificar propriedades de qualidade. Para os artefatos produzidos por participantes Programadores (ou Desenvolvedores), as métricas podem ser utilizadas para avaliar, por exemplo, a complexidade e manutenibilidade. Para artefatos produzidos por participantes Arquitetos, as métricas aplicadas procuram cobrir, por exemplo, a modularidade e a reusabilidade.

A complexidade está relacionada com estruturas de códigos-fonte complicadas, difíceis de serem mantidas e testadas. Ela pode aumentar o tempo e os custos da manutenção, bem como dos testes [PRE10]. A manutenibilidade está relacionada com a facilidade de um código-fonte ser modificado para receber novas funcionalidades, corrigir falhas, ou melhorar o desempenho. Portanto, um código-fonte com boa manutenibilidade é fácil de ser compreendido, modificado e testado [IEE90]. A modularidade está relacionada com a capacidade de decompor um componente de software em outros módulos de forma que a mudança em um dos módulos cause impacto mínimo nos demais [IEE90]. Já a reusabilidade está relacionada com a capacidade de reutilizar componentes de softwares existentes em outros sistemas [IEE90].

Assim, como os artefatos são produzidos por participantes do processo de desenvolvimento, pode-se concluir que as propriedades de qualidade dos artefatos estão diretamente dependentes dos participantes que os constroem. Elas são determinadas pela capacidade do participante, ou seja, pela respectiva habilidade (ou experiência) em construir os artefatos [YU10].

Os participantes possuem diferentes características que determinam a qualidade do artefato produzido por eles. Tomando como exemplo os desenvolvedores, são exemplos de características:

construir classes com forte ou fraco acoplamento, classes com complexidade de estrutura alta, classes com complexidade moderada, classes com muitas subclasses, cometer ou não cometer erros de sintaxe básicos, não utilizar o depurador de código (*Debug*), dentre outros.

Desta forma, é importante que as equipes de desenvolvimento de software conheçam as características dos seus participantes para monitorar as propriedades de qualidade do software construído, para que os gestores possam organizar as equipes, direcionar atividades mais complexas para os participantes mais experientes e, principalmente, para fornecer suporte para ajudar os participantes a desenvolver e melhorar as suas características.

As características dos participantes podem ser classificadas em estáticas e dinâmicas. As características estáticas não se alteram. São exemplos de características estáticas: e-mail, competências, habilidades em tecnologias, tempo de serviço, metas e preferências. As dinâmicas se modificam com o passar do tempo e com o ganho de experiência. Um participante pode começar com pouca experiência em uma categoria, mas, com a prática diária e a troca de informações com os mais experientes, pode adquirir características de participante dito Sênior.

Existem métricas de qualidade de software [HON09] que são utilizadas para avaliar a qualidade de artefatos e podem ser utilizadas para caracterizar participantes.

Para o caso dos participantes desenvolvedores, existem métricas para avaliar a complexidade da estrutura como a Complexidade Ciclomática de McCabe (CCM) [MCC76] e métricas para avaliar a código-fonte orientado a objetos como as de Chimdaber e Kemerer [CHI91], Henderson-Sellers [HEN96], Lorenz e Kidd (HENDERSON-SELLERS 1996 *apud* LORENZ e KIDD, 1994) [HEN96] e Robert C. Martin [MAR94].

As características de um participante podem ser representadas em um perfil, chamado genericamente de modelo do usuário. Os modelos de usuários são abordagens que relacionam uma grande variedade de conhecimentos sobre usuários [RIC83]. Eles podem ser utilizados para representar as características que os participantes do processo de desenvolvimento de software vão adquirindo e modificando ao longo do tempo. Neste caso são chamados de modelos ou perfis dinâmicos [ZHO11].

Para manter o perfil dinâmico atualizado pode-se acompanhar os participantes em seus trabalhos diários e coletar automaticamente (em tempo real) as métricas de qualidade para o perfil. Para isso são necessárias ferramentas de coleta de dados ligadas às ferramentas utilizadas na construção dos artefatos (IDEs, Ferramentas para Modelagem de Sistemas, Gerenciamento de Requisitos e de Projetos, p. ex.). Nem todas ferramentas, porém, são facilmente acopláveis a sensores de coletas de dados. As ferramentas utilizadas pelos desenvolvedores, como um IDE, possuem esse tipo de sensor [JOH03].

**Motivação**

O perfil dinâmico dos desenvolvedores pode ser uma ferramenta útil para as equipes de desenvolvimento colaborativo de software preocupadas com a complexidade e a manutenibilidade do software construído por seus desenvolvedores. Desta forma, é importante conhecer as características dos desenvolvedores para acompanhar a complexidade e a manutenibilidade do software que está sendo desenvolvido.

Os gestores de projetos podem utilizar o perfil dinâmico dos desenvolvedores para organizar as suas equipes e direcionar atividades mais complexas para os desenvolvedores mais experientes (ou eficientes em função de critérios que podem ser estabelecidos).

Na programação em pares, da metodologia ágil Extreme Programming (XP) [PAD03], por exemplo, dois programadores colaboram na construção de código-fonte em um único computador com objetivo de produzir sistemas com menos defeitos e com maior qualidade. Enquanto um escreve o código-fonte, o outro revisa em busca de problemas [MUJ05]. Aiken [AIK04] relata que a programação em par promove a aprendizagem reforçada entre as duplas, ou seja, os parceiros aprendem um com o outro por meio de compartilhamento de conhecimentos. Desta forma, é sugerido que as duplas sejam "rotacionadas" e que sejam formadas por integrantes com experiências diferentes. Para que isso aconteça, é necessário conhecer as características de cada desenvolvedor para poder formar uma dupla colaborativa produtiva. Para casos como este, o modelo pode ser utilizado para classificar os desenvolvedores e orientar na formação das duplas.

O benefício mais importante que o perfil dinâmico do desenvolvedor pode proporcionar é a classificação do desenvolvedor de acordo com suas características. A classificação pode ser utilizada para orientar treinamentos e suporte para ajudar os desenvolvedores a aperfeiçoarem as suas características.

Desta forma, o presente trabalho apresenta um método para a construção do perfil dinâmico do desenvolvedor com base em características identificadas no código-fonte.

**Objetivos**

O objetivo principal deste trabalho é definir, implementar e avaliar um método para construir o perfil dinâmico do participante desenvolvedor em um projeto de desenvolvimento colaborativo de software, a partir de características obtidas do código-fonte.

Objetivos específicos:

- Identificar as características do código-fonte que podem ser utilizadas para caracterizar o

desenvolvedor e projetar o perfil, definindo suas características e sua forma de representação;

- Construir uma base com dados históricos sobre o trabalho de codificação do desenvolvedor, por meio de coleta de dados automática e não invasiva;
- Desenvolver um algoritmo capaz de classificar o desenvolvedor em um dos diferentes tipos de desenvolvedores previamente definidos de acordo com as características definidas no perfil do desenvolvedor.

### Hipóteses de Trabalho

As hipóteses deste trabalho são:

i. *"É possível construir um perfil dinâmico do desenvolvedor com dados obtidos do código-fonte produzido por ele?"*
ii. *"Com o perfil dinâmico do desenvolvedor é possível classificar os desenvolvedores em diferentes perfis?"*

### Contribuição Científica

A principal contribuição científica deste trabalho é disponibilizar um método para a construção do perfil dinâmico que pode ser utilizado para fazer a classificação dos desenvolvedores em diferentes perfis. Outra contribuição é a lista de características do desenvolvedor utilizada para compor o perfil dinâmico.

Uma outra contribuição, não menos importante, é uma base com dados coletados durante o trabalho de codificação dos desenvolvedores que será disponibilizada e poderá ser utilizada por outros pesquisadores.

### Escopo

O escopo deste trabalho limita-se à construção do perfil dinâmico dos desenvolvedores, codificando em Java, em equipes de desenvolvimento colaborativo de software, em projetos cujo código é desenvolvido utilizando-se o paradigma orientado a objetos.

### Organização do Documento

Este trabalho está organizado em 6 capítulos. Este é o primeiro e contém as considerações

iniciais, a motivação, objetivos, hipóteses de trabalho, contribuições científicas e escopo.

O Capítulo 2 apresenta a fundamentação teórica que aborda 4 tópicos principais necessários para o entendimento deste trabalho. São eles: A Modelagem do Usuário (2.1), Coleta de Dados para a Aquisição do Modelo (2.2), Aprendizado do Modelo: Classificação (2.3) e Métricas de Código-fonte Orientado a Objetos (2.4).

O Capítulos 3 apresenta alguns trabalhos relacionados organizados nas seguintes seções: Modelagem de Usuário na Engenharia de Software (3.1), Sensores para Coleta de Dados sobre os Desenvolvedores de Software (3.2), Ferramentas para a Obtenção de Métricas de Código-Fonte (3.3) e Aprendizagem de Máquina na Engenharia de Software (3.4).

O Capítulo 4 descreve o método proposto para a construção do perfil dinâmico do desenvolvedor. O método é detalhado em 4 tópicos: Aquisição dos Perfis (4.1), Modelo de Classificação (4.2), Utilização do Modelo (4.3) e Manutenção do Modelo (4.4).

O Capítulo 5 apresenta o protótipo computacional devMODEL utilizado para avaliar o método de construção do perfil dinâmico do desenvolvedor.

O Capítulo 6 apresenta os experimentos realizados e os resultados obtidos.

No final do documento são apresentadas as conclusões, trabalhos futuros, dificuldades encontradas durante a pesquisa e referências bibliográficas consultadas.

# Capítulo 2

# Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos necessários para o entendimento da proposta do perfil dinâmico e posterior classificação dos desenvolvedores apresentada sob a forma de um método no Capítulo 4.

Inicialmente é apresentada a modelagem de usuário. Entender a modelagem de usuários é fundamental para compreender o perfil dinâmico proposto.

Na seção 2.2 são apresentados o Framework HACKYSTAT e o Plugin METRICS. HACKYSTAT é uma ferramenta para coletar dados sobre o desenvolvimento de software e o METRICS é um plugin para coletar métricas de código-fonte. Conhecer estas ferramentas é importante para entender a forma como os dados para o modelo são coletados.

Na seção 2.3 é apresentada a classificação, uma tarefa de Aprendizagem de Máquina [MIT97]. Para aprender o perfil dinâmico do desenvolvedor podem ser aplicados algoritmos de Aprendizagem de Máquina para classificação, por isso, é fundamental conhecer a tarefa e suas etapas.

Na seção 2.4 são apresentadas métricas de código-fonte orientado a objetos voltadas à complexidade, herança, tamanho e acoplamento. As métricas são importantes porque elas serão utilizadas para indicar a qualidade do código-fonte construído por um desenvolvedor e para representar as respectivas no perfil dinâmico.

## 2.1. A Modelagem do Usuário

Um modelo do usuário é uma representação estruturada de características e dados sobre um usuário. Estes dados podem ser estáticos (nome, sexo, e-mail, áreas de interesses, competências, habilidades, objetivos, metas, preferências) ou sobre o comportamento e a interação do usuário com o sistema, ditos dinâmicos [WAN12]. Para Rich [RIC83] pode relacionar uma grande variedade de conhecimentos sobre usuários. De modo geral, pode ser interpretada como uma representação de

conhecimentos e preferências de usuários que eles vão adquirido e modificando ao longo do tempo [BIS11].

Existem diferentes padrões de design para modelagem de usuário. O modelo (ou perfil) estático é composto por características estáticas que não se alteram ao longo do tempo. As modificações neste modelo são feitas por meio da edição do perfil. E o perfil dinâmico, onde as características do usuário se modificam ao longo do tempo, conforme as interações do usuário com o sistema evoluem [WAN12].

Um exemplo de perfil estático de usuário pode ser um site de uma livraria que identifica as preferências do usuário por determinadas categorias de livros com base em indicação explícita do mesmo. Caso o usuário queira modificar uma das suas preferências, ele necessita editá-la. Um exemplo de perfil dinâmico é o perfil do desenvolvedor proposto neste trabalho. Este perfil é construído a partir de dados colhidos sobre o trabalho de programação do desenvolvedor, sem a necessidade dele editar seu perfil. As modificações são capturadas por sensores de coleta implícita que segue o desenvolvedor durante o seu trabalho diário para atualizar o perfil automaticamente.

O processo de construção de modelos de usuários envolve uma série de etapas. O processo apresentado em [BAR10] é bastante interessante, pois organiza a construção dos modelos em etapas bem definidas. São elas: Composição do Modelo, Representação, Aquisição, Aprendizado e Manutenção do modelo. Em cada uma das etapas são aplicadas técnicas e métodos conforme descrito a seguir.

### 2.1.1. Composição do modelo

O primeiro passo para a criação de um modelo de usuário é a identificação dos campos (características, preferências, interesses, conhecimentos, etc.) que irão compor o modelo. Estes campos são definidos de acordo com a utilização do modelo. Se o modelo de usuário será utilizado para recomendar filmes, o modelo deverá conter dados sobre gêneros, por exemplo.

### 2.1.2. Representação do modelo

Esta etapa consiste na aplicação de técnicas para representação do modelo. O modelo de usuário pode ser representado por diversas formas. Barth em [BAR10] descreve e relaciona exemplos das representações baseadas em dados históricos, vetores e conjunto de regras.

As representações baseadas em dados históricos utilizam dados históricos sobre os usuários para representar o modelo. Estes dados podem ser listas de itens consultados, listas de compras efetuadas e históricos de navegação. Nas representações baseadas em vetores, o modelo é

representado por um vetor formado por características, onde cada característica possui um valor associado que representa a sua importância para o usuário. Os modelos baseados em conjunto de regras, podem ser representados por Árvores de Decisão [WEB02] ou por regras em Prolog [CLO03].

Tabelas de dados ou arquivos em formato XML[1] também podem ser utilizados na representação de modelos de usuários. A representação em formato XML é interessante para os modelos dinâmicos, pois o XML fornece uma estruturação flexível que permite a adição de elementos com facilidade. Por isso, ele pode ser indicado como alternativa para o modelo de usuário dinâmico, no qual novas características podem ser adicionadas ao modelo ao longo do tempo.

A Listagem 1 mostra um exemplo de modelo de usuário representado em formato XML. Trata-se do modelo estático de um usuário apreciador de filmes. O modelo possui elementos sobre gêneros de filmes, onde para cada gênero existe um valor associado que é o valor de interesse para o apreciador. Estes valores podem ser modificados a qualquer momento pelo próprio usuário por meio da edição do seu perfil.

**Listagem 1**- Um Exemplo Simples de Modelo de Usuário (Fonte: Autor)

```
<APRECIADOR_FILME>
<!-- Gêneros de filmes e valores de interesse para o Apreciador-->
        <NOME>JOÃO DA SILVA DO BRASIL</NOME>
        <ACAO>4</ACAO>
        <DRAMA>1</DRAMA>
        <AVENTURA>3</AVENTURA>
        <ROMANCE>2</ROMANCE>
        <TERROR>0</TERROR>
</APRECIADOR_FILME>
```

Analisando o modelo da Listagem 1, para o apreciador João da Silva do Brasil o gênero Ação tem valor maior, portanto, é o gênero de maior interesse para ele. Este modelo poderia ser utilizado por um sistema de recomendação para indicar filmes do gênero Ação para o João da Silva do Brasil.

A Figura 1 mostra um exemplo de modelo de usuário representado em Árvores de Decisão, obtido de Barth (2010, v.6, p.62) [BAR10]. Nesta árvore, o autor destaca a seguinte regra que pode ser observada em um de seus ramos:

---

[1] Maiores informações em: <http://www.w3.org/TR/REC-xml/>

SE dia = sexta
E urgente = sim
E período = manhã
ENTÃO Sim
Esta regra significa: se o dia da semana for sexta-feira e a reunião é considerada urgente e o período proposto é na parte da manhã então o sistema poderá marcar a reunião. (2010, v.6, p.61).

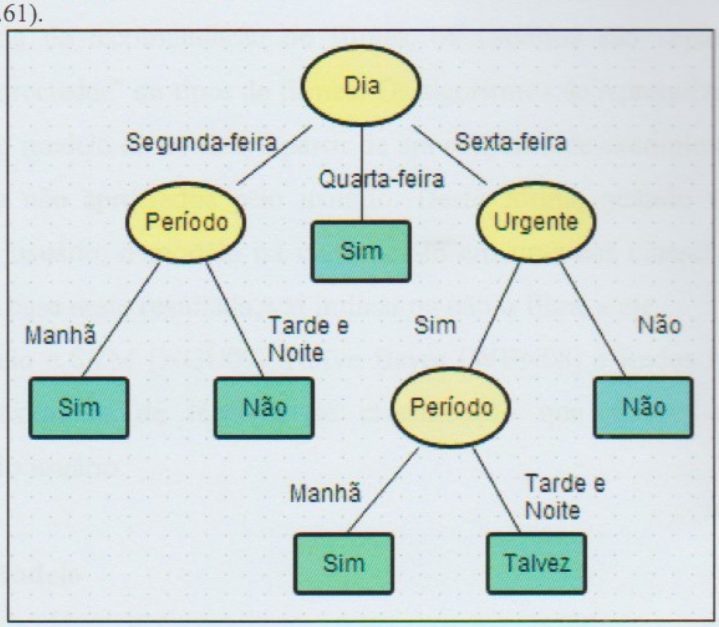


**Figura 1**- Exemplo de Modelo de Usuário utilizando Árvores de Decisão (Adaptado de: Barth, 2010, v.6, p.62 [BAR10])

O modelo de usuário representado na Figura 1 poderia ser utilizado por um sistema de agendamento automático de compromissos, onde, de acordo com as preferências do usuário (dia da semana, período do dia e tipo do compromisso (urgente ou não urgente)) poderia, ou não, realizar um agendamento.

### 2.1.3. Aquisição do Modelo

Esta etapa corresponde à aplicação de métodos para aquisição dos modelos. Os modelos de usuários podem ser adquiridos por meio de métodos implícitos ou explícitos [PAP01]. No método explícito as dados estáticos são fornecidos pelo usuário por meio de formulários ou questionários. No método implícito, os dados dinâmicos são obtidos automaticamente por meio de sensores de coletas não invasiva como o Framework HACKYSTAT e o Plugin METRICS que serão descritos na próxima seção 2.2. Os dados obtidos pelo método implícito formam um conjunto de exemplos que podem ser utilizados para o aprendizado do modelo.

### 2.1.4. Aprendizado do Modelo

Nesta etapa, podem ser utilizadas as tarefas de classificação, da Aprendizagem de Máquina. Elas são utilizadas quando se quer separar em classes (ou perfis) os usuários [SCH02]. Por exemplo, em um sistema de recomendação de filmes, os usuários são separados em classes "Apreciador" ou "Não apreciador" de tipos de filmes. Os algoritmos de Aprendizagem de Máquina aprendem (ou induzem) o modelo do usuário a partir de um conjunto de exemplos com dados sobre os filmes apreciados ou não apreciados pelo usuário. Desta forma, quando um novo filme é disponibilizado para um usuário, o modelo irá classificá-lo em uma das classes "Apreciador" ou "Não Apreciador" e com base neste resultado, vai indicar ou não o filme a ele.

Árvores de Decisão e SVM [NGU09], Naïve Bayes [WEN08] e Redes Neurais Artificiais [CHE91], são alguns exemplos de técnicas de classificação que podem ser utilizadas no aprendizado do modelo do usuário.

### 2.1.5. Manutenção do Modelo

As características, interesses, preferências e habilidades dos usuários se modificam ao longo do tempo, sendo necessário fazer a manutenção do modelo do usuário. A manutenção é realizada para manter o modelo com os dados atualizados.

Para a manutenção do modelo, podem ser utilizados dois tipos de métodos: explícito e o implícito. No método explícito o próprio usuário pode fazer as alterações em seus dados por meio da edição do perfil. Este método não é muito interessante porque exige esforço e disposição do usuário para atualizar o seu perfil. Além disso, ele pode atualizar o perfil com dados incorretos.

No método implícito, é o sistema que faz de forma automática a manutenção do modelo, com base no monitoramento e observações sobre as ações do usuário.

A manutenção dos modelos dinâmicos envolve o treinamento constante dos algoritmos de Aprendizagem de Máquina. Como as características e preferências do usuário se modificam, tanto em nível de valores quanto à adição de novos atributos, os algoritmos precisam ser novamente treinados para manter o modelo atualizado.

Para a construção de modelos dinâmicos de usuários, pode-se utilizar o método implícito tanto para a aquisição do modelo quanto para a manutenção. Desta forma, serão apresentados o Framework HACKYSTAT e o Plugin METRICS, ambas ferramentas para coleta de dados implícita.

## 2.2. Coleta de Dados para Aquisição do Modelo

Nesta seção serão apresentadas duas ferramentas para a coleta de dados implícita e não invasiva. No Capítulo 3, serão apresentados alguns trabalhos que utilizam outros tipos de sensores, diferentes dos que serão apresentados a seguir.

### 2.2.1 Framework HACKYSTAT

O *framework* HACKYSTAT é uma ferramenta de código aberto para coleta e análise de dados referentes ao processo de desenvolvimento de software. Ele monitora, captura e armazena automaticamente os dados por meio de sensores ligados às ferramentas como o IDE ECLIPSE[2].

O processo de coleta se dá por meio de um sensor ligado ao IDE de forma não invasiva. Uma série de eventos gerados durante a utilização do ambiente, denominadas instâncias *SensorData*, são enviados a um serviço *web* chamado de *Sensorbase* para armazenamento em formato XML. A Figura 2 mostra o processo de coleta do HACKYSTAT utilizando um sensor para a ferramenta ECLIPSE, o HACKYSTAT SENSOR ECLIPSE[3].

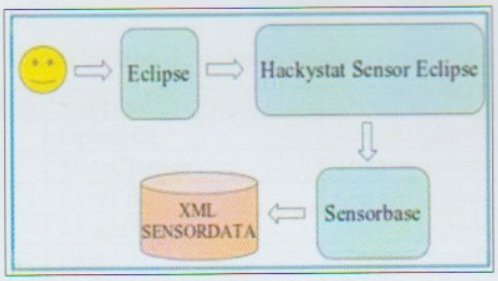


**Figura 2** - Processo de coleta de dados utilizando o HACKYSTAT SENSOR ECLIPSE (Fonte: Autor)

De acordo com a documentação do HACKYSTAT, cada instância é composta de sete campos: *Timestamp*, *Resource*, *Runtime*, *SensorDataType*, *Owner*, *Tool* e *Properties*, sendo os seis primeiros campos obrigatórios e o sétimo (*Properties*) formado por uma lista de pares chave-valor que fornece dados de tipos específicos. A partir da Tabela 1 é possível relaciona os campos obrigatórios de uma instância *SensorData*.

---

[2] Maiores informações em: <http://www.eclipse.org/>
[3] Maiores informações em: < http://code.google.com/p/hackystat-sensor-eclipse/>

**Tabela 1**- Campos Obrigatórios de uma Instância SensorData ([HAC12b]).

| Campo | Descrição |
|---|---|
| Timestamp | Momento (tempo) em que os dados foram coletados, em xs: *DateTime* formato (ou *XMLGregorianCalendar*). Ex.: 2012-04-04T11:50:31.798-03:00. |
| Runtime | Indica quais instâncias de dados de sensores pertencem a um conjunto. Ex.: 2012-04-04T11:50:31.798-03:00. |
| SensorDataType | O tipo de dados. Ex.: *DevEvent*, *Build*, *Commit*, *Codeissue*, *FileMetric*, *Coupling*, *Coverage*, *UnitTest*. |
| Resource | Artefato a partir do qual os dados do sensor foram gerados. Ex.: file:/C:/Pro/src/Soma.java. |
| Owner | A conta do usuário. Ex.: francielebeal@gmail.com. |
| Tool | O nome da ferramenta que gerou os dados. Ex.: ECLIPSE, ANT, SVN, CHECKSTYLE, PMD, FINDBUG e JDEPEND. |

A lista de propriedades com pares chave-valor é construída de acordo com o sensor utilizado. Os sensores determinam a lista de acordo com os dados que eles coletam. A Tabela 2 mostra uma lista com os tipos de dados coletados pelo HACKYSTAT SENSOR ECLIPSE para um arquivo Java ativo. No Apêndice C, esta tabela é reapresentada com exemplos para cada um dos eventos coletados.

**Tabela 2** - Principais tipo de dados coletados (Fonte: Adaptado de [USE13]).

| Tool | SensorDataType | Type | Subtype | Descrição |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | OpenFile | Gerado quando um novo arquivo é aberto. |
| Eclipse | DevEvent | Edit | StateChange | Gerado quando o arquivo ativo mudou. |
| Eclipse | DevEvent | Edit | SaveFile | Gerado quando o arquivo corrente é salvo. |
| Eclipse | DevEvent | Edit | CloseFile | Gerado quando o arquivo corrente é fechado. |
| Eclipse | DevEvent | Edit | Exit | Gerado quando se sai do Eclipse. |
| Eclipse | DevEvent | Edit | ProgramUnit(Add) | Gerado quando é adicionado à uma unidade de programa, uma declaração de importação, classe, campo ou método. |
| Eclipse | DevEvent | Edit | ProgramUnit(Rename) | Gerado quando um classe, campo ou método é renomeado. |
| Eclipse | DevEvent | Edit | ProgramUnit(Remove) | Gerado quando uma importação, classe, campo ou método é removido. |
| Eclipse | DevEvent | Edit | ProgramUnit(Move) | Gerado quando uma importação, classe, campo ou método é movido de um local para outro. |
| Eclipse | DevEvent | Build | Compile | Gerado quando uma construção (ou compilação) falha. Captura o erro que ocorrem em uma classe java ativa no editor do ECLIPSE. |
| Eclipse | DevEvent | Debug | Breakpoint(set/unset) | Gerado quando um Breakpoint é inserido/removido. |
| Eclipse | DevEvent | Debug | Start | Gerado quando é iniciada uma sessão de Debug. |
| Eclipse | DevEvent | Debug | StepInto | Gerado quando, em um sessão de Debug, entra em um bloco de código. |
| Eclipse | DevEvent | Debug | StepOver | Gerado quando, em um sessão de Debug, passa-se sobre um bloco de código. |
| Eclipse | DevEvent | Debug | Terminate | Gerada quando a sessão de Debug é finalizada. |

O HACKYSTAT vem sendo utilizado tanto em projetos acadêmicos quanto em âmbito profissional. No meio acadêmico é utilizado na área de Engenharia de Software como uma ferramenta de apoio para estudos em métricas de software e Gerência de Projetos [SOM11]. Na área profissional, é integrado à estrutura de desenvolvimento e fornece dados úteis para apoiar o planejamento de projetos, gerência de recursos e processos de garantia de qualidade [HAC12a].

### 2.2.2 Plugin METRICS

O Plugin METRICS [MET13] é um projeto sob licença EPL[4] (do inglês, *Eclipse Public License)* para o IDE ECLIPSE. O Plugin calcula métricas de qualidade de código-fonte Java, fornecendo o valor da medida, a média e o desvio padrão por recursos (projeto/classe/método).

A Figura 3 mostra a guia do plugin que exibe o cálculo de métricas de um projeto exemplo, em formato de árvore, onde os níveis podem ser expandidos em nível de pacote e classes. A Figura é uma captura de tela da guia do plugin no IDE ECLIPSE.

Problems | Javadoc | Declaration | Metrics - MyBMI - McCabe Cyclomatic Complexity (avg/max per method)

| Metric | Total | Mean | Std. Dev. | Resource causing Maximum |
|---|---|---|---|---|
| McCabe Cyclomatic Complexity (avg/max per | | 1,667 | 1,434 | /MyBMI/src/IMC.java |
| Number of Parameters (avg/max per method) | | 1,167 | 0,687 | /MyBMI/src/IMC.java |
| Nested Block Depth (avg/max per method) | | 1,25 | 0,722 | /MyBMI/src/Main.java |
| Afferent Coupling (avg/max per packageFragn | | 0 | 0 | /MyBMI/src |
| Efferent Coupling (avg/max per packageFragm | | 0 | 0 | /MyBMI/src |
| Instability (avg/max per packageFragment) | | 1 | 0 | /MyBMI/src |
| Abstractness (avg/max per packageFragment) | | 0,167 | 0,167 | /MyBMI/src/bd |
| Normalized Distance (avg/max per packageFra | | 0,167 | 0,167 | /MyBMI/src/bd |
| Depth of Inheritance Tree (avg/max per type) | | 0,833 | 0,373 | /MyBMI/src/IMC.java |
| Weighted methods per Class (avg/max per typ | 20 | 3,333 | 2,055 | /MyBMI/src/IMC.java |
| Number of Children (avg/max per type) | 1 | 0,167 | 0,373 | /MyBMI/src/bd/Connect.java |
| Number of Overridden Methods (avg/max per | 0 | 0 | 0 | /MyBMI/src/IMC.java |
| Lack of Cohesion of Methods (avg/max per typ | | 0,222 | 0,314 | /MyBMI/src/IMC.java |
| Number of Attributes (avg/max per type) | 6 | 1 | 1,414 | /MyBMI/src/IMC.java |
| Number of Static Attributes (avg/max per type | 3 | 0,5 | 1,118 | /MyBMI/src/Util.java |
| Number of Methods (avg/max per type) | 8 | 1,333 | 1,374 | /MyBMI/src/bd/Inicializa.java |
| Number of Static Methods (avg/max per type) | 4 | 0,667 | 1,106 | /MyBMI/src/Util.java |
| Specialization Index (avg/max per type) | | 0 | 0 | /MyBMI/src/IMC.java |
| Number of Classes (avg/max per packageFragr | 6 | 3 | 0 | /MyBMI/src |
| Number of Interfaces (avg/max per packageFra | 1 | 0,5 | 0,5 | /MyBMI/src/bd |
| Number of Packages | 2 | | | |
| Total Lines of Code | 89 | | | |
| src | 89 | | | |
| (default package) | 61 | | | |
| IMC.java | 25 | | | |
| Util.java | 23 | | | |
| Main.java | 13 | | | |
| bd | 28 | | | |
| Method Lines of Code (avg/max per method) | 38 | 3,167 | 3,848 | /MyBMI/src/IMC.java |

**Figura 3**- Guia de visualização das métricas calculadas pelo METRICS (Fonte: Autor).

[4] Maiores informações em: < http://opensource.org/licenses/EPL-1.0>

O plugin oferece uma opção para exportar as métricas calculadas para um arquivo XML. O arquivo XML segue a estrutura da árvore de visualização. A opção para exportar as métricas fica localizada na guia de visualização e pode ser acessada pelo botão *"Export XML"*.

Por ser um software *open source*, o METRICS foi estendido para exportar automaticamente as métricas calculadas para um arquivo XML em disco, não sendo necessário clicar no botão *"Export XML"*. Desta forma, o METRICS estendido gera automaticamente um arquivo XML sempre que ocorrer uma mudança em qualquer uma das métricas monitoradas, não sendo necessário a intervenção do desenvolvedor para coletar os dados. Este arquivo é salvo em um diretório configurado manualmente nas preferências do Plugin.

O METRICS é uma ferramenta que vem sendo usada pela comunidade científica da engenharia de software, na investigação de diversos problemas, como uma ferramenta de apoio para coleta de dados sobre métricas orientadas a objetos. Um exemplo é o trabalho [BRU04] que tem como objetivo definir e avaliar um conjunto de métricas que pode ser utilizado para avaliar a testabilidade das classes de um sistema Java. Outro exemplo é o trabalho de [COR10] que utiliza o plugin para coletar métricas orientadas a objetos para correlacioná-las com métricas físicas de sistemas embarcados, para orientar os projetistas na busca de soluções de qualidade tanto em relação ao reuso, a manutenibilidade e a eficiência no uso de recursos.

Nesta seção, foram apresentados o sensor HACKYSTAT e o plugin METRICS para coleta de dados durante a codificação de software. Eles podem ser utilizados na etapa de aquisição do perfil dinâmico do desenvolvedor. Além da etapa de aquisição, existe a etapa de aprendizado do modelo. Em um perfil dinâmico, os dados se modificam ao longo do tempo, sendo necessária a aplicação de algoritmos de Aprendizagem de Máquina para aprender o modelo. Desta forma, na próxima seção, será apresentada a Classificação, uma tarefa de Aprendizagem de Máquina que pode ser utilizada no aprendizado do modelo [BAR10].

## 2.3. Aprendizado do Modelo: Classificação

A Classificação consiste em associar um determinado objeto a uma categoria de objetos com base na similaridade entre ele e os objetos da categoria. Cada objeto é representado por um vetor de valores de atributos (características) e um rótulo (classe) associado [REZ05]. Os vetores formam um conjunto de registros (ou instâncias) que é utilizado como entrada para a tarefa de classificação. Este conjunto também é chamado de base de dados.

A classificação consiste em aprender uma função alvo $f$ (modelo de classificação) para mapear cada conjunto de atributos em uma classe predefinida. É uma abordagem sistemática que utiliza como entrada um conjunto de dados rotulados para aprender um modelo de classificação. Para aprender o modelo utiliza técnicas de classificação que fazem o uso de algoritmos de Aprendizagem de Máquina (AM) para inferir o modelo adequado para os dados apresentados [TAN09]. O objetivo da classificação é encontrar um modelo que adapte-se bem aos dados de entrada e que possa prever corretamente o rótulo para instâncias desconhecidas (conjunto de teste) [FAC11]. O conjunto de teste é aplicado após a construção do modelo para avaliar o desempenho do modelo gerado.

Os rótulos são utilizados durante o processo de treinamento para aprender a descrição da instância associada. A este processo dá-se o nome de aprendizado supervisionado [WEB02].

A Figura 4 mostra um diagrama para a tarefa de classificação. Uma abordagem geral é apresentada.

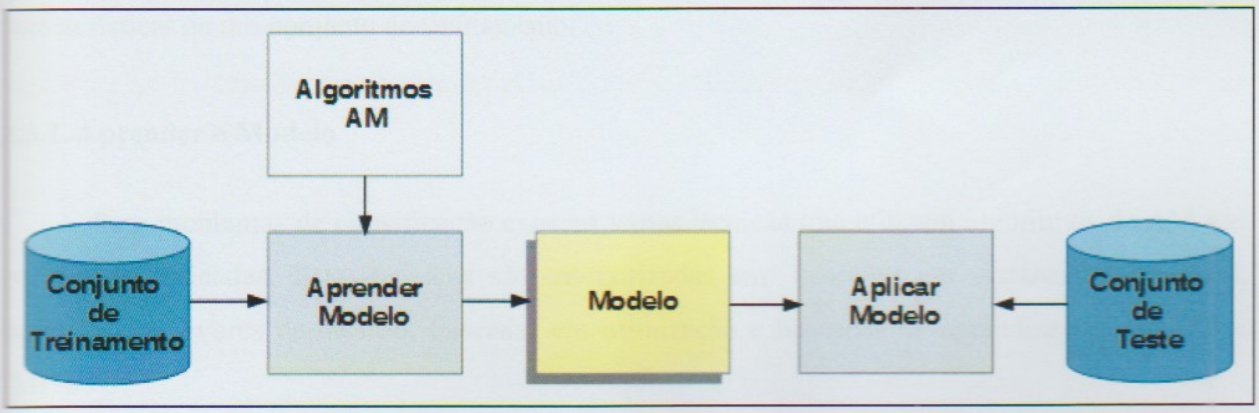


**Figura 4-** Abordagem geral para a tarefa de classificação (Fonte: Adaptado de [TAN09]).

Na sequência, a abordagem apresentada na Figura 4 é detalhada.

### 2.3.1. Conjunto de Treinamento

O conjunto de treinamento é uma base com dados utilizados no treinamento dos algoritmos de AM para aprender um modelo.

Os dados devem ser pré-processados antes de serem apresentados ao algoritmos de AM. A qualidade dos dados influencia no desempenho dos algoritmos. Existem técnicas que são utilizadas para melhorar a qualidade dos dados. São técnicas para limpeza de dados, redução da dimensionalidade, transformação e integração de dados [WIT05].

A limpeza dos dados é necessária porque os dados podem conter ruídos (distorções de valores ou erro aleatório em valor gerado), *outliers* (valores aberrantes ou muito afastados dos limites aceitáveis), valores faltantes (ausência de valor) ou valores duplicados.

A redução da dimensionalidade é aplicada quando a quantidade de atributos é muito grande, consiste em selecionar os atributos discriminantes que melhor diferenciam os objetos de diferentes classes e minimizam as diferenças entre objetos da mesma classe.

A transformação de dados consiste na conversão de tipos ou normalização de dados para facilitar o uso por diferentes algoritmos. Existem algoritmos que trabalham melhor com atributos discretos, outros com atributos contínuos [TAN09]. Atributos discretos armazenam valores finitos ou contavelmente infinitos. Estes valores podem ser categorizados (p.ex., tipo sanguíneo e tamanho {P, M, G}) ou numéricos (p.ex. contadores e binários (0,1)). Já os atributos contínuos armazenam valores reais (p.ex., peso e altura).

Existem situações em que os dados são obtidos de diversas fontes (arquivos, textos, tabelas de dados, p.ex.) e precisam ser integrados para formar os atributos e até mesmo os vetores de características de um conjunto de treinamento.

### 2.3.2. Aprender o Modelo

Para problemas de classificação existem várias técnicas que utilizam algoritmos de AM que podem ser aplicadas. Estas técnicas são categorizadas em: baseadas em instância, bayesianas, baseadas em árvores de decisão, baseadas em otimização e baseadas na aprendizagem estatística [MIT97].

O algoritmo *k*-NN pertence à categoria das técnicas baseadas em instância. Ele classifica um novo objeto com base nos *k* exemplos do conjunto de treinamento que são próximos a ele. Este algoritmo não constrói modelos, apenas memoriza os dados de treinamento. Isto faz com que seja chamado de algoritmo *lazy* ("preguiçoso"). Para classificar um novo objeto é necessário calcular a distância dele em relação a todos os outros objetos do conjunto de treinamento. O treinamento é simples, porém a classificação pode ser custosa para um conjunto de treinamento com um número grande de objetos [FAC11].

O classificador Naive Bayes é um classificador probabilístico que faz parte das técnicas bayesianas. Considera "ingenuamente" cada atributo e rótulo de classe como variável independente (aleatória), o que faz com que seu cálculo seja simplificado. A classificação consiste em: dado um novo exemplo (desconhecido), calcular a probabilidade deste pertencer a cada possível classe e, dentre estas, prever a classe mais provável [MIT97].

A técnica baseada em Árvores de Decisão é uma das mais utilizadas em Aprendizagem de Máquina. Elas utilizam a estratégia dividir para conquistar. Um problema complexo é dividido em subproblemas menores. As soluções dos subproblemas são representadas em uma árvore de decisão. Uma árvore de decisão é formada por um conjunto de nós. Existe o nó raiz que identifica o topo da árvore, nós internos que contém um teste condicional para um atributo e os nós folhas que representam as classes [FAC11]. O processo de classificação ocorre de cima para baixo, um atributo é testado em cada nó, iniciando do nó raiz, até encontrar um nó folha que determina a sua classe [DUD02].

As Redes Neurais Artificiais (RNA) são sistemas computacionais distribuídos inspirados na estrutura cerebral humana. Fazem parte das técnicas baseadas em otimização. A RNA é formada por neurônios artificiais interconectados que processam funções matemáticas. Os neurônios de uma RNA podem estar distribuídos em uma ou mais camadas. As RNAs com uma camada são chamadas Perceptron de camada única e as com mais de uma camada são chamadas de Multilayer Perceptron (MLP). As redes Perceptron de camada única são utilizadas em problemas de classificação linearmente separáveis. Já as redes MLP são para problemas não linearmente separáveis. O algoritmo de treinamento mais comum das MLP é o Back-propagation, baseado em aprendizagem por correção e erro [HAY01]. RNAs são técnicas com grande capacidade de generalização, são robustas a erros e ruídos em dados de treinamento. Uma rede pode levar muito tempo para ser treinada, porém, depois de treinada, o processo de classificação de um novo exemplo é bastante rápido [MIT97].

As Máquinas de Vetores de Suporte (Support Vector Machines, SVM) [COR95] são técnicas de Aprendizagem de Máquina baseadas na teoria da aprendizagem estatística. O SVM é bem aceito pela comunidade científica porque vêm apresentando bons resultados em diversas aplicações como, por exemplo, a categorização de textos. Esta técnica utiliza um subconjunto de exemplos de treinamento para representar a fronteira (ou limite) de decisão. São os chamados vetores de suporte. Basicamente, o SVM busca construir um hiperplano como fronteira de decisão maximizando a margem de decisão entre os exemplos positivos e negativos. Para casos não linearmente separáveis, ele mapeia os dados para um novo espaço de alta-dimensionalidade, chamado de espaço de características, para que a fronteira de decisão se torne linear. Para isso, utiliza funções de *kernel* [HAY01] [TAN09].

No SVM, os *kernels* mais empregados são os Polinomiais, os Gaussianos ou RBF (*Radial-Basis Function*) e os Sigmoidais. Cada *kernel* possui parâmetros que devem ser configurados pelo usuário afim de aumentar a precisão do classificador, o *kernel* mais utilizado é o RBF [LOR03].

Além dos classificadores *k*-NN, Naive Bayes, Árvores de Decisão, RNA e SVM, existem

outros que podem ser encontrados em [DUD02] e [MIT97].

Independente do algoritmo de AM selecionado, ele é utilizado com o conjunto de treinamento para gerar um modelo de classificação. A avaliação do desempenho do algoritmo (classificador) pode ser feita utilizando diversos métodos. Neste trabalho serão abordados os métodos *Validação Cruzada* e *Holdout*.

No método *Holdout* o conjunto de dados é particionado em dois conjuntos diferentes, o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento é utilizado para treinar um classificador. O modelo de classificação inferido tem o seu desempenho avaliado no conjunto de teste. Geralmente se utiliza 2/3 dos dados para treinamento e 1/3 para teste. A proporção 50-50 também é aceita. Uma desvantagem deste método é a diminuição de registros para treinamento, já que parte deles são separados para o teste [TAN09].

O método *Validação Cruzada* consiste em dividir o conjunto de dados em $n$ partes de tamanho igual. No treinamento, durante cada execução, uma parte é selecionada para teste e as demais são utilizadas para o treinamento. Este processo é repetido $n$ vezes até que todas as $n$ partes sejam usadas uma vez para teste. Uma vantagem deste método é que ele visa utilizar ao máximos os dados para treinamento [TAN09].

### 2.3.3. Modelo

Um modelo de classificação é gerado após o treinamento de um algoritmo de AM. Um bom modelo de classificação deve se adaptar bem aos dados de entrada e ter boa capacidade de generalização, ou seja, prever corretamente os rótulos para instâncias desconhecidas com a menor taxa de erro possível.

Para avaliar o desempenho de um modelo aplica-se o conjunto de teste. O seu desempenho pode ser visualizado em uma *Matriz de Confusão* que mostra o número de acertos e erros em cada classe. A *Matriz de Confusão* exibe, de forma tabular, o número de classificações corretas e incorretas de cada classe de um conjunto de dados [FAC11].

A Figura 5 mostra um exemplo de *Matriz de Confusão* clássica para um problema de 2 classe.

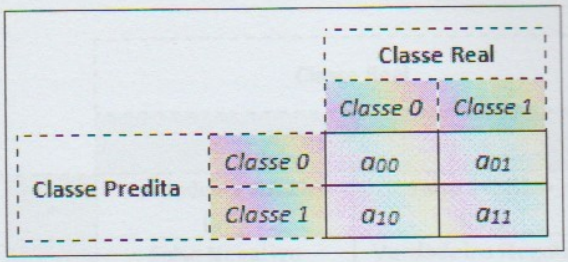


**Figura 5** - Matriz de Confusão Clássica para 2 classes (Fonte: adaptado de [TAN09]).

Analisando a *Matriz de Confusão* da Figura 5, os valores que se encontram na diagonal principal referem-se às classificações corretas para as classes 0 e 1. Já os valores acima e abaixo da diagonal principal, referem-se às classificações incorretas tanto para a classe 0 quanto para a classe 1. O total de predições corretas feitas por um modelo pode ser encontrado por meio da soma das classificações corretas da diagonal principal ($a_{00} + a_{11}$). O total de predições incorretas pela soma das classificações incorretas ($a_{01} + a_{10}$).

Existem outras duas medidas que podem ser deduzidas a partir dos valores da *Matriz de Confusão*: a *Acurácia* (Equação 2.1) e a *Taxa de Erro* (Equação 2.2). Ambas podem ser utilizadas para avaliar o desempenho de modelos diferentes. O melhor modelo pode ser considerado aquele que apresenta a maior *Acurácia* (ou menor *Taxa de Erro*) quando submetido a um conjunto de teste com instâncias desconhecidas [TAN09].

$$Acurácia = \frac{a_{00} + a_{11}}{a_{00} + a_{11} + a_{01} + a_{10}} \quad (2.1)$$

$$Taxa\ de\ Erro = \frac{a_{01} + a_{10}}{a_{00} + a_{11} + a_{01} + a_{10}} \quad (2.2)$$

Para problemas com conjuntos de dados desbalanceados, ou seja, conjuntos onde o número de instâncias de diferentes classes são desproporcionais, existem outras medidas no nível de classe que podem ser utilizadas para avaliar o desempenho de modelos diferentes. São exemplos: *Precision*, *Recall* e *F-Measure* [TAN09]. Para entender como estas medidas são calculadas, considera-se a *Matriz de Confusão* da Figura 6.

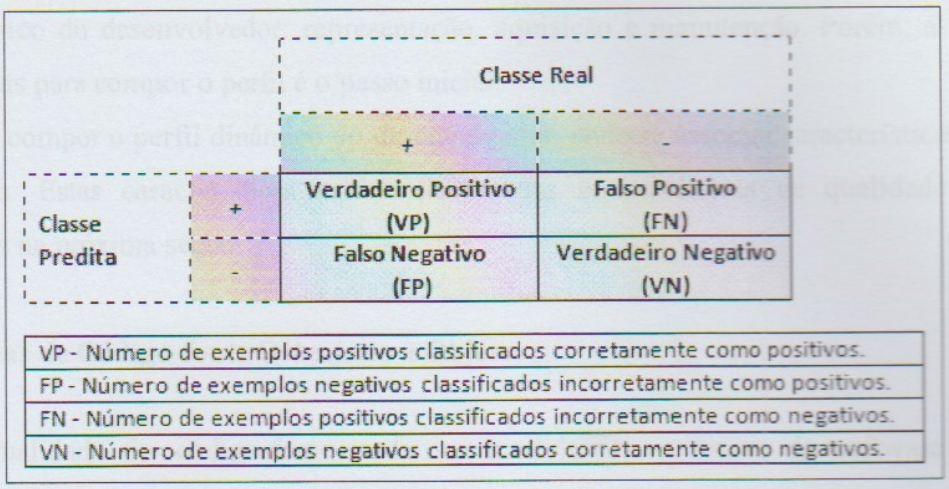


**Figura 6-** Matriz de Confusão com definição de VP, FP, FN e VN (Fonte: adaptado de [TAN09].

A *Precision* representa o percentual de exemplos positivos classificados corretamente como sendo da classe positiva (Equação 2.3). A *Recall* é o percentual de exemplos positivos classificados corretamente pelo modelo (Equação 2.4). A *F-Measure* leva em consideração as medidas *Precision* e *Recall*. Ela faz a média harmônica destas (Equação 2.5) [TAN09].

$$Precision = \frac{VP}{VP + FP} \tag{2.3}$$

$$Recall = \frac{VP}{VP + FN} \tag{2.4}$$

$$F - Measure = \frac{2}{\frac{1}{R} + \frac{1}{P}}, \text{ Sendo R = } Recall \text{ e P = } Precision. \tag{2.5}$$

### 2.3.3. Conjunto de Teste

Consiste em um conjunto de dados com instâncias desconhecidas ao modelo de classificação. O objetivo deste conjunto é avaliar o desempenho do modelo em classificar instâncias novas, que ainda não foram apresentadas a ele. É a partir deste conjunto de dados que é possível avaliar a capacidade de generalização do modelo.

Nesta seção, a tarefa de classificação foi apresentada. A classificação pode ser utilizada para o aprendizado do perfil dinâmico do desenvolvedor.

A etapa de aprendizado do perfil é tão importante quanto as demais etapas do processo do

perfil dinâmico do desenvolvedor: representação, aquisição e manutenção. Porém, a seleção das características para compor o perfil é o passo inicial.

Para compor o perfil dinâmico do desenvolvedor, pode-se associar características obtidas do código-fonte. Estas características estão relacionadas com métricas de qualidade que serão apresentadas na próxima seção.

## 2.4. Métricas de Código-fonte Orientado a Objetos

A qualidade do código-fonte pode ser medida por métricas de software voltadas à complexidade e à manutenibilidade. Estas medidas procuram gerenciar e reduzir a complexidade das estruturas, melhorar a manutenibilidade e o desenvolvimento dos códigos-fonte e consequentemente o próprio software [YU10].

A seguir, são apresentadas métricas de código-fonte bastante conhecidas pela comunidade de desenvolvimento de software orientado a objetos. Estas medidas foram agrupadas nas seguintes categorias: complexidade, herança, tamanho e agrupamento. Cada grupo é apresentado em diferentes tabelas que exibem a definição e a fórmula para cada métrica. É importante salientar que o agrupamento adotado neste trabalho é somente para fins de apresentação. Na literatura estas medidas podem aparecer em agrupamentos diferentes.

A Tabela 3 apresenta as métricas do grupo complexidade. A complexidade do software dificulta a compreensão, a manutenção e o teste do software. Estas medidas podem ser utilizadas para medir quão complexo está o software no nível de código-fonte.

**Tabela 3**- Definições e Fórmulas de Métricas de Complexidade (Fonte: Autor).

| Métrica | Definição | Fórmula |
|---|---|---|
| Complexidade Ciclomática de McCabe (CCM) [MCC76] | Mede a quantidade de caminhos lógicos, linearmente independentes, de um código-fonte. Em que a estrutura de um programa é representada por um grafo *V(G)*, com *n* vértices e com *e* arestas. Onde vértices são os comandos e as arestas conectam os comandos que podem ser executados um imediatamente após o outro. | $V(G) = e - n + 2$ |
| Métodos Ponderados por Classe (WMC) [CHI91] | Mede a complexidade de uma classe levando em consideração a complexidade de seus métodos. É o somatório das complexidades de cada método de uma classe. A complexidade dos métodos pode ser calculada utilizando CCM. | $WMC = \sum_{i=1}^{n} Ci$<br>Onde, $n$ = nº de métodos e $Ci$ = complexidade dos métodos. |
| Falta de Coesão em Métodos (LCOM*) [HEN96] | Mede a coesão entre os métodos de uma classe. Uma classe coesa representa apenas um conceito. Caso isso não ocorra, a classe deve ser dividida em classes menores. Classes que executam mais de uma função são mais complexas e difíceis de serem compreendidas e testadas | $LCOM^* = \frac{(\frac{1}{a}\sum_{i=1}^{a} m(Ai)) - m}{1 - m}$ |

Continua.

| | | |
|---|---|---|
| | [HOR04].<br>Uma classe é considerada coesa quando todos os seus métodos acessam todos os atributos (LCOM* = 0).<br>LCOM* próximo de 0 = coesão;<br>LCOM* próximo de 1 = falta de coesão;<br>LCOM* > 1 = problemas sérios de coesão. | Onde, *m(Ai)* é o nº de métodos que acessam cada atributo. |
| Profundidade de Blocos Aninhados (NBD) [MAR02] | Calcula a profundidade de blocos de instruções aninhados.<br>Esta medida é utilizada para avaliar a complexidade do código.<br>Um valor alto de NBD indica código complexo difícil de ser compreendido e mantido. | $NBD = bd$<br>Onde, *bd* é a profundidade de blocos aninhados. |

Em Apêndice B, são apresentados exemplos de aplicações de algumas das métricas de complexidades apresentadas, as mais difíceis de serem compreendidas.

As próximas métricas a serem apresentadas são as métricas de herança. Estas medidas são importantes porque podem indicar problemas de abstração e manutenção de software. Na Tabela 4 são apresentadas as definições e fórmulas para as métricas de herança.

**Tabela 4-** Definições e Fórmulas de Métricas de Herança (Fonte: Autor).

| Métrica | Definição | Fórmula |
|---|---|---|
| Profundidade da Árvore de Herança (DIT) [CHI91] | Mede a quantidade de classes ancestrais que antecedem uma classe numa árvore de herança.<br>Uma classe com DIT alto significa que ela herda muitos métodos de superclasses o que torna sua manutenção mais difícil [HEN96]. | $DIT = n$<br>Onde, $n$ = nº de classes ancestrais. |
| Número de Filhos (NOC) [CHI91] | Mede o número de subclasses imediatas de uma classe em uma árvore de herança.<br>Uma classe com muitos filhos (subclasses) pode indicar reuso, mas, pode indicar também problemas de abstração [ROS99]. | $NOC = n$<br>Onde, $n$ = nº de subclasses imediatas. |
| Métodos Sobrescritos (NORM) (HENDERSON-SELLERS 1996 *apud* LORENZ e KIDD, 1994) [HEN96] | Mede o número de métodos sobrescritos por uma subclasse.<br>Um valor alto de NORM indica que a subclasse sobrescreveu o comportamento da superclasse. O ideal nestes casos, é que a subclasse crie novos métodos em vez de sobrescrever os métodos da superclasse. | $NORM = mo$<br>Onde, *mo* é o nº métodos sobrescritos. |
| Índice de Especialização (SIX) (HENDERSON-SELLERS 1996 *apud* LORENZ e KIDD, 1994) [HEN96] | Calcula o índice de especialização de cada subclasse.<br>Um número alto de SIX pode indicar problemas de abstração, subclasses de uma hierarquia estão reescrevendo muitos métodos das superclasses [PRE02]. | $SIX = \frac{NORM * DIT}{NOM}$<br>Onde, NOM é nº de métodos da subclasse. |

Outro grupo de métricas abordado neste trabalho é o das métricas de tamanho. Estas métricas podem indicar problemas de compreensão, manutenção e reuso. O grupo é apresentado na

Tabela 5.

**Tabela 5-** Definições e Fórmulas de Métricas de Tamanho (Fonte: Autor).

| Métrica | Definição | Fórmula |
|---|---|---|
| Linhas de Código por Métodos (MLOC) [HEN96] | Calcula o número de linhas de código de métodos, desconsiderando linhas de comentários e linhas em branco. Métodos com MLOC alto são mais difíceis de serem mantidos. O aumento de MLOC torna o método mais complexo. | $MLOC = n$ Onde, $n$ = nº de linhas de código do método. |
| Número de Atributos por Classe (NAC) [PRE02] | Refere-se a contagem de atributos de uma classe. Um número grande de atributos pode dificultar a compreensão e a manutenção de uma classe. | $NAC = n$ Onde, $n$ = nº de atributos da classe. |
| Número de Atributos Estáticos (NSF) [HOR04] | Calcula a quantidade de atributos estáticos de uma classe. Campos estáticos dificultam o entendimento e a manutenção do software. | $NSF = ns$ Onde, $ns$ = nº de atributos estáticos da classe. |
| Número de Métodos Estáticos (NSM) [HOR04] | Calcula a quantidade de métodos estáticos de uma classe. Os atributos e métodos estáticos podem ser chamados diretamente sem necessidade de criar um objeto da classe. Apesar de eles melhorarem a performance de um sistema, eles devem ser usados para fins específicos. Métodos estáticos em excesso podem indicar programação procedural e não OO. | $NSM = sm$ Onde, $sm$ é o nº de métodos estáticos. |
| Número de Parâmetros (PAR) [HOR04] | Calcula a quantidade de parâmetros passados em um método. Uma classe com um número muito grande de parâmetros pode indicar que ela executa mais de uma função e precisa ser dividida em subclasses. | $PAR = np$ Onde, $np$ é o nº de parâmetros. |
| Número de Interfaces (NOI) [HOR04] | Refere-se à quantidade de interfaces. As interfaces promovem o reuso, pois declaram um conjunto de métodos e suas assinaturas, sem implementações, deixando a cargo das classes que as implementam definir as implementações dos métodos. Interfaces reduzem o acoplamento entre as classes. | $NOI = i$ Onde, $i$ = nº de interfaces. |
| Número de Pacotes (NOP) [HOR04] | É a contagem de pacotes definidos em um projeto. Os pacotes estruturam e organizam classes relacionadas, separando classes especificas de classes utilitárias que podem ser compartilhadas com outros programas. Os pacotes promovem o reuso e facilitam a manutenção. | $NOP = p$ Onde, $p$ = nº de pacotes. |

O forte acoplamento em software inviabiliza o resuo e dificulta a manutenção. Na Tabela 6 são apresentadas algumas métricas de acoplamento.

**Tabela 6-** Definições e Fórmulas de Métricas de Acoplamento (Fonte: Autor).

| Métrica | Definição | Fórmula |
|---|---|---|
| Acoplamento Aferente (Ca) [MAR94] | É o número de classes de outros pacotes que dependem de classes de um pacote considerado. Um software que apresenta seus pacotes fortemente acoplados (ou dependentes) é mais difícil de ser mantido, modificado e reutilizado. | $Ca = n$ Onde, $n$ é o nº de classes de outros pacotes que dependem de classes de um pacote considerado. |

Continua.

| | | |
|---|---|---|
| Acoplamento Eferente (Ce) [MAR94] | É o número de classes de um pacote que dependem de classes de outros pacotes.<br>Um software que apresenta seus pacotes fortemente acoplados (ou dependentes) é mais difícil de ser mantido, modificado e reutilizado. | $Ce = n$<br>Onde, *n* é o nº de classes de um pacote que dependem de classes de outros pacotes. |
| Instabilidade (I) [MAR94] | Um pacote é instável quando ele depende de muitos outros pacotes. Quando menor a dependência, mais estável ele é, indicando condições de reuso.<br>Tem como resultado valores entre o intervalo [0,1]. Valores próximos de zero, indicam estabilidade e valores próximos de um, instabilidade. | $I = \frac{Ce}{(Ca+Ce)}$ |
| Abstração (A) [MAR94] | Mede o número de classes abstratas e interfaces, dividido pelo número total de tipos em um pacote.<br>Um pacote concreto inviabiliza a sua extensão porque suas classes não são abstratas. Ao mesmo tempo em que um pacote concreto e instável dificulta a manutenção. Pacotes abstratos são ideais porque suas classes abstratas podem ser estendidas sem necessidade de modificação.<br>Resulta em um valor entre o intervalo [0,1], onde um valor próximo de zero indica um pacote concreto, ou seja, um pacote onde a maioria das classes são concretas. Valor próximo de 1, indica pacote abstrato, formado na sua maior parte de classes abstratas. | $A = \frac{Na}{Nc}$<br>Onde, *Na* é o nº de classes abstratas e *Nc* é o nº de classes concretas do pacote. |
| Distância Normalizada a partir de Sequência Principal (Dn) [MAR94] | Está relacionada com a Abstração e a Instabilidade. A Sequência Principal representa os valores de equilíbrio entre a Instabilidade e a Abstração.<br>Os pacotes que se encontram na Sequência Principal apresenta um número equilibrado de classes abstratas e concretas em relação às dependências Aferentes e Eferentes. A localização ideal de um pacote na Sequência Principal é uma das extremidades da Sequência.<br>Martin [MAR94] utiliza um gráfico para demonstrar esta relação (*Figura 7*), onde, o eixo X representa a Instabilidade e o eixo Y a Abstração, ambos com valores variando entre o intervalo [0,1]; com uma reta perpendicular traçada pelas coordenadas (0,1) e (1,0) que é a chamada Sequência Principal. | $Dn = \lvert A + I - 1 \rvert$<br>Onde, *A* é o valor de Abstração e *I* o valor de Instabilidade. |

A Figura 7 mostra o Gráfico da Abstração *versus* Instabilidade utilizado para demonstrar a Sequência Principal utilizada no cálculo da Dn. As coordenadas (0,1) e (1,0) representam os tipos de pacotes ideias: *I*=0 e *A*=1 (pacote estável e abstrato) e *I* = 1 e *A* = 0 (instável e concreto).

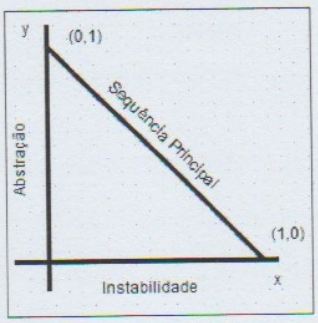


**Figura 7**- Gráfico Abstração *X* Instabilidade (Fonte: adaptado de [MAR94]).

A localização ideal de um pacote na Sequência Principal é uma das extremidades da Sequência [MAR94].

## 2.5. Considerações Finais

Neste capítulo foram abordados conceitos fundamentais que orientam este trabalho. Na seção 2.1 foi abordada a modelagem de usuário. Técnicas e métodos para composição, representação, aquisição, aprendizado e manutenção do modelo de usuário foram apresentadas.

Na seção 2.2 foram apresentados duas ferramentas para a coleta de dados implícita não invasiva: o Framework HACKYSTAT e o Plugin METRICS. Ambas foram utilizadas para coleta de dados para o perfil dinâmico do desenvolvedor proposto.

Na seção 2.3 foi abordada a tarefa de classificação da Aprendizagem de Máquina. As técnicas *k*-NN, Naive Bayes, Árvore de Decisão, RNA e SVM foram apresentadas.

Na seção 2.4 diversas métricas de qualidade de software para medir a qualidade de código-fonte foram apresentadas. Foram apresentadas métricas para softwares orientados a objetos para complexidade, herança, tamanho e acoplamento. As métricas apresentadas estão relacionadas com as características do modelo do desenvolvedor.

# Capítulo 3

# Trabalhos Relacionados

Neste capítulo são apresentados trabalhos que se relacionam com o escopo desta pesquisa e com os itens apresentados no Capítulo 2. São trabalhos que aplicam conceitos como modelagem de usuário para desenvolvedores de software, que aplicam sensores para coleta implícita de dados do código-fonte, ferramentas para obtenção de métricas a partir do código-fonte e trabalhos que aplicam a Aprendizagem de Máquina (AM) na Engenharia de Software.

Para uma melhor apresentação, os trabalhos estão agrupados por temas. Estes temas são: Modelagem de Usuário na Engenharia de Software (3.1), Sensores para Coleta de Dados sobre os Desenvolvedores de Software (3.2), Ferramentas para a Obtenção de Métricas de Código-Fonte (seção 3.3) e Aprendizagem de Máquina na Engenharia de Software (seção 3.4).

Foram pesquisados trabalhos relacionados ao problema de pesquisa, trabalho que fizessem a construção do perfil dinâmico de um participante do processo de desenvolvimento de software e a classificação dos mesmos, porém, até o momento não foi encontrado nenhum trabalho. Então, foram pesquisados trabalhos com algum tipo de relação com o trabalho de pesquisa proposto. Esses trabalhos são descritos nas próximas seções.

## 3.1. Modelagem de Usuário na Engenharia de Software

A modelagem do usuário vem sendo aplicada em diversas áreas como: sistemas de recomendação [LIU07] sistemas de hipermídia adaptativa e sistemas inteligentes de tutoria [KAV00]. Na área da engenharia de software, até o presente momento, foi encontrado apenas

um trabalho (MODUS-SD) com a aplicação da modelagem de usuário.

O MODUS-SD (*User Modelling in Cooperative Software Development*) é um módulo integrado à arquitetura CSCW-SD para modelagem de usuários proposto em [WAN12]. Em sua primeira versão, modela os participantes desenvolvedores (programadores) de um processo de desenvolvimento colaborativo de software em pequenas equipes.

O CSCW-SD (*Computer Supported Cooperative Work in Software Development*) é um projeto de pesquisa que tem como objetivo estudar e especificar ferramentas de apoio ao trabalho cooperativo suportado por computador (CSCW) em equipes de desenvolvimento colaborativo de software. É baseado em uma arquitetura multi-agente (MAS, do inglês *Multi-Agent System*) que integra diferentes ferramentas que são disponibilizadas aos usuários por meio de uma interface única fornecida por um Agente Pessoal (PA, do inglês *Personal Agent*) [FRE09].

O MODUS-SD aplica os padrões de design estático e dinâmico. Utiliza o *framework* HACKYSTAT para coletar os dados referentes ao tempo dedicado ao desenvolvimento de software durante a construção do código-fonte no IDE ECLIPSE. Por meio destes dados, a gerência de projetos pode acompanhar a evolução do projeto através de medidas, tais como: o tempo empregado na construção de cada classe por desenvolvedor, o número de linhas codificadas por desenvolvedor e, por fim, o desenvolvedor mais produtivo [WAN12].

Um aditivo para o MODUS-SD foi apresentado em [GAL13]. Uma arquitetura para a modelagem dinâmica dos usuários do MODUS-SD utilizando Mapas de Kohonens [HAY01]. O objetivo desta arquitetura é aplicar um algoritmo de Aprendizagem de Máquina sobre os dados coletados por um sensor HACKYSTAT integrado ao MODUS-SD e criar um perfil dinâmico do usuário. O modelo é uma rede SOM treinada e que representa um tipo de usuário específico.

O que diferencia a arquitetura apresentada em [GAL13] deste trabalho é que ele utiliza no aprendizado do modelo um algoritmo de agrupamento, já o método proposto utiliza algoritmos de classificação. Além disto, do modelo obtido em [GAL13] não é possível diferenciar os desenvolvedores entre si.

Considerando o MODUS-SD, o que o diferencia deste trabalho, é que ele é voltado aos dados de produtividade do desenvolvedor e não modela dados relacionados com a qualidade do código-fonte produzido, além de não fazer a classificação dos desenvolvedores em diferentes perfis.

### 3.2. Sensores para Coleta de Dados sobre os Desenvolvedores de Software

A arquitetura eXTaDEi é uma arquitetura para identificar ações executadas por um ator (desenvolvedor, analista, arquiteto, gerente, dentre outros) em um contexto de desenvolvimento de software. As ações são identificadas a partir de dados coletados por sensores Hackystat ligados ao IDE ECLIPSE. Utiliza ontologias para representar o contexto e dar significado aos dados coletados [SOU12].

O objetivo desta arquitetura é reconhecer as atividades executadas, dentre outras, por um desenvolvedor durante a codificação do código-fonte. O reconhecimento é feito com base em dados coletados pelos sensores que são armazenados em sequências temporais. A partir desta base de dados fazer inferências sobre qual ator executou determinada atividade.

O que relaciona o trabalho de Souza *et al.* (2012) com o presente trabalho é a utilização de sensores de coletas de dados para capturar dados sobre como os desenvolvedores constroem os códigos-fonte. A arquitetura eXTaDEi utiliza os dados para representar sequências de ações dos desenvolvedores e o presente trabalho utiliza para a construção de perfis dinâmicos dos desenvolvedores [SOU12].

### 3.3. Ferramentas para a Obtenção de Métricas de Código-Fonte

Sonar é uma ferramenta baseada na web *open source* para gerenciamento de qualidade de código-fonte. Esta ferramenta cobre mais de 20 linguagens de programação, dentre elas o Java. Sonar se propõe a cobrir sete eixos da qualidade de código: arquitetura e design, código duplicado, testes de unidade, complexidade, bugs potenciais, regras de codificação e comentários [SON13a]. Ela utiliza métricas de qualidade clássicas como: CCM, DIT, NOC e LOC, dentre outras [SON13b].

O PROM (PRO Metrics) é uma ferramenta para coleta e análise de métricas de software. A coleta dos dados é feita por plugins que fazem a coleta automática, implícita e não invasiva, de dados de ferramentas de desenvolvimento. Um dos tipos de métricas de código-fonte obtidas com o PROM são as métricas orientadas a objetos [SIL03].

### 3.4. Aprendizagem de Máquina na Engenharia de Software

Os algoritmos de AM vêm sendo aplicados em diversas tarefas da Engenharia de

Software, principalmente para prever ou estimar propriedades do processo de desenvolvimento de software como: custo, esforço, tamanho, defeitos, falhas, confiabilidade, reusabilidade, testabilidade, entre outros [DU02].

Em [BRU09] foi utilizada uma Rede Neural Artificial para classificar automaticamente artefatos do desenvolvimento de software de acordo com atividades de desenvolvimento que os originou e de acordo com o seus status, ou seja, se ainda estão em progresso ou já foram finalizados. Os artefatos possuem um conjunto de atributos que são informados manualmente pelos desenvolvedores. Estes atributos são referentes a descrição do artefato, nome, data de criação, prazo para finalização, atividade, status, desenvolvedor (ou equipe) que criou o artefato, link para outros artefatos e anotações (e.g. requisitos de software). A classificação automática tem como objetivo orientar os desenvolvedores a informar a atividade e o status do artefato.

Em [CHE11] é apresentada uma abordagem para avaliação de processo de software usando técnicas de AM. Os algoritmos de AM são utilizados para aprender sequências de execuções de processos rotuladas como *normal* ou *anormal*. Uma sequência de execução é *normal* quando atende os padrões organizacionais para processos. As sequências de execuções de processos representam as evoluções de artefatos ou atividades nos processos, em ordem temporal. Um exemplo de sequência de execução de processo é a mudança de um requisito que pode ser representada por um sequência de estados do tipo *New, Assessed, Assigned, Resolved, Verified and Closed*. Neste trabalho foram aplicados os algoritmos SVM, Naive Bayes e Árvore de Decisão (C4.5), sobre um conjunto de dados de processos de gerenciamento de defeitos de software. A abordagem de [CHE11] tem como objetivo superar as limitações do método de avaliação tradicional que é feito manualmente por usuários que avaliam as execuções dos processos e verificam se estão de acordo com os padrões organizacionais.

Em [JON13] uma abordagem é descrita para aumentar a eficiência da manipulação de relatórios de bugs, ou *Anomaly Reports* (ARs), em grandes organizações aplicando técnicas de AM. Um AR quando chega até uma organização geralmente é direcionada para equipe que analisa onde está localizada a falha na estrutura do sistema. Dependendo da localização (componente) da falha, o AR é encaminhado para a equipe responsável pelo componente. A abordagem do autor tem como objetivo diminuir o tempo deste encaminhamento por meio de uma classificação feita por algoritmos de AM. O objetivo da classificação é classificar ARs

recém-chegados para uma Classe AR. A classe AR irá representar qual equipe deve ser atribuído o AR ou qual o componente que está localizada a falha. Nos seus experimentos realizados em [JON13] foi utilizado uma combinação dos classificadores Naive Bayes, SVM, Redes Bayesianas, k-*NN* e Árvore de Decisão.

### 3.5. Considerações Finais

Neste Capítulo foram apresentados alguns trabalhos que possuem algum tipo de relação com o método proposto. Na seção 3.1 foram apresentados trabalhos que aplicam a modelagem dinâmica de usuário na Engenharia de Software. Na seção 3.2, foram apresentadas pesquisas que fazem a coleta de dados sobre o trabalho de codificação dos desenvolvedores e, que representam estes dados de alguma forma. Na seção 3.3, foram apresentados algumas ferramentas para a coleta de dados implícita sobre o trabalho dos desenvolvedores. Já na seção 3.4, foram apresentados trabalhos que aplicam algoritmos de Aprendizagem de Máquina em problemas de Engenharia de Software.

A Tabela 7 mostra 6 trabalhos considerados mais importantes comparados com o método proposto.

**Tabela 7** - Trabalhos Relacionados mais importantes(Fonte: Autor).

| Trabalhos | Perfil Dinâmico do Usuário | Engenharia de Software | Sensores para coleta de dados | Aprendizagem de Máquina - Classificação |
|---|---|---|---|---|
| [WAN12] | X | X | X | |
| [GAL13] | X | X | X | |
| [SOU12] | | X | X | |
| [BRU09] | | X | | X |
| [CHE11] | | X | | X |
| [JON13] | | X | | X |
| Método Proposto | X | X | X | X |

Como pode-se observar na Tabela 7, os dois primeiros trabalhos ([WAN12] e [GAL13]) são os que mais se aproximam do método proposto, porém, nenhum dos dois propõem algoritmos de AM para fazer a classificação de desenvolvedores.

O próximo Capítulo apresenta o método proposto para a construção do perfil dinâmico do participante desenvolvedor do desenvolvimento colaborativo de software.

# Capítulo 4

# Método Proposto

Neste capítulo é apresentado um método para a construção do perfil dinâmico do desenvolvedor. Como já foi discutido na introdução deste trabalho, tanto a complexidade quanto a manutenibilidade de software possuem origens nos desenvolvedores. Elas são determinadas pela habilidade (ou experiência) deles em construir códigos-fonte [YU10].

Neste trabalho os dados que podem significar a qualidade dos desenvolvedores são representados no perfil dinâmico do usuário. Defendemos a hipótese de que o perfil dinâmico de um desenvolvedor é um conjunto estruturado de características obtidas dos códigos-fonte produzidos por ele; e que a partir destas características é possível classificar os desenvolvedores em diferentes perfis.

Para construir o perfil dinâmico do desenvolvedor, inicialmente uma lista de características deve ser investigada para formar o perfil. Estas características devem ser obtidas dos artefatos produzidos pelos desenvolvedores, no caso o código-fonte.

Neste trabalho foram utilizadas as seguintes medidas de código-fonte para formar a lista de características: WMC, LCOM* e NBD (métricas de complexidade); DIT, NOC, NORM e SIX (métricas de herança); MLOC, NAC, NSF, NSM, PAR, NOI, NOP (métricas de tamanho); e, I, A e Dn (métricas de acoplamento).

A seleção deste conjunto de métricas se deu pelas seguintes razões: as métricas WMC, DIT e NOC pertencem ao conjunto de métricas de Chimdaber e Kemerer (CK) que, de acordo com [SOL10] é um dos conjuntos mais importantes de métricas orientadas a objetos, e, segundo [PRE10] é o conjunto mais amplamente referenciado; as métricas de tamanho e CK, segundo [PLO10], são utilizadas em muitos trabalhos para monitorar a qualidade do software em desenvolvimento, estabelecendo limiares para indicar se medidas de interesses foram

violadas, ou não; a métrica LCOM* de Henderson-Sellers foi selecionada por ser uma revisão da métrica LCOM de CK; as métricas de acoplamento (I, A e Dn), as métricas de herança (NORM e SIX) e a métrica de complexidade NBD foram selecionadas por serem aceitas e referenciadas pela comunidade científica.

Além das métricas de código-fonte, foram utilizadas as características *Debug*, *Breakpoint*, *Refactoring*, *Código de Erros* e *Quantidade do Erro*. A característica *Debug* é utilizada para indicar se o desenvolvedor utilizou, ou não, o debug. *Breakpoint* indica se foi utilizado o recurso de breakpoint. *Refactoring* indica se houve, ou não, refatoração, ou seja, se foi renomeado, removido ou movido um método, atributo ou classe. A característica *Código de Erros* se refere aos códigos dos erros cometidos pelos desenvolvedores em tempo de desenvolvimento. *Quantidade do Erro* indica o total de erros cometidos para um código de erro.

Desta forma, a lista de características do desenvolvedor é apresentada na Tabela 8. Esta lista foi obtida após estudos realizados sobre métricas de qualidade ligadas à manutenibilidade e complexidade de software. Estas métricas podem ser utilizadas como características do desenvolvedor porque os seus valores, de certa forma, revelam a qualidade do desenvolvedor.

Uma descrição para cada métrica da lista encontra-se em Métricas de Qualidade de Código-Fonte no Capítulo 2.

**Tabela 8**- Lista de Características do Desenvolvedor (Fonte: Autor).

| **Característica** |
|---|
| WMC - Métodos Ponderados por Classe |
| LCOM* - Falta de Coesão entre Métodos |
| NBD - Profundidade de Blocos Aninhados |
| DIT - Profundidade da Árvore de Herança |
| NOC - Número de Filhos |
| NORM - Número de Métodos sobrescritos |
| SIX - Índice de Especialização |
| MLOC - Linhas de Código de Métodos |
| NAC - Números de Atributos da Classe |
| NSF - Número de Atributos Estáticos de uma Classe |
| NSM - Número de Métodos Estáticos de uma Classe |
| PAR - Número de Parâmetros |

Continua.

| |
|---|
| NOI - Número de Interfaces |
| NOP - Número de Pacotes |
| I –Instabilidade |
| A –Abstração |
| Dn - Distância Normalizada |
| *Debug* - Utilização, ou não, do mecanismo de debug |
| *Breakpoint* - Utilização, ou não, do recurso breakpoint |
| *Refactoring* - Realização, ou não, de refactoring |
| *Código do Erro* - código de erro |
| *Quantidade do Erro* - quantidade de erros |

Um perfil de usuário deve ser representado em algum formato. Uma boa opção é o formato XML. Ele é uma representação simples e de fácil interpretação, tanto humana quanto computacional. A outra motivação para sua escolha é a portabilidade, já que é independente de plataforma. Além disso, a flexibilidade de sua estrutura pode acomodar novas características que podem surgir com o passar do tempo, caso seja necessário modificar o perfil dinâmico.

A Listagem 2 mostra um exemplo de vetor XML de representação do perfil dinâmico do desenvolvedor formado pelas 22 características exibidas na Tabela 8. Além das características, o modelo apresenta outras informações como a data da obtenção do modelo, hora e minuto. Também apresenta a informação da classe (ou rótulo) do desenvolvedor e o projeto do qual foram coletados os dados.

**Listagem 2**- Representação de um Perfil em Formato XML (Fonte: Autor).

```
<?xml version="1.0" encoding="UTF-8"?>
<Modelo_Dinâmico>
    <Developer>devmodel.p4005</Developer>
    <Data>06-11-2013</Data><!--Data da obtenção, ou coleta, do UM-->
    <Hora>20</Hora><!--Hora da obtenção, ou coleta, do UM-->
    <Minuto>37</Minuto><!--Minuto da obtenção, ou coleta, do UM-->
    <WMC>5.667</WMC><!--Métodos ponderados por classe-->
    <LCOM*>0.444</LCOM*><!--Falta de coesão entre métodos-->
    <NBD>1.065</NBD><!--Profundidade de blocos aninhados-->
    <DIT>3.667</DIT><!--Profundidade da árvore de herança-->
    <NOC>3</NOC><!--Número de filhos-->
    <NORM>0.667</NORM><!--Número de métodos sobrescritos-->
    <SIX>0.667</SIX><!--Índice de especialização-->
    <MLOC>1.806</MLOC><!--Linhas de código de métodos-->
    <NAC>3.667</NAC><!--Número de atributos-->
    <NSF>0.667</NSF><!--Número de atributos estáticos-->
    <NSM>0.333</NSM><!--Número de métodos estáticos-->
    <PAR>0.677</PAR><!--Número de parâmetros de métodos-->
    <NOI>0</NOI><!--Número de interfaces-->
    <NOP>2</NOP><!--Número de pacotes-->
    <I>1</I><!--Instabilidade-->
    <A>0.167</A><!--Abstração-->
    <Dn>0.167</Dn><!--Distância normalizada-->
    <Debug>0</Debug><!-- Utilização do debug-->
    <Breakpoint>0</Breakpoint><!-- Utilização do breakpoint-->
    <Refactoring>0</Refactoring><!-- Realização de refatoração de código-->
    <CodErro>0</CodErro><!--Código do erro cometido-->
    <qtdeErro>0</qtdeErro><!--Quantidade do erro cometido-->
    <projeto>MyBMI</projeto><!--Projeto do qual foram obtidos os dados para o UM-->
    <classe>P4000</classe><!--Classe que pode ser P2000, P4000 ou P6000-->
</Modelo_Dinâmico>
```

O método proposto utiliza a lista de características da Tabela 8 para construir o perfil dinâmico do desenvolvedor. As etapas do método são apresentadas na Figura 8. As etapas estão destacadas por retângulos tracejados. A primeira etapa corresponde a Aquisição dos Perfis e a formação de uma base. A segunda, refere-se à geração de um Modelo de Classificação, a terceira corresponde a utilização do Modelo gerado e, quarta etapa à manutenção do Modelo. Na sequência, cada etapa é descrita.

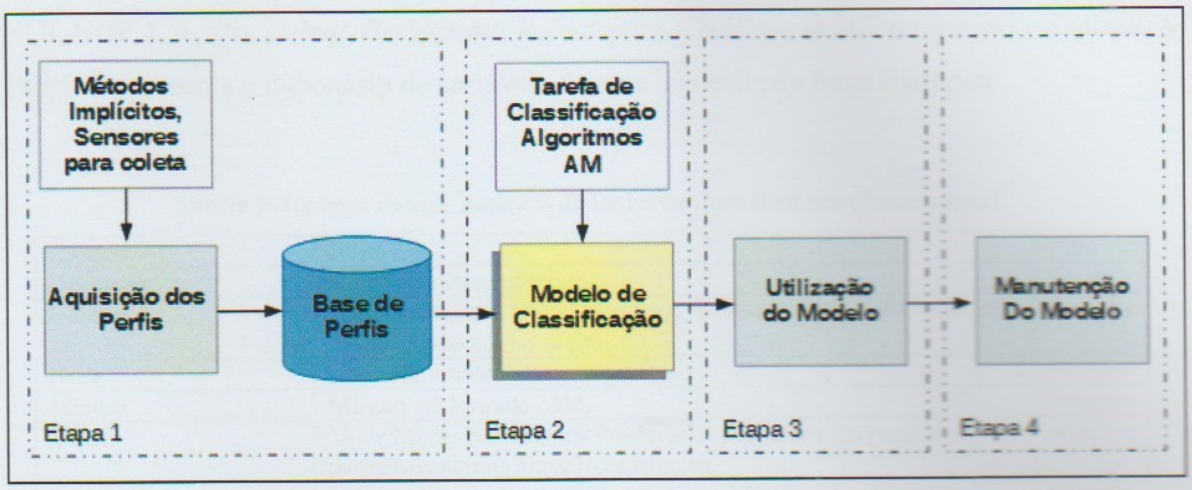


**Figura 8**- Etapas do Método Proposto (Fonte: Autor).

### 4.1. Aquisição dos Perfis

A etapa de Aquisição dos Perfis refere-se à coleta de dados para adquirir os perfis dos desenvolvedores. Os perfis são adquiridos ao longo do trabalho de codificação formando uma base histórica de perfis.

Os perfis são formados pelas características da Tabela 10. Sempre que o valor de uma das características do perfil de um desenvolvedor mudar, o perfil antigo do desenvolvedor deve ser mantido e um novo perfil deve ser criado e preenchido com os valores mais recentes, formando assim a base de perfis.

Para obter os perfis, os dados devem ser obtidos automaticamente do código-fonte por meio de sensores de coleta ligados aos ambientes de desenvolvimento. Os sensores devem utilizar o método implícito para a coleta.

A construção da base de perfis é necessária porque para construir o perfil dinâmico do desenvolvedor proposto neste trabalho é necessária uma base com dados coletados sobre o desenvolvimento de software realizado por diferentes desenvolvedores Java. Este tipo de base não está disponível publicamente, por isso, deve ser construída.

A base histórica de perfis (ou somente Base Histórica) armazena instâncias de dados, onde, cada instância corresponde a um perfil de um desenvolvedor que foi coletado ao longo do tempo.

As instâncias de dados são compostas pelos seguintes campos: *developer*, data, hora, minuto, WMC, LCOM, NBD, DIT, NOC, NORM, SIX, MLOC, NAC, NSF, NSM, PAR,

NOI, NOP, I, A, Dn, *Debug*, *Breakpoint*, *Refactoring*, *CodErro*, *qtdeErro*, projeto e a classe. A Tabela 9 apresenta o dicionário de variáveis de uma instância da Base Histórica.

**Tabela 9**- Campos de uma Instância de Dados da Base Histórica (Fonte: Autor).

| Campo | Descrição | Tipo |
|---|---|---|
| Developer | Nome do desenvolvedor. | Texto |
| Data | Data em formato DD:MM:AAAA, onde DD – dia, MM – mês e AAAA – ano com 4 dígitos. | Número |
| Hora | Hora em formato HH. | Número |
| Minuto | Minuto em formato MM. | Número |
| WMC | Valor Médio de Métodos Ponderados por Classe do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| LCOM* | Valor Médio de Falta de Coesão entre Métodos do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| NBD | Valor Médio de Profundidade de Blocos Aninhados de NBD do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| DIT | Valor Médio de Profundidade da Árvore de Herança do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| NOC | Valor Médio de Número de Filhos do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| NORM | Valor Médio de Número de Métodos sobrescritos do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| SIX | Valor Médio de Índice de Especialização do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| MLOC | Valor Médio de linhas de Código de Métodos do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| NAC | Valor Médio de Números de Atributos da Classe do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| NSF | Valor Médio de Número de Atributos Estáticos de uma Classe do desenvolvedor na Data/Hora/Minuto. | Número |
| NSM | Valor Médio de Número de Métodos Estáticos de uma Classe do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| PAR | Valor Médio de Número de Parâmetros do desenvolvedor na Data/Hora/Minuto. | Número |
| NOI | Valor Médio de Número de Interfaces do desenvolvedor na Data/Hora/Minuto. | Número |
| NOP | Valor Médio de Número de Pacotes do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| I | Valor Médio de Instabilidade do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| A | Valor Médio de Abstração do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| Dn | Valor Médio de Distância Normalizada do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| Debug | Valor de Utilização do mecanismo de debug do projeto do desenvolvedor na Data/Hora/Minuto.<br>Debug = 0 --> uso do debug;<br>Debug = 1 --> não uso do debug. | Número |
| Breakpoint | Valor de utilização do recurso breakpoint do projeto do desenvolvedor na Data/Hora/Minuto.<br>Breakpoint = 0 --> uso do breakpoint;<br>Breakpoint = 1 --> não uso do breakpoint. | Número |
| Refactoring | Valor de realização de refactoring do projeto do desenvolvedor na | Número |

Continua.

| | | |
|---|---|---|
| | Data/Hora/Minuto.<br>Refactoring = 0 --> uso do refactoring;<br>Refactoring = 1 --> não uso do refactoring. | |
| codErro | Código do Erro. É um número que identifica o erro cometido no projeto do desenvolvedor na Data/Hora/Minuto. Caso não ocorreu erro no instante de tempo, assume valor 0. | Número |
| qtdeErro | Quantidade do erro cometido. É um número inteiro que conta o erros cometidos para o código de erro, do projeto do desenvolvedor na Data/Hora/Minuto. | Número |
| Projeto | Projeto do qual foram coletadas as medidas para o desenvolvedor. | Texto |
| Classe | Classe (ou rótulo) do desenvolvedor. | Texto |

Para obter dados para a Base Histórica, deverão ser coletados dados de desenvolvedores com experiência na linguagem Java diferentes. Os desenvolvedores deverão ter perfis (classes) de experiências diferentes.

Os dados devem ser coletados enquanto os desenvolvedores escrevem o código-fonte. Sensores devem ser ligados ao IDE para coletar os dados automaticamente enquanto eles codificavam. A Figura 9 resume o processo de coleta de dados pelos sensores enquanto os desenvolvedores utilizam um IDE.

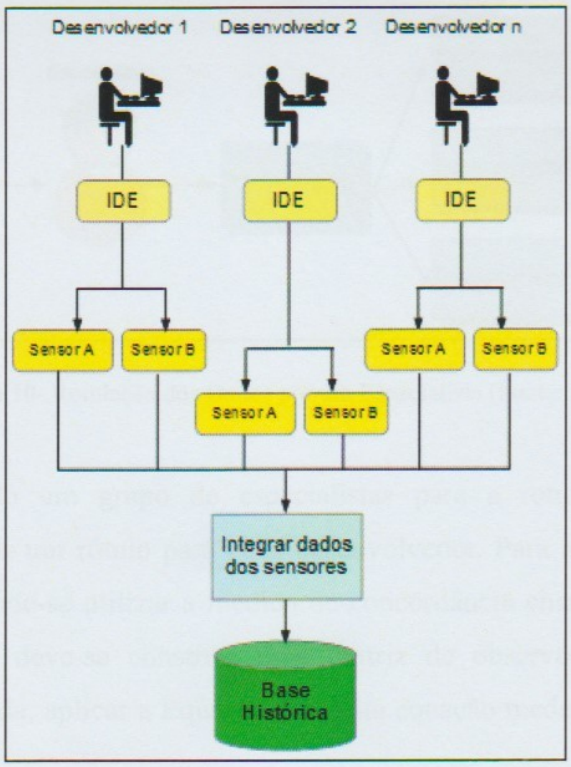


**Figura 9**- Processo de Coleta de Dados por Sensores (Fonte: Autor).

Quando utilizado mais de um sensor de coleta, os dados coletados por eles devem ser integrados (unidos) para formar as instâncias de dados da Base Histórica.

As instâncias da Base Histórica serão utilizadas para treinar algoritmos de AM para gerar um modelo de classificação. Um especialista, ou grupo, deve atribuir uma classe para cada desenvolvedor que participou da formação da base histórica. As instâncias de cada desenvolvedor deverão ser rotuladas com a classe ao qual ele pertença.

O especialista (ou o grupo) deve ter conhecimentos em Recursos Humanos, Engenharia de Software e, principalmente conhecer os desenvolvedores e o nível de experiência de cada um para atribuir a classe correta. A Figura 10 mostra um exemplo de como deve ser feita a rotulação das instâncias da Base Histórica, onde, um especialista usa as classes "Pouco Experiente", "Experiente" e "Muito Experiente" para rotular os desenvolvedores. Outros exemplos de classes, como "Iniciante" ou "Avançado" ou "Junior", "Pleno" ou "Sênior" podem ser utilizadas. Classes genéricas como "P2000", "P4000" e "P6000" também podem ser usadas.

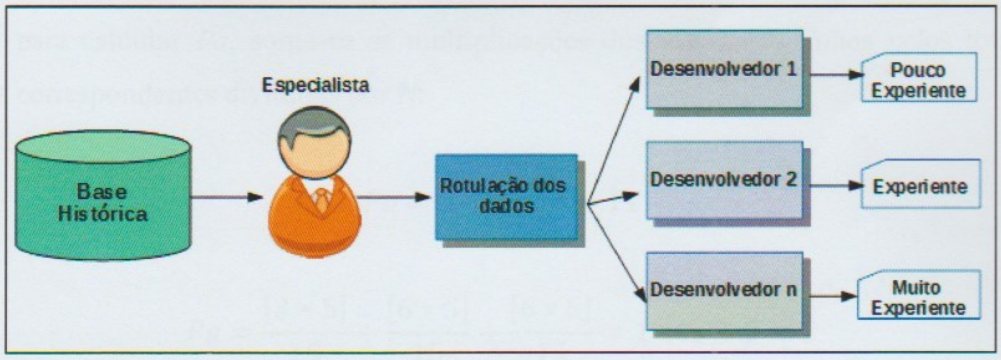


**Figura 10**- Rotulação dos Dados por um Especialista (Fonte: Autor).

Quando utilizado um grupo de especialistas para a rotulação dos dados, cada especialista deve atribuir um rótulo para cada desenvolvedor. Para avaliar o grau de acordo entre os especialistas pode-se utilizar a medida de concordância chamada coeficiente Kappa [COH60]. Nesse caso, deve-se construir uma matriz de observações para cada par de especialistas e em seguida, aplicar a Equação 5.1. Esta equação mede o grau de concordância entre os especialistas.

$$k = \frac{Po - Pa}{N - Pa} \qquad (5.1)$$

Onde, *Po* é a proporção de unidades que os especialistas classificaram nas mesmas classes, *Pa* a proporção de unidades classificadas pelos especialistas nas mesmas classes por mera coincidência e *N* o número de observações [FON07].

A Tabela 10 mostra um exemplo de matriz de observações de dois especialistas que rotularam 15 desenvolvedores nas classes genéricas "P2000", "P4000" e "P6000". Cada desenvolvedor poderia ser rotulado com uma das classes.

**Tabela 10**- Matriz de observações(Fonte: adaptado de [FON07]).

| | **Especialista 1** | | | |
|---|---|---|---|---|
| **Especialista 2** | P2000 | P4000 | P6000 | Total |
| P2000 | 3 | 1 | 1 | 5 |
| P4000 | 0 | 4 | 1 | 5 |
| P6000 | 0 | 1 | 4 | 5 |
| Total | 3 | 6 | 6 | 15 |

Para encontrar o valor de *Po*, soma-se os valores das células da diagonal principal da matriz; para calcular *Pa*, soma-se as multiplicações dos totais das linhas pelos totais das colunas correspondentes divididas por *N*:

$$Po = 3 + 4 + 4 = 11$$

$$Pa = \frac{[3 * 5]}{15} + \frac{[6 * 5]}{15} + \frac{[6 * 5]}{15} = 1 + 2 + 2 = 5$$

$$N = 15$$

Aplicando a Equação 5.1:

$$k = \frac{11 - 5}{15 - 5} = \frac{6}{10} = 0{,}6$$

Embora não exista um valor específico que indique o valor adequado de Kappa, pode-se encontrar na literatura sugestões de valores de referência para o índice, por exemplo, os valores de propostos em Fleiss [FLE81] apresentados na Tabela 11:

**Tabela 11**- Valores de concordância Kappa propostos por [FLE81] (Fonte: adaptado de [FON07]).

| Valor do Kappa | Grau de concordância |
|---|---|
| < 0.40 | baixo |
| 0.40 a 0.75 | satisfatório a bom |
| > 0.75 | excelente |

Interpretando o resultado do exemplo utilizando os valores de referência da Tabela 11, onde $k=0,6$, tem-se um grau de concordância entre os especialistas *satisfatório a bom*.

O processo de Aquisição de Perfis é dinâmico, conforme o trabalho dos desenvolvedores evolui, os perfis são obtidos e armazenados na Base Histórica. Com o tempo esta base pode se tornar gigantesca, por isso, deve-se estabelecer um prazo de tempo para manter os perfis na base. Um critério que pode ser adotado é manter os perfis dos últimos 2 anos de trabalho de cada desenvolvedor, ou manter os últimos 2.000 perfis de cada desenvolvedor, e, sempre que for adicionado um novo perfil, remover o mais antigo.

A próxima etapa do método corresponde ao Modelo de Classificação. Nesta etapa, algoritmos de AM são utilizados para aprender um modelo para classificar os desenvolvedores de acordo com as classes atribuídas pelo especialista. Esta fase é apresentada na seção seguinte.

## 4.2. Modelo de Classificação

Nesta etapa uma tarefa de classificação é utilizada para construir um modelo de classificação. O modelo será utilizado para classificar os desenvolvedores em diferentes categorias (classes) determinadas por um especialista.

Um conjunto de algoritmos (classificadores) candidatos deve ser analisado e testado. O algoritmo que apresentar melhor desempenho dentre os candidatos deve ser o escolhido.

Os classificadores utilizados pelo método proposto são: o algoritmo *k*-NN, o classificador Naïve Bayes, Árvore de Decisão, as Redes Neurais Artificiais e o SVM. Estes classificadores foram selecionados porque são difundidos e aceitos pela comunidade científica. Também são os mais conhecidos dentre os classificadores dos métodos aos quais eles pertencem e por representarem diferentes paradigmas de AM.

Para o treinamento dos classificadores é sugerida a *Validação Cruzada* e para analisar os resultados dos desempenhos, a *Matriz de Confusão, Acurácia* e *F-Measure*.

Para a tarefa de classificação desta etapa, como entrada são utilizadas as instâncias da

Base Histórica. Deve ser formado um conjunto com instâncias para o treinamento classificadores.

A Figura 11 mostra o processo de treinamento dos classificadores e avaliação dos desempenhos. Inicialmente, seleciona-se um classificador candidato (por exemplo: *k*-NN, Naïve Bayes, árvore de Decisão, RNA e SVM) e este é treinado utilizando a Base de Treinamento com *Validação Cruzada*. O resultado do desempenho de cada modelo pode ser avaliado pela *Matriz de Confusão, Acurácia* e *F-Measure*. O processo termina quando todos os classificadores forem treinados e avaliados.

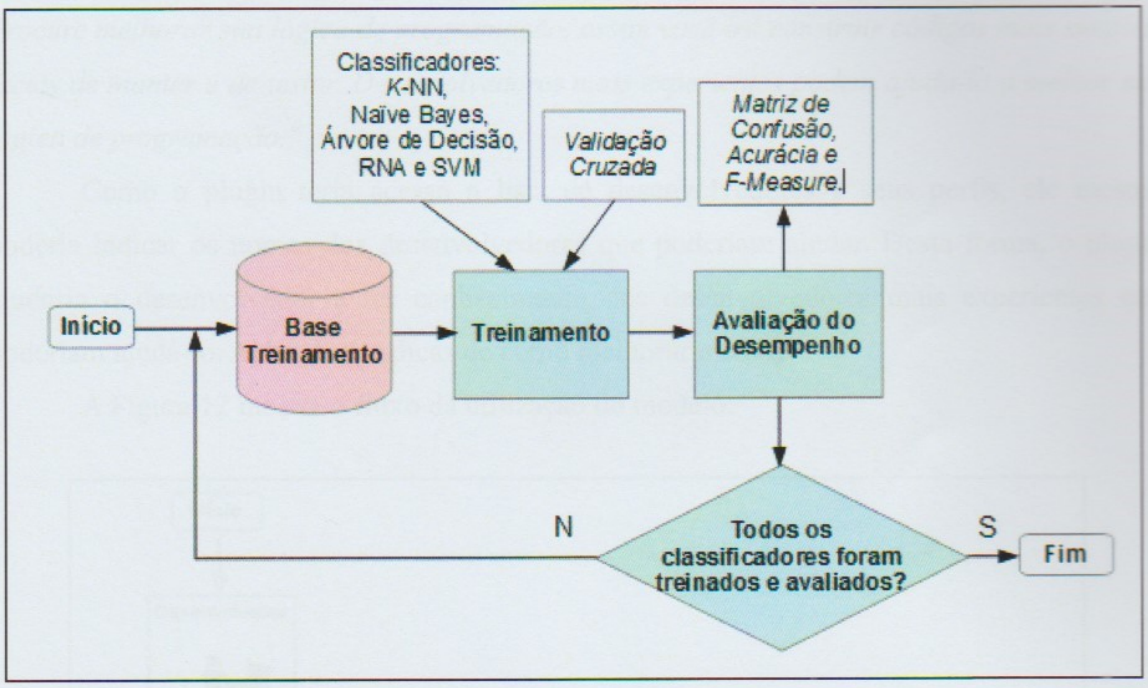


**Figura 11-** Processo de Treinamento e Avaliação de Desempenho dos Modelos (Fonte: Autor).

Após todos os classificadores serem avaliados, deve ser escolhido aquele com melhor desempenho avaliado na *Matriz de Confusão*, *Acurácia* e *F-Measure*.

Selecionado o classificador, a terceira etapa corresponde a utilização do modelo. Esta etapa é apresentada na seção seguinte.

### 4.3. Utilização do Modelo

Para utilizar o modelo (perfil dinâmico), ele poderia ser implementado em forma de um plugin para ser acoplado a um IDE. A medida que os sensores coletam novos perfis de um

desenvolvedor, que são adicionados à Base Histórica, este plugin seria aplicado sobre estas novas instâncias para classificar o desenvolvedor. Com a classificação, disponibiliza-se em algum painel adicionado ao IDE, dicas para ajudar o desenvolvedor.

Um exemplo de ajuda que o plugin poderia fornecer é: classificado um desenvolvedor com classe exemplo *Iniciante*, dar dicas de como resolver os erros de sintaxe que aparecem no editor. Um exemplo de dica seria: *"Quando seu IDE indicar uma linha de código com erro, procure ler as dicas que o próprio IDE fornece para resolvê-lo. Caso não consiga resolver, você pode buscar ajuda com um desenvolvedor mais experiente. Ele poderá ajudá-lo."* . Ou: *"Muitos comandos aninhados como IF, WHILE e FOR tornam o seu código complexo. Procure melhorar sua lógica de programação, assim você irá construir códigos mais simples, fáceis de manter e de testar. Desenvolvedores mais experientes podem ajudá-lo a melhor sua lógica de programação."*

Como o plugin teria acesso a lista de desenvolvedores e seus perfis, ele mesmo poderia indicar os nomes dos desenvolvedores que poderiam ajudar. Desta forma, o plugin ajudaria o desenvolvedor a ter conhecimento dos desenvolvedores mais experientes que poderiam ajudá-lo. Além de dar dicas de como melhorar o código.

A Figura 12 mostra o fluxo da utilização do modelo.

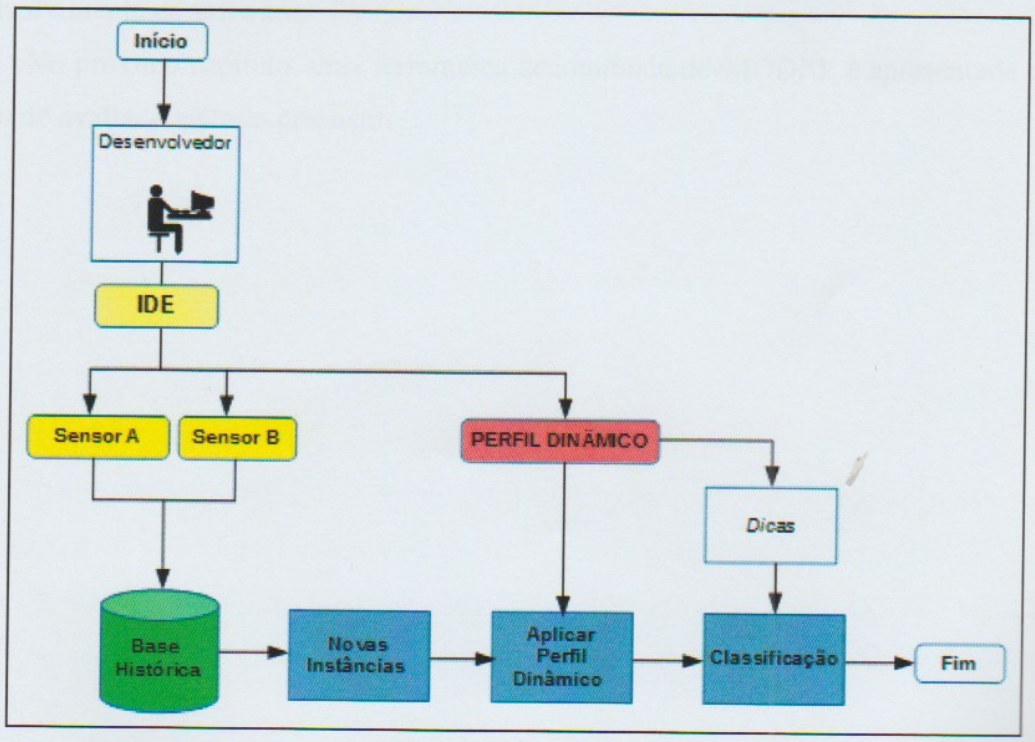


**Figura 12** - Fluxo da Utilização do Perfil Dinâmico (Fonte: Autor).

O perfil dinâmico, que poderia ser implementado na forma de plugin deve ser atualizado, ou seja, deve ser treinado periodicamente pois os desenvolvedores tendem naturalmente, como o passar do tempo, mudar de classe. Este processo é a etapa 4 do método e corresponde a Manutenção do Modelo. A Etapa 4 é apresentada na sequência.

### 4.4. Manutenção do Modelo

Com o passar do tempo e como ganho de experiência, os desenvolvedores mudam o seu perfil. Por isso, o modelo deve ser atualizado. A Etapa 2 do método deve ser repetida para que os classificadores seja treinados com dados mais recentes, e assim, gerar um novo modelo atualizado.

### 4.5. Considerações Finais

O método para a construção do perfil dinâmico do desenvolvedor foi apresentado. Acredita-se que seja possível especializá-lo para outros participantes como Arquiteto, Gerente de Projetos, Testador, entre outros. Uma das grandes dificuldades em especializar o método para outros participantes é desenvolver sensores para cada ferramenta utilizada no processo de desenvolvimento de software.

No próximo capítulo, uma ferramenta denominada devMODEL é apresentada com o intuito de avaliar o método proposto.

# Capítulo 5

# Protótipo Computacional: devMODEL

Este capítulo apresenta o desenvolvimento de um protótipo computacional para avaliar o método de construção do perfil dinâmico do desenvolvedor proposto neste trabalho. Este protótipo chama-se devMODEL. Ele é formado por um conjunto de ferramentas e cobre as etapas1 (Aquisição dos Perfis) e 2 (Modelo de Classificação) do método.

Para a coleta implícita dos dados dos desenvolvedores foi utilizado o sensor HACKYSTAT e o Plugin METRICS para o IDE ECLIPSE. Eles foram utilizados para coletar dados para as 22 características da Tabela 8. O sensor HACKYSTAT foi utilizado para coletar dados para as características *Debug*, *Breakpoint*, *Refactoring*, *codErro* e *qtdeErro*; e o METRICS para coletar dados para as demais características.

O HACKYSTAT coleta dados a cada instante de tempo e registra em uma base APACHE DERBY[5]. Ele permite uma configuração de intervalo em minutos para envio dos dados. Neste caso, o intervalo foi configurado para enviar dados a cada 2 minutos. O Plugin METRICS coleta dados sempre que ocorre uma mudança em um dos atributos monitorados (métricas de código-fonte) e armazena em arquivos XML em disco.

Para integrar os dados dos sensores, um módulo foi implementado em linguagem Java. O módulo extrai os dados da base do HACKYSTAT e dos arquivos XML do METRICS. Estes dados são armazenados em uma base MYSQL[6] chamada de Repositório dos Sensores. A razão para se criar este repositório é para facilitar a seleção dos dados para formar as

---

5 APACHE DERBY - Maiores informações em: < db.apache.org/derby/>.

6 MYSQL - Maiores Informações em: < www.mysql.com>.

instâncias da Base Histórica. Os sensores podem coletar uma série de dados sobre o desenvolvimento maior do que a necessária. Por isso, os dados colhidos devem passar por um processo de filtragem para selecionar somente os dados referente às 22 características da Tabela 8.

Os erros coletados pelo HACKYSTAT tiveram que passar por um tratamento. O HACKYSTAT coleta os erros cometidos pelos desenvolvedores em tempo de codificação, que podem ser de diferentes tipos. Para simplificar os erros resolveu-se agrupá-los. Para tal, foi convidado um especialista em linguagem de programação Java para analisar os erros e definir os grupos (tipos).

O especialista analisou os erros coletados e os agrupou em 17 grupos diferentes. A utilização de um especialista foi necessária porque não foi encontrada na literatura uma lista de grupos de erros para a linguagem Java. Uma lista com os grupos dos erros indicados pelo especialista é apresentada no Apêndice C.

Com o repositório dos sensores formado, a integração leva em consideração as informações de data, hora, minuto, projeto e desenvolvedor de cada registro do HACKYSTAT e do METRICS. Os registros são unidos levando em consideração estes campos. A união dos dados formam as instâncias da Base Histórica. A Base Histórica é armazenada em uma base MYSQL.

O módulo que integra os dados dos sensores e forma a Base Histórica possui também uma opção para gerar automaticamente a base Treinamento em formato .arff[7]. Esta base é usada na ferramenta WEKA [WEK13] para treinamento dos classificadores.

Os algoritmos treinados foram o IBK, J48, Naive Bayes, MultilayerPerceptron e LibSVM, disponíveis na ferramenta WEKA versão 3.7. Estes algoritmos foram escolhidos porque são os mais conhecidos e utilizados dentre os algoritmos de aprendizagem supervisionada disponíveis no WEKA. Para avaliar o desempenho dos modelos foram utilizadas a *Matriz de Confusão, Acurácia* e *F-Measure*.

A Figura 13 mostra o fluxo do protótipo computacional devMODEL para a aquisição dos perfis e formação da Base Histórica.

---

[7] Formato dos arquivos de dados aceitos pela ferramenta WEKA.

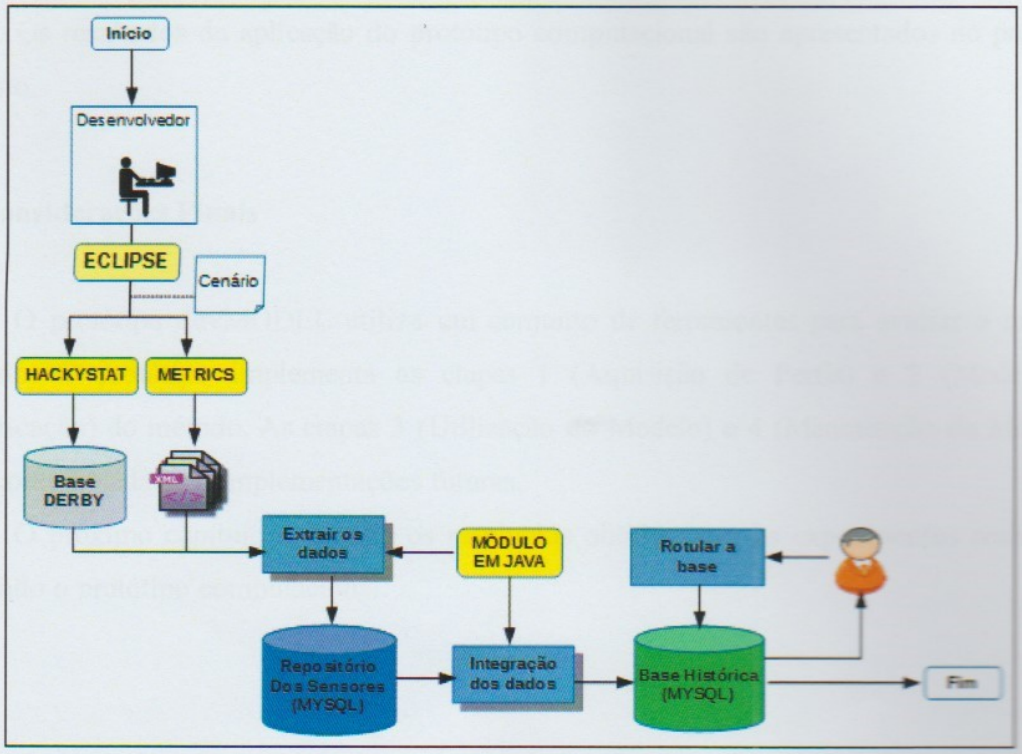


**Figura 13-** Fluxo do Protótipo para a Aquisição de Perfis (Fonte: Autor).

A Figura 14 mostra fluxo do protótipo computacional devMODEL para o Modelo de Classificação.

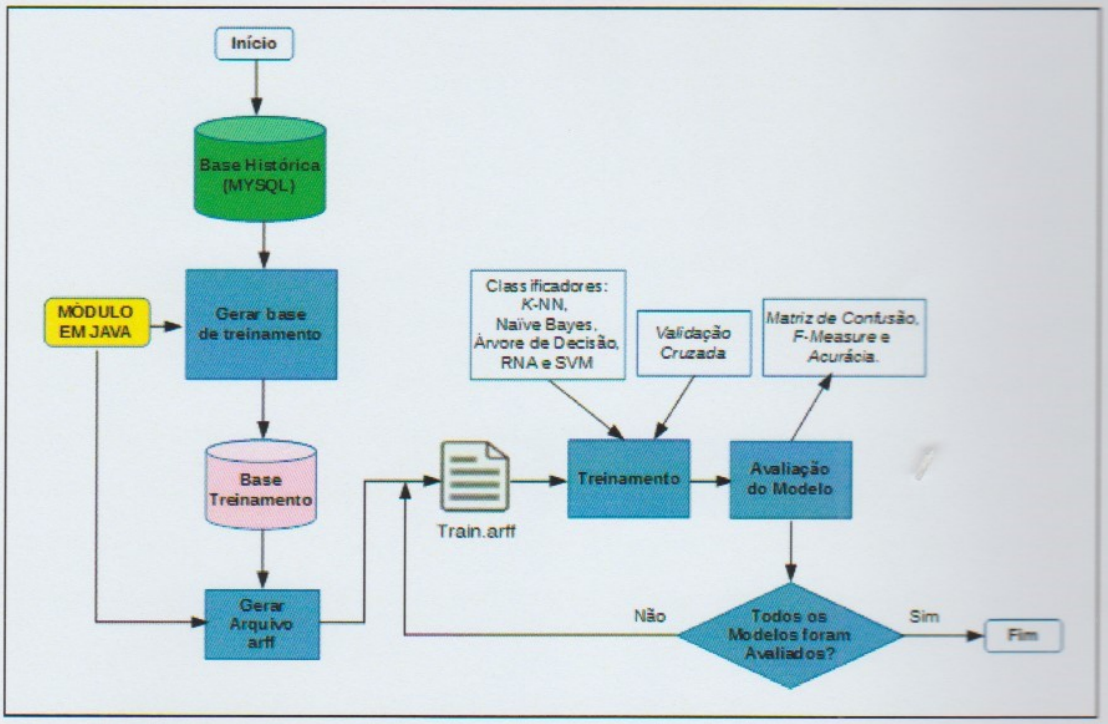


**Figura 14-** Fluxo do Protótipo para o Modelo de Classificação (Fonte: Autor).

Os resultados da aplicação do protótipo computacional são apresentados no próximo Capítulo.

## 5.1. Considerações Finais

O protótipo devMODEL utiliza um conjunto de ferramentas para avaliar o método proposto. O protótipo implementa as etapas 1 (Aquisição de Perfis) e 2 (Modelo de Classificação) do método. As etapas 3 (Utilização do Modelo) e 4 (Manutenção do Modelo) serão contempladas em implementações futuras.

O próximo capítulo apresenta os resultados obtidos com os experimentos realizados utilizando o protótipo computacional.

# Capítulo 6

# Experimentos Realizados e Análise dos Resultados

Neste capítulo são apresentados a Base Histórica formada por desenvolvedores voluntários, os resultados dos treinamentos dos classificadores utilizando o conjunto de dados Treinamento gerado a partir da Base, e uma análise dos desempenhos dos classificadores. O objetivo deste capítulo é mostrar a viabilidade da construção do perfil dinâmico do desenvolvedor e a classificação em diferentes perfis propostos nesta pesquisa.

## 6.1. Formação da Base Histórica

Para formar a Base Histórica, um grupo de desenvolvedores voluntários foi convidado. Foram feitas tentativas para coletar dados de desenvolvedores profissionais, porém, a proposta não foi aceita. As empresas contatadas receiam que parte de seus dados possam ser repassados a terceiros. Por isso, buscou-se voluntários no meio acadêmico.

Foram convidados então, 71 desenvolvedores com diferentes experiências na linguagem Java, para executar um mesmo cenário. Os participantes foram alunos do curso Bacharelado em Sistema de Informação (BSI) da Pontifícia Universidade Católica do Paraná (PUC-PR). Participaram 26 alunos matriculados na disciplina de Programação II (PROG.II), 21 alunos matriculados na disciplina de Programação IV (PROG.IV) e 24 alunos na disciplina de Programação VI (PROG.VI). A disciplina PROG.II é ministrada no segundo semestre do curso, a disciplina de PROG.IV no quarto e a PROG.VI, no sexto semestre. Todas utilizam a linguagem de programação Java.

Optou-se por três turmas de alunos porque o mercado de trabalho geralmente trabalha com três níveis diferentes como: JÚNIOR, PLENO e SÊNIOR.

Os dados foram coletados no final do último semestre de 2013, em datas diferentes para cada grupo de voluntários. O primeiro grupo a participar foi o dos alunos da disciplina PROG.II, na sequência o grupo da turma PROG.IV e, por último, o grupo da PROG.VI. Os voluntários utilizaram um laboratório da Escola Politécnica da PUC-PR que foi previamente preparado para a execução do cenário intitulado MyBMI que encontra-se no Apêndice D. Os três grupos tiveram o mesmo tempo para a execução do cenário, que foi de 01h30min.

Para a coleta de dados, os desenvolvedores utilizaram o IDE ECLIPSE. A escolha pelo ECLIPSE se deve ao fato que é um IDE de código aberto muito utilizado pela comunidade de desenvolvedores Java, tanto no meio acadêmico como no profissional. Ele também permite a ligação com os sensores de coleta de dados HACKYSTAT e METRICS.

Durante o processo de formação da Base Histórica, foi realizado o pré-processamento dos dados e foi constatado que, por algum motivo técnico para alguns desenvolvedores, um dos sensores falhou e não realizou a coleta. Neste caso foram 7 desenvolvedores, sendo 2 desenvolvedores da turma de PROG.IV e 5 da turma de PROG.VI. Os 7 foram desconsiderados. Além disso, registros duplicados ou com dados zerados foram desconsiderados também.

Após obter a Base Histórica, os registros dos voluntários foram rotulados por um especialista que utilizou como critério a turma de programação na qual os voluntários estavam matriculados. Os registros dos alunos da PROG.II foram rotulados com classe P2000, os registros dos alunos da turma PROG.IV como P4000 e os registros dos alunos da PROG.VI, como P6000. A classe P2000 representa os alunos pouco experientes, com até 1 ano de experiência na linguagem Java. A classe P4000 concentra alunos com 1 a 2 anos de experiência na linguagem. A categoria P6000 concentra os alunos mais experientes, com conhecimento na linguagem Java de 3 a 4 anos.

### 6.2. Formação do Conjunto de Dados para Treinamento dos Classificadores

Após a rotulação da Base Histórica, os registros foram utilizados para formar uma base de treinamento. Dos 64 desenvolvedores (26 da classe P2000, 19 da classe P4000 e 19 da classe P6000) que formam a Base Histórica, foram selecionados 57 desenvolvedores, sendo 19 desenvolvedores de cada uma das classes. Como na classe P2000 existiam 26 desenvolvedores, foram selecionados aleatoriamente 19. A base de Treinamento resultante é formada por 2191 registros.

Para o treinamento dos algoritmos de classificação, foi gerado um arquivo em formato *arff* a partir da base Treinamento. A Tabela 12 mostra a distribuição dos registros por classe para o conjunto de treinamento (*train.arff*).

**Tabela 12**- Distribuição dos Registros por Classe - Conjunto de Treinamento (Fonte: Autor).

| Classe | Nº de Registros |
|---|---|
| P2000 | 665 |
| P4000 | 847 |
| P6000 | 679 |
| **Total** | **2191** |

Apesar do número de desenvolvedores ser o mesmo para cada classe, os desenvolvedores geraram números de registros diferentes entre eles. Alguns produziram mais registros que outros. Isto depende muito de como eles desenvolvem. Foi observado que alguns desenvolvedores leram todo o cenário e depois começaram a implementação, já outros, a medida que faziam a leitura, implementavam os requisitos do cenário. A velocidade de codificação variou de um desenvolvedor para outro.

O conjunto de dados *train.arff* foi utilizado para treinar todos os classificadores indicados no método. Os resultados dos treinamentos são apresentados na próxima seção.

### 6.3. Treinamento dos Classificadores e Avaliação do Desempenho

Para avaliar o desempenho dos classificadores no treinamento foi utilizado o método de Validação Cruzada com *k* partes igual a 10.

Os classificadores possuem um conjunto de parâmetros que podem ser ajustados pelo usuário. A ferramenta WEKA traz valores *default* para os parâmetros de cada classificador. Para o treinamento foram mantidos os valores *default*. Com exceção do parâmetro *KNN* (número de vizinhos) do IBK que foi alterado para 3 e o parâmetro *unpruned*(poda) do J48 que foi modificado para *false*.

As próximas tabelas mostram os resultados do treinamento dos classificadores. Os dados sobre *Acurácia* e *F-Measure* foram obtidos do WEKA. A Tabela 13 mostra a *Acurácia* de cada um dos modelos. A Tabela 14 apresenta a medida *F-Measure* para cada classe e, na última coluna, a média ponderada da *F-Measure* das 3 classes.

**Tabela 13**- *Acurácia* dos Modelos (Fonte: Autor).

| Classificador | Acurácia |
|---|---|
| J48 | 90,96% |
| MLP | 86,44% |
| IBK (*KNN=3*) | 85,08% |
| SVM | 79,37% |
| Naive Bayes | 65,91% |

**Tabela 14**- *F-Measure* dos Modelos (Fonte: Autor).

| Classificador | F-Measure | | | |
|---|---|---|---|---|
| | P2000 | P4000 | P6000 | Média Pond. |
| J48 | 0,899 | 0,934 | 0,890 | 0,910 |
| MLP | 0,857 | 0,879 | 0,854 | 0,865 |
| IBK (*KNN=3*) | 0,838 | 0,872 | 0,837 | 0,851 |
| SVM | 0,806 | 0,785 | 0,791 | 0,793 |
| Naive Bayes | 0,726 | 0,559 | 0,666 | 0,634 |

Na Tabela 13 pode-se observar o modelo J48 obteve melhor *Acurácia* de 90,96% e com melhor *F-Measure* igual 0,910 (Tabela 14) , em relação aos outros. Seguido pelo MLP com *Acurácia* = 86,44% e *F-Measure* = 0,865 (Tabela 14) e IBK com *Acurácia* = 85,08% e *F-Measure* = 0,851 (Tabela 14). O SVM e o Naive Bayes tiveram desempenhos inferiores.

Analisando as matrizes de confusão dos três classificadores com melhor desempenho (J48, MLP e IBK) na Tabela 15, percebe-se que a classe P2000 apresenta o menor erro de classificação. Já as classes P4000 e P6000 apresentam maior confusão.

As confusões observadas na classe P4000 podem ter ocorrido porque existem alguns alunos que reprovaram na disciplina PROG.II mas estavam cursando a disciplina PROG.IV. O que pode justificar as classificações incorretas da classe P4000 como P2000. Da mesma forma ocorre com os alunos da PROG.VI, onde existem alguns alunos que reprovaram na disciplina PROG.IV e estão cursando a PROG.VI, o que pode justificar as classificações incorretas da classe P6000 como P4000.

Em relação às confusões da classe P4000 onde registros foram classificados como sendo classe P6000, elas podem ter ocorrido porque existem alguns alunos que já possuem uma experiência um pouco mais avançada na linguagem Java, alguns fazem iniciação científica onde trabalham em projetos que utilizam a linguagem Java e outros trabalham profissionalmente com desenvolvimento. A turma PROG.II também possui alguns alunos com

experiência em desenvolvimento, inclusive alguns já atuam como estagiários, o que pode justificar as classificações incorretas da classe P2000 como classe P6000.

**Tabela 15**- Matrizes de Confusão - J48, MLP e IBK (Fonte: Autor).

| J48 | MLP | IBK (*KNN=3*) |
|---|---|---|
| === Confusion Matrix ===<br><br>a b c <-- classified as<br>614 16 35 \| a = P2000<br>34 780 33 \| b = P4000<br>53 27 599 \| c = P6000 | === Confusion Matrix ===<br><br>a b c <-- classified as<br>601 33 31 \| a = P2000<br>71 728 48 \| b = P4000<br>65 49 565 \| c = P6000 | === Confusion Matrix ===<br><br>a b c <-- classified as<br>580 50 35 \| a = P2000<br>62 732 53 \| b = P4000<br>78 49 552 \| c = P6000 |

Esta seção apresentou os resultados dos treinamentos dos classificadores utilizados uma base de dados que foi obtida a partir de um cenário executado durante 01h30min. Em um novo experimento buscou-se avaliar a viabilidade da aplicação de outros cenários, com tempos maiores de duração. Este experimento é descrito na seção seguinte.

## 6.4. Avaliação da Viabilidade da Aplicação de Novos Cenários

Para avaliar a viabilidade da aplicação de novos cenários com tempo de duração maiores, foi realizado um experimento onde, o conjunto de treinamento foi inicialmente ordenado. A ordenação se deu da seguinte forma: os registros de cada desenvolvedor, foram ordenados por data, hora e minuto. Isto foi possível porque o conjunto de treinamento traz os dados do desenvolvedor, data, hora e minuto em cada um dos registros, embora estes atributos não participem do treinamento.

Após a ordenação, o conjunto foi dividido em 3 subconjuntos de treinamento. O primeiro subconjunto possui os registros iniciais, aqueles que foram coletados nos primeiros minutos da execução do cenário. O segundo subconjunto possui os registros intermediários e o terceiro o registros gerados no final do experimento.

De posse dos subconjuntos de dados, eles foram submetidos ao grupo de classificadores candidatos. No treinamento foi utilizado *Validação Cruzada* $k = 10$. Foram mantidos os mesmo parâmetros de cada classificador e os resultados são apresentados a seguir (Tabela 16, Tabela 17, Tabela 18, Tabela 19, Tabela 20 e Tabela 21):

**Tabela 16-** *Acurácia* Subconjunto Parte 1 (Fonte: Autor).

| Parte 1 - Subconjunto dos Registros Iniciais | |
|---|---|
| Classificador | Acurácia |
| J48 | 77,07% |
| MLP | 71,17% |
| IBK (*KNN=3*) | 65,82% |
| SVM | 65,40% |
| Naive Bayes | 47,96% |

**Tabela 17-** *Acurácia* Subconjunto Parte 2 (Fonte: Autor).

| Parte 2 - Subconjunto dos Registros Intermediários | |
|---|---|
| Classificador | Acurácia |
| J48 | 96,77% |
| MLP | 94,94% |
| IBK (*KNN=3*) | 90,86% |
| SVM | 82,42% |
| Naive Bayes | 73,42% |

**Tabela 18-** *Acurácia* Subconjunto Parte 3 (Fonte: Autor).

| Parte 3 - Subconjunto dos Registros Finais | |
|---|---|
| Classificador | Acurácia |
| J48 | 97,01% |
| MLP | 96,75% |
| IBK (*KNN=3*) | 93,76% |
| SVM | 86,35% |
| Naive Bayes | 73,73% |

**Tabela 19-** *F-Measure* Subconjunto Parte 1 (Fonte: Autor).

| Parte 1 - Subconjunto dos Registros Iniciais | | | | |
|---|---|---|---|---|
| Classificador | F-Measure | | | |
| | P2000 | P4000 | P6000 | Média Pond. |
| J48 | 0,752 | 0,806 | 0,751 | 0,772 |
| MLP | 0,704 | 0,731 | 0,697 | 0,712 |
| IBK (*KNN=3*) | 0,638 | 0,720 | 0,605 | 0,659 |
| SVM | 0,660 | 0,669 | 0,628 | 0,654 |
| Naive Bayes | 0,582 | 0,347 | 0,481 | 0,459 |

**Tabela 20-** *F-Measure* Subconjunto Parte 2 (Fonte: Autor).

| Parte 2 - Subconjunto dos Registros Intermediários | | | | |
|---|---|---|---|---|
| Classificador | F-Measure | | | |
| | P2000 | P4000 | P6000 | Média Pond. |
| J48 | 0,981 | 0,968 | 0,954 | 0,968 |
| MLP | 0,956 | 0,939 | 0,957 | 0,949 |
| IBK (*KNN=3*) | 0,905 | 0,907 | 0,915 | 0,909 |
| SVM | 0,858 | 0,820 | 0,795 | 0,824 |
| Naive Bayes | 0,746 | 0,694 | 0,770 | 0,733 |

**Tabela 21-** *F-Measure* Subconjunto Parte 3 (Fonte: Autor).

| Parte 3 - Subconjunto dos Registros Finais | | | | |
|---|---|---|---|---|
| Classificador | F-Measure | | | |
| | P2000 | P4000 | P6000 | Média Pond. |
| J48 | 0,974 | 0,970 | 0,966 | 0,970 |
| MLP | 0,977 | 0,964 | 0,962 | 0,967 |
| IBK (*KNN=3*) | 0,925 | 0,950 | 0,935 | 0,938 |
| SVM | 0,873 | 0,843 | 0,878 | 0,863 |
| Naive Bayes | 0,817 | 0,637 | 0,740 | 0,724 |

Podemos observar nas tabelas de resultados (Tabela 16, Tabela 17, Tabela 18, Tabela 19, Tabela 20 e Tabela 21) que tanto a *Acurácia* quanto a *F-Measure* tiveram uma melhora crescente do conjunto Parte 1 para o conjunto Parte 3. Tomando como exemplo o classificador J48, que obteve o melhor desempenho, esta melhora pode ser visualizada nos gráficos abaixo (**Gráfico *1*** e Gráfico 2) que mostram *Acurácia* e a *F-Measure* do classificador.

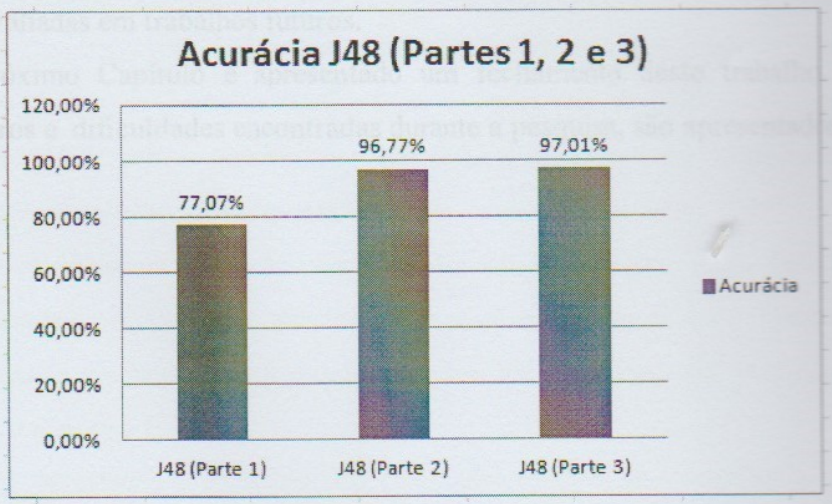


**Gráfico 1-** Resultado da *Acurácia* do J48 - Partes 1, 2 e 3 (Fonte: Autor).

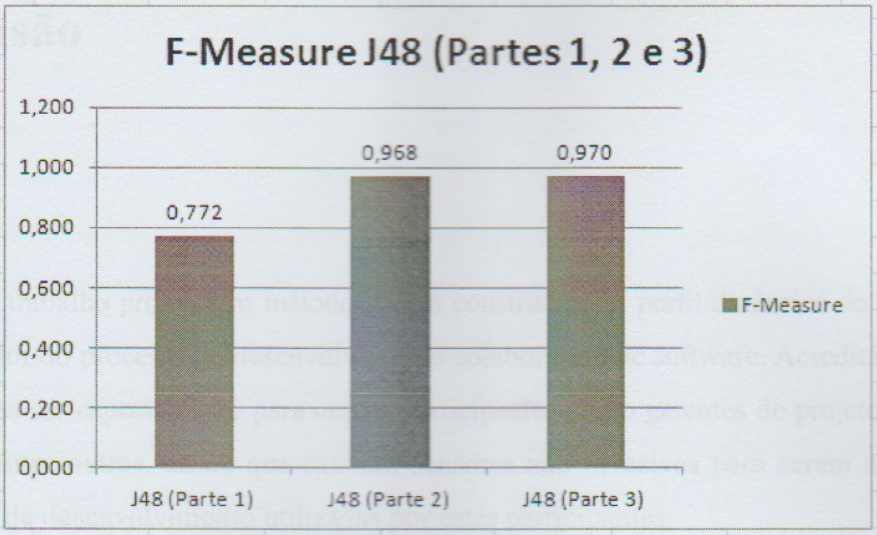


**Gráfico 2-** Resultado da *F-Measure* do J48 - Partes 1, 2 e 3 (Fonte: Autor).

Analisando a melhora crescente do classificador J48, podemos considerar que há indícios que o desempenho do classificador pode melhorar com uma coleta de dados em tempo mais longo. Assim, pode-se dizer que a aplicação de novos cenários com tempo de duração maior que 01h30min é viável.

### 6.5. Considerações Finais

Neste Capítulo foram apresentados os resultados obtidos com a aplicação do protótipo computacional DevMODEL. O objetivo do protótipo é avaliar as etapas 1 e 2 do método proposto que são Aquisição do Perfil e Modelo de Classificação. As demais etapas do método deverão ser avaliadas em trabalhos futuros.

No próximo Capítulo é apresentado um fechamento deste trabalho. Conclusões, trabalhos futuros e dificuldades encontradas durante a pesquisa, são apresentados.

# Conclusão

Este trabalho propôs um método para a construção do perfil dinâmico do participante desenvolvedor do processo de desenvolvimento colaborativo de software. Acredita-se que este método possa ser especializado para outros participantes como gerentes de projetos, analistas, arquitetos, entre outros, desde que existam sensores não invasivos para serem acoplados às ferramentas de desenvolvimento utilizadas por estes participantes.

Para a composição do perfil dinâmico proposto pelo método, foram utilizadas como características do desenvolvedor: métricas de complexidade (WMC, LCOM* e NBD), métricas de herança (DIT, NOC, NORM e SIX), métricas de tamanho (MLOC, NAC, NSF, NSM, PAR, NOI e NOP), métricas de acoplamento (I, A e Dn), utilização de *Debug*, utilização de *Breakpoint*, utilização de *Refactoring*, códigos de erros cometidos e quantidades.

A motivação maior para a escolha deste conjunto de características partiu da ideia que a complexidade e manutenibilidade de software são dois fatores importantes que influenciam diretamente a qualidade do código-fonte, e que estes dois fatores tem origem nos desenvolvedores. Os desenvolvedores possuem diferentes características que determinam a qualidade do código-fonte produzido por eles. Estas características podem ser obtidas do código-fonte e representadas em um perfil dinâmico do desenvolvedor.

Uma ferramenta chamada devMODEL foi apresentada para avaliar o método proposto. Ela utiliza as características indicadas pelo método para a composição dos perfis e os representa em perfis em formato XML. Para a aquisição dos perfis utiliza os sensores HACKYSTAT e METRICS ligados ao IDE ECLIPSE. Constrói uma base com dados históricos sobre o trabalho dos desenvolvedores, chamada de Base Histórica. Esta base é utilizada para construir modelos de classificação. Os algoritmos IBK, J48, Naive Bayes, Multilayer Perceptron e LibSVM são treinados com os dados da base histórica utilizando *Validação Cruzada*. Os resultados dos desempenhos dos algoritmos são avaliados pela *Matriz de Confusão, Acurácia* e *F-Measure*.

A partir de estudos e experimentos realizados chegou-se a conclusão que é possível construir o perfil dinâmico do desenvolvedor com dados obtidos do código-fonte produzido

por ele, por meio de sensores ligadas aos IDEs utilizados nas atividades de programação. Os classificadores que foram treinados com um conjunto de dados obtidos pelos sensores, apresentaram resultados promissores que indicam possibilidade de classificação dos desenvolvedores em três classes diferentes.

O classificador que apresentou o melhor desempenho e que melhor se adaptou aos dados foi uma Árvore de Decisão (J48), seguida de uma Rede Neural Artificial do tipo Multilayer Perceptron (MLP), do IBK e do SVM com resultados bem próximos. Um experimento secundário mostrou que é viável a coleta de dados por mais tempo para aumentar o conjunto de dados de treinamento para melhorar o desempenho dos classificadores.

Este método pode orientar o processo de construção do perfil dinâmico do desenvolvedor de software. Além disso, a classificação do desenvolvedor fornecida pelo sistema pode orientar tanto a gerência de projetos quanto as equipes de desenvolvimento na identificação dos perfis de seus desenvolvedores e assim fornecer treinamentos e suportes. Dar dicas com base na classificação para ajudar os desenvolvedores a melhorar suas características. Orientar a formação de equipes de duplas de programação. Finalmente, que a base histórica gerada pelos experimentos possa ser disponibilizada e aproveitada por outros pesquisadores da área.

**Trabalhos Futuros**

Como trabalho futuro, pretende-se estender os sensores existentes e utilizar novos sensores, para coletar outros tipos de dados sobre os desenvolvedores, para assim ampliar a lista de características do perfil dinâmico. Pretende-se também pesquisar sensores de coletas de dados para outros participantes do processo de desenvolvimento colaborativo de software como gerentes de projeto, arquitetos, testadores, entre outros.

Outra possibilidade consistem em aplicar novos cenários com tempo de duração maior para coletar uma quantidade maior e mais significativa de dados também é um trabalho futuro a ser realizado.

Além disto, para melhorar o desempenho dos classificadores, pode-se avaliar diferentes valores de parâmetros, fazer seleção de atributos e aplicar técnicas de Normalização e Discretização de dados.

As etapas Utilização e Manutenção do Modelo não foram avaliadas neste trabalho, portanto deve-se estender o protótipo devMODEL para que estas etapas possam ser avaliadas.

Em relação a etapa de Utilização do Modelo, realizar estudos e experimentos para investigar o momento certo para chamar o modelo em uma utilização real. Por exemplo, quando um desenvolvedor está realizando suas atividades de programação e o modelo está acoplado ao seu IDE, após quanto tempo o modelo pode classificar o desenvolvedor? Após atingir um determinado número de registros coletados? Ou após 30 minutos? Experimentos deverão ser realizados para identificar qual o melhor momento para executar o modelo para realizar a classificação.

Sobre a manutenção do modelo, com o passar do tempo os desenvolvedores mudam suas características sendo necessário treinar novamente o modelo com os novos dados. Neste sentido, estudos deverão ser realizados para encontrar um mecanismo que identifique o momento certo que o modelo deve ser treinado novamente.

Implementar o perfil dinâmico do desenvolvedor em formato de um plugin para IDE ECLIPSE, para fazer classificação em tempo real é outra etapa futura da pesquisa. Este plugin deve coletar os dados durante a atividade do desenvolvedor, classificar o desenvolvedor em um perfil e fornecer algum tipo de suporte ou assistência para ajudá-lo a melhorar e desenvolver as suas características.

**Dificuldades Encontradas Durante a Pesquisa**

Uma das grandes dificuldades foi a participação voluntária de desenvolvedores para a formação da Base Histórica de dados. Inicialmente a ideia do projeto era trabalhar com desenvolvedores profissionais e, então, classificá-los nas categorias JUNIOR, PLENO e SÊNIOR. Porém, não foi possível obter um número suficiente de voluntários nesta configuração. Quando se buscou a participação das empresas para permitir a coleta dentro de seus ambientes de desenvolvimento com os seus colaboradores, elas ficaram receosas por causa da coleta dos dados e preferiram não participar. Em função desta dificuldade buscou-se voluntários no meio acadêmico.

Como se tratou de uma pesquisa que envolveu a participação de seres humanos (os desenvolvedores) houve uma preocupação em preservar a privacidade dos voluntários. Foi tomado todo o cuidado para não disponibilizar qualquer tipo de informação que pudesse identificar qualquer voluntário. Desta forma, tiveram que ser utilizadas denominações fictícias para identificá-los. O projeto foi aprovado pelo Comitê de Ética da PUCPR (documentos em

Apêndice E).

Outra dificuldade foi em relação às tecnologias de sensores de coleta de dados não invasiva. Como a proposta é um método para construir um perfil dinâmico se fez necessário o uso deste tipo de ferramenta. Apenas sensores para a coleta de dados sobre o desenvolvimento de software são encontrados com mais facilidade. Uma das ferramentas utilizadas neste trabalho, o METRICS, teve que ser estendido para fazer a coleta e armazenamento dos dados automaticamente.

# Referências Bibliográficas

[AIK04] AIKEN, J. Technical and human perspectives on pair programming. SIGSOFT Software Engineering Notes, v. 29, n.5, p.1–14, Sept. 2004.

[BAR10] BARTH, F. J. Modelando o perfil do usuário para a construção de sistemas de recomendação: um estudo teórico e estado da arte. Revista de Sistemas de Informação, v.6, p.59-71, 2010.

[BIS11] BISWAS, P.; ROBINSON, P. A brief survey on user modelling in HCI. In *Intelligent Techniques for Speech, Image and Language Processing*. 2011.

[BRU09] BRUEGGE, B.; DAVID, J.; HELMING, J.; KOEGEL, M. Classification of tasks using machine learning. In *Proceedings of the 5th International Conference on Predictor Models in Software Engineering* (PROMISE '09). ACM, New York, NY, USA, Article 12, 11 pages, 2009.

[BRU04] BRUNTINK, M.; van DEURSEN, A. Predicting class testability using object-oriented metrics. Source Code Analysis and Manipulation, 2004. Fourth IEEE International Workshop on, p.136-145, 15-16 Sept. 2004.

[CHE11] CHEN, N.; HOI, S.C.H.; XIAO, X. Software process evaluation: A machine learning approach. *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on* , p.333,342, 6-10 Nov. 2011.

[CHE91] CHEN, Q.; NORCIO, A.F. A neural network approach for user modeling. *Systems, Man, and Cybernetics, 1991. 'Decision Aiding for Complex Systems, Conference Proceedings., 1991 IEEE International Conference on* , vol.2, p.1429-1434, 1991.

[CHI91] CHIDAMBER, R. S.; KEMERER, C. F. Towards a metrics suite for object-oriented design. In: Proceedings of the OOPSLA 91 Conference, p. 197-211, 1991.

[CLO03] CLOCKSIN, W. F.; MELLISH, C. S. Programming in PROLOG. 5Th ed, Springer, 2003.

[COH60] COHEN, J. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20, 37-46, 1960.

[COR10] CORRÊA, U.B.; LAMB, L.; CARRO, L.; BRISOLARA, L.; MATTOS, J., Towards Estimating Physical Properties of Embedded Systems using Software Quality Metrics. Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on , p.2381,2386, June 29 2010-July 1 2010.

[COR95] CORTES, C; VAPNIK, V. Support Vector Networks. Machine Learning, 1995.

[DUD02] DUDA, R.; HART, P.; STORK, D. Pattern Classification. 2ed. Willey Interscience, 2002.

[FAC11] FACELI, K.; LORENA, A.C.; GAMA, J.; De CARVALHO, A. C. P. L. F. Inteligência Artificial: Uma abordagem de Aprendizagem de Máquina. Rio de Janeiro: LCT, 2011.

[FLE81] FLEISS, J. Statistical methods for rates and proportions. 2 ed.. New York: John Wiley & Sons, 1981.

[FON07] FONSECA, R. SILVA, P. SILVA, R. Acordo inter-juízes: O caso do Coeficiente Kappa. Laboratório de Psicologia, I.S.P.A. 5(1): 81-90, 2007.

[FRE09] FREDDO, A. R.; TACLA, C. A.; SATO, G. Y.; PARAISO, E. C.; CAMPAGNOLO, B.; RAMOS, M. P. Supporting Small Teams in Cooperative Software Development with a Multi-agent Architecture. SBSC, pp.178-183, 2009 Simpósio Brasileiro de Sistemas Colaborativos, 2009.

[GAL13] GALETE, L.; RAMOS, M. P.; NIEVOLA, J.C.; PARAISO, E.C. Dynamically modeling users with MODUS-SD and Kohonen's map, Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on , pp.17,22, 27-29 June 2013.

[GRU94] GRUDIN, J. Computer-supported cooperative work: history and focus. *Computer*, vol.27, no.5, p.19,26, May 1994.

[HAC12a] HACKYSTAT. Disponível em: <http://code.google.com/p/hackystat/>. Acesso em: 24

jun. 2012.

[HAC12b] HACKYSTAT SensorDataTypeSpecification. Disponível em: <http://code.google.com/p/hackystat/wiki/sensordatatypespecifications>. Acesso Em: 10 set. 2012.

[HAY01] HAYKIN, S. Redes Neurais: princípios e práticas. 2 ed. Porto Alegre: Bookman, 2001.

[HEN96] HENDERSON-SELLERS, Brian. Object-Oriented Metrics: Measures of Complexity. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[HON09] HONGLEI, T.; WEI, S.; YANAN, Z.. The Research on Software Metrics and Software Complexity Metrics. *Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on*, p.131,136, 25-27 Dec. 2009.

[HOR04] HORSTMANN, Cay. Big Java. PortoAlegre: Bookman, 2004.

[IEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, pp.1, 1990.

[JOH03] JOHNSON, P. M.; KOU, H; AGUSTIN, J.; CHAN, C.; MOORE, C.; MIGLANI, J.; ZHEN, S; DOANE, W. E. J. Beyond the Personal Software Process: metrics collection and analysis for the differently disciplined. *In Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. IEEE Computer Society, Washington, DC, USA, p. 641-646, 2003.

[JON13] JONSSON, L., Increasing anomaly handling efficiency in large organizations using applied machine learning. Software Engineering (ICSE), 2013 35th International Conference on , p.1361,1364, 18-26 May 2013.

[KAV00] KAVCIC, A., The role of user models in adaptive hypermedia systems. *Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean* , vol.1, p.119,122, 2000.

[LIU07] LIU, K.; CHEN, W.; BU, J.; CHEN, C.; ZHANG, L. User Modeling for Recommendation in Blogspace. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web*

*Intelligence and Intelligent Agent Technology - Workshops* (WI-IATW '07). IEEE Computer Society, Washington, DC, USA, p.79-82, 2007.

[LOR03] LORENA, A. C.; CARVALHO, A. C. P. L. F. de. Introdução às Máquinas de Vetores Suporte (Support Vector Machines). Technical Report nº 192. Instituto de Ciências Matemáticas e de Computação da USP, 2003.

[MAR94] MARTIN, R. OO Design Quality Metrics: An Analysis of Dependencies. October 28, 1994. Disponível em: <http://www.objectmentor.com/resources/articles/oodmetrc.pdf>. Acesso em: 28/08/2013.

[MAR02] MARTIN, R. C. Agile Software Development: Principles, Patterns, and Practices. Prentice Hall, 2002.

[MCC76] MCCABE, T., A Complexity Measure. *Software Engineering, IEEE Transactions on*, vol.SE-2, no.4, p. 308- 320, Dec. 1976.

[MET13] METRICS. Disponível em: < http://metrics2.sourceforge.net/ >. Acesso em: 28/08/2013.

[MUJ05] MUJEEB-U-REHMAN, M.; YANG, X; DONG, J.; GHAFOOR, M. A. Heterogeneous and homogenous pairs in pair programming: an empirical analysis. *Electrical and Computer Engineering, 2005. Canadian Conference on*, p.1116,1119, 1-4 May 2005.

[MIT97] MITCHELL T. Machine Learning. WCB McGraw–Hill, 1997.

[NGU09] NGUYEN, L. A Proposal of Discovering User Interest by Support Vector Machine and Decision Tree on Document Classification. *Computational Science and Engineering, 2009. CSE '09. International Conference on* , vol.4, p.809,814, 29-31 Aug. 2009.

[PAP01] PAPATHEODOROU, C. Machine Learning in User Modeling. In: *Machine Learning and Its Applications, Advanced Lectures*. Springer-Verlag, London, UK, UK, p. 286-294, 2001.

[PAD03] PADBERG, F.; MULLER, M.M. Analyzing the cost and benefit of pair programming, Software Metrics Symposium, 2003. Proceedings. Ninth International, p. 166- 177, 3-5 Sept. 2003.

[PLO10] PLOSCH, R.; GRUBER, H.; KÖRNER, C.; SAFT, M., A Method for Continuous Code Quality Management Using Static Analysis. *Quality of Information and Communications Technology (QUATIC), 2010* Seventh International Conference on the, p.370,375, Sept. 29 2010-Oct. 2 2010.

[PRE02] PRESSMAN, R.S. Engenharia de Software. 5th ed. Rio de Janeiro: McGraw-Hill, 2002.

[PRE10] PRESSMAN, R.S. Software engineering: a practitioner's approach. 7th ed. McGraw-Hill Higher Education, 2010.

[REZ05] REZENDE, S. O. Sistemas Inteligentes: fundamentos e aplicações. Barueri, SP: Manole, 2005.

[RIC83] RICH, E. Users are individuals- Individualizing user models. *International Juornal of Man-Machine Studies*, vol. 18, p. 199-214, 1983.

[ROS99] ROSENBERG, L.; RUTH, D.; GALLO, S. A. Risk-based Object Oriented Testing. In: Proceedings of the 24 th annual Software Engineering Workshop, NASA, Software Engineering Laboratory, 1999. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.7509&rep=rep1&type=pdf>, Acesso em: 26/08/2013.

[SCH02] SCHWAB, I.; KOBSA, A. Adaptivity through Unobstrusive Learning. Special Issue on Adaptivity and User Modeling. KI, vol. 16, no. 3, p. 5-9, 2002.

[SIL03] SILLITTI, A.; JANES, A.; SUCCI, G.; VERNAZZA, T. Collecting, integrating and analyzing software metrics and personal software process data. *Euromicro Conference, 2003. Proceedings. 29Th* , p.336,342, 1-6 Sept. 2003.

[SOL10] SOLIMAN, T.H.A.; EL-SWESY, A.; AHMED, S.H. Utilizing CK metrics suite to UML models: A case study of Microarray MIDAS software. *Informatics and Systems (INFOS), 2010 The 7th International Conference on* , p.1,6, 28-30 March 2010.

[SOM11] SOMMERVILLE, I. Engenharia de software. São Paulo: Pearson Prentice Hall, 2011.

[SON13a] SONAR. Disponível em: <http://www.sonarsource.org/>. Acesso em: 12/04/2013.

[SON13b] SONAR. Documentação do Sonar. Disponível em: <http://docs.codehaus.org/display/SONAR/Metric+definitions>. Acesso em: 12/04/2013.

[SOU12] SOUZA, J.P. De; LUGO, G.A.G.; TACLA, C.A. Inferring activities of an actor by means of context ontologies. *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En* , p.1,10, 1-5 Oct. 2012.

[TAN09] TAN, Pang-Ning; STEINBACH, Michael; KUMAR, Vipin. Introdução ao DATAMINING Mineração de Dados. Rio de Janeiro: Editora Ciência Moderna Ltda., 2009.

[USE13] USER GUIDE ECLIPSE SENSOR. Disponível em: <https://code.google.com/p/hackystat-sensor-eclipse/wiki/UserGuide>. Acesso em: 18/12/2013.

[WAN12] WANDERLEY, G. M. P.; RAMOS, M. P.; TACLA, C. A.; SATO, G. Y.; da SILVA, E. J.; PARAISO, E. C. MODUS-SD: User modeling in collaborative software development. CSCWD 2012: p.372-378, 2012.

[WIT05] WITTEN, I.H.; FRANK, E. Data mining: practical machine learning tools and techniques. 2 ed, Morgan Kaufmann, 2005.

[WEB02] WEBB, A. Statistical Pattern Recognition. 2nd edition, John Wiley & Sons, Inc., 2002.

[WEN08] WEN, H.; FANG, L.; GUAN, L. Modelling an individual's Web search interests by utilizing navigational data. *Multimedia Signal Processing, 2008 IEEE 10th Workshop on* , p.691,695, 8-10 Oct. 2008.

[WEK13] WEKA. Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/documentation.html>. Acesso em: 09 abr. 2013.

[YU10] YU, S.; ZHOU, S. A survey on metric of software complexity. *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on* , p.352,356, 16-18 April 2010.

[DU02] Du Zhang; Tsai, J.J.-P. Machine learning and software engineering, *Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings. 14th IEEE International Conference on* , pp.22,29, 2002.

[ZHO11] ZHOU, B.; ZHANG, B.; LIU, Y.; XING, K. User Model Evolution Algorithm: Forgetting and Reenergizing User Preference. *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, p.444,447, 19-22 Oct. 2011.

# Apêndice A

# Exemplos de Eventos Coletados pelo Sensor HACKYSTAT

Neste apêndice, a Tabela 2 (Capítulo 2) é reapresentada (Tabela 22) com exemplos para cada um dos eventos coletados pelo HACKYSTAT SENSOR ECLIPSE de um arquivo Java ativo.

**Tabela 22**- Exemplos de Eventos Coletados pelo HACKYSTAT SENSOR ECLIPSE (Adaptado de [USE13]).

| Tool | SensorDataType | Type | Subtype | Descrição |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | OpenFile | Gerado quando um novo arquivo é aberto. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T18:25:37.716-03:00 | 2013-10-11T18:25:37.716-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Subtype | Open |
| devmodel.p2001 | 2013-10-11T18:25:37.716-03:00 | 2013-10-11T18:25:37.716-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Type | Edit |
| devmodel.p2001 | 2013-10-11T18:25:37.716-03:00 | 2013-10-11T18:25:37.716-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Unit-Type | file |
| devmodel.p2001 | 2013-10-11T18:25:37.716-03:00 | 2013-10-11T18:25:37.716-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Unit-Name | Teste.java |

| Eclipse | DevEvent | Edit | StateChange | Gerado quando o arquivo ativo mudou. |
|---|---|---|---|---|

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Subtype | StateChange |
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Type | Edit |
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Current-Size | 175 |
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Class-Name | Teste |
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Current-Statements | 1 |
| devmodel.p2001 | 2013-10-11T18:26:07.764-03:00 | 2013-10-11T18:26:07.764-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Teste.java | Current-Methods | 1 |

| Eclipse | DevEvent | Edit | SaveFile | Gerado quando o arquivo corrente é salvo. |
|---|---|---|---|---|

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Subtype | Save |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Type | Edit |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Current-Size | 45 |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Language | java |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Unit-Type | file |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Class-Name | Medico |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Current-Statements | 0 |
| devmodel.p2001 | 2013-10-11T18:44:12.577-03:00 | 2013-10-11T18:44:12.577-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Current-Methods | 0 |

| Eclipse | DevEvent | Edit | CloseFile | Gerado quando o arquivo corrente é fechado. |
|---|---|---|---|---|

**Exemplo:**

Continua.

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T18:45:35.176-03:00 | 2013-10-11T18:45:35.176-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Subtype | Close |
| devmodel.p2001 | 2013-10-11T18:45:35.176-03:00 | 2013-10-11T18:45:35.176-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Type | Edit |
| devmodel.p2001 | 2013-10-11T18:45:35.176-03:00 | 2013-10-11T18:45:35.176-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Language | java |
| devmodel.p2001 | 2013-10-11T18:45:35.176-03:00 | 2013-10-11T18:45:35.176-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Unit-Type | file |
| devmodel.p2001 | 2013-10-11T18:45:35.176-03:00 | 2013-10-11T18:45:35.176-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.java | Unit-Name | Medico.java |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | Exit | Gerado quando se sai do Eclipse. |
| Eclipse | DevEvent | Edit | ProgramUnit(Add) | Gerado quando é adicionado à uma unidade de programa, uma declaração de importação, classe, campo ou método. |

**Exemplo de adição de um método:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Subtype | ProgramUnit |
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Type | Edit |
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Language | java |
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Unit-Type | Method |
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Unit-Name | Medico() |
| devmodel.p2001 | 2013-10-11T19:35:47.469-03:00 | 2013-10-11T19:35:47.469-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Subsubtype | Add |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | ProgramUnit(Rename) | Gerado quando um classe, campo ou método é renomeado. |

**Exemplo de renomeação de campo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Subtype | ProgramUnit |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Type | Edit |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Language | java |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Unit-Type | Field |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | To-Unit-Name | String localmedico |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | From-Unit-Name | local |
| devmodel.p2002 | 2013-10-11T18:45:16.853-03:00 | 2013-10-11T18:45:16.853-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Subsubtype | Rename |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | ProgramUnit(Remove) | Gerado quando uma importação, classe, campo ou método é removido. |

**Exemplo de remoção de um campo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Subtype | ProgramUnit |
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Type | Edit |
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Language | java |
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Unit-Type | Field |
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Unit-Name | calorias Meta |
| devmodel.p2001 | 2013-10-11T18:49:35.428-03:00 | 2013-10-11T18:49:35.428-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Medico.j... | Subsubtype | Remove |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Edit | ProgramUnit(Move) | Gerado quando uma importação, classe, campo ou método é movido de um local para outro. |

**Exemplo de remoção de método:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Subtype | ProgramUnit |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Type | Edit |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Language | java |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Unit-Type | Method |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | To-Unit-Name | class Usuario |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | From-Unit-Name | class Medico |
| devmodel.p2003 | 2013-10-11T18:47:23.669-03:00 | 2013-10-11T18:47:23.669-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Subsubtype | Move |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Build | Compile | Gerado quando uma construção (ou compilação) falha. Captura o erro que ocorrem em uma classe java ativa no editor do ECLIPSE. |

**Exemplo:**

Continua.

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2003 | 2013-10-11T18:47:37.093-03:00 | 2013-10-11T18:47:37.093-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Subtype | Compile |
| devmodel.p2003 | 2013-10-11T18:47:37.093-03:00 | 2013-10-11T18:47:37.093-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Type | Build |
| devmodel.p2003 | 2013-10-11T18:47:37.093-03:00 | 2013-10-11T18:47:37.093-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Success | false |
| devmodel.p2003 | 2013-10-11T18:47:37.093-03:00 | 2013-10-11T18:47:37.093-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Usuario.j... | Error | Syntax error on token... |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Debug | Breakpoint(set/unset) | Gerado quando um Breakpoint é inserido/removido. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2019 | 2013-10-11T19:21:40.014-03:00 | 2013-10-11T19:21:40.014-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Cadastr... | Subtype | BreakPoint |
| devmodel.p2019 | 2013-10-11T19:21:40.014-03:00 | 2013-10-11T19:21:40.014-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Cadastr... | Type | Debug |
| devmodel.p2019 | 2013-10-11T19:21:40.014-03:00 | 2013-10-11T19:21:40.014-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Cadastr... | Set | set |
| devmodel.p2019 | 2013-10-11T19:21:40.014-03:00 | 2013-10-11T19:21:40.014-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Cadastr... | Line | 20 |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Debug | Start | Gerado quando é iniciada uma sessão de Debug. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2022 | 2013-10-11T19:04:20.405-03:00 | 2013-10-11T19:04:20.405-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Menu.java | Subtype | Start |
| devmodel.p2022 | 2013-10-11T19:04:20.405-03:00 | 2013-10-11T19:04:20.405-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Menu.java | Type | Debug |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Debug | StepInto | Gerado quando, em um sessão de Debug, entra em um bloco de código. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p4021 | 2013-11-06T20:28:35.158-02:00 | 2013-11-06T20:28:35.158-02:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/model/Main.j... | Subtype | Step into |
| devmodel.p4021 | 2013-11-06T20:28:35.158-02:00 | 2013-11-06T20:28:35.158-02:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/model/Main.j... | Type | Debug |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Debug | StepOver | Gerado quando, em um sessão de Debug, passa-se sobre um bloco de código. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p4021 | 2013-11-06T20:28:25.816-02:00 | 2013-11-06T20:28:25.816-02:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/model/Main.j... | Subtype | Step over |
| devmodel.p4021 | 2013-11-06T20:28:25.816-02:00 | 2013-11-06T20:28:25.816-02:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/model/Main.j... | Type | Debug |

| | | | | |
|---|---|---|---|---|
| Eclipse | DevEvent | Debug | Terminate | Gerada quando a sessão de Debug é finalizada. |

**Exemplo:**

| Owner | timestamp | runtime | tool | sensordatatype | resource | key | value |
|---|---|---|---|---|---|---|---|
| devmodel.p2022 | 2013-10-11T19:03:28.549-03:00 | 2013-10-11T19:03:28.549-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Menu.java | Subtype | Terminate |
| devmodel.p2022 | 2013-10-11T19:03:28.549-03:00 | 2013-10-11T19:03:28.549-03:00 | Eclipse | DevEvent | file:/C:/Java/Eclipse/Eclipse_Indigo/workspace/MyBMI/src/Menu.java | Type | Debug |

# Apêndice B

# Exemplos de Cálculos das Métricas de Complexidade

Neste Apêndice serão apresentados exemplos de como calcular algumas das métricas de complexidade apresentadas no Capítulo 2. Para as demais métricas (tamanho, herança e acoplamento) não foram apresentados exemplos porque foram consideradas de fácil entendimento.

## A.1. Exemplo de Cálculo da CCM:

A Figura 15 mostra um exemplo de aplicação da métrica CCM em um pequeno programa escrito em linguagem JAVA que imprime na tela os números pares entre 0 e 20. Nela pode-se visualizar também o grafo que representa a estrutura deste programa.

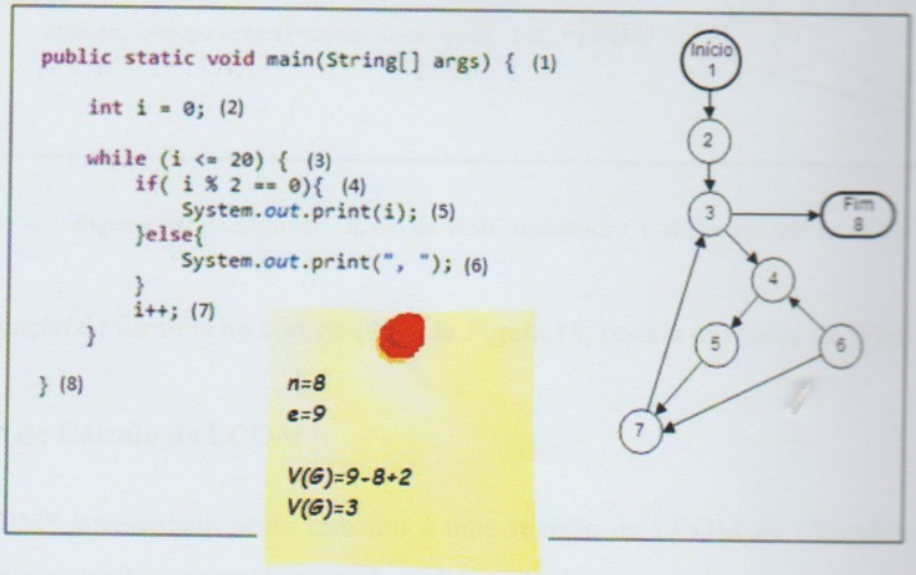


**Figura 15-** Exemplo de Aplicação da Métrica CCM (Fonte: Autor).

Analisando a Figura 15 pode-se ver uma estrutura de código-fonte representada num grafo com 8 nós e 9 arestas. No código-fonte, cada linha contém o número do nó correspondente do grafo,

entre parênteses. Aplicando a fórmula, obtém-se o resultado igual a 3.

### A.2. Exemplo de Cálculo da WMC:

A Figura 16 mostra um exemplo de cálculo da WMC que utiliza a CCM para calcular a complexidade dos métodos.

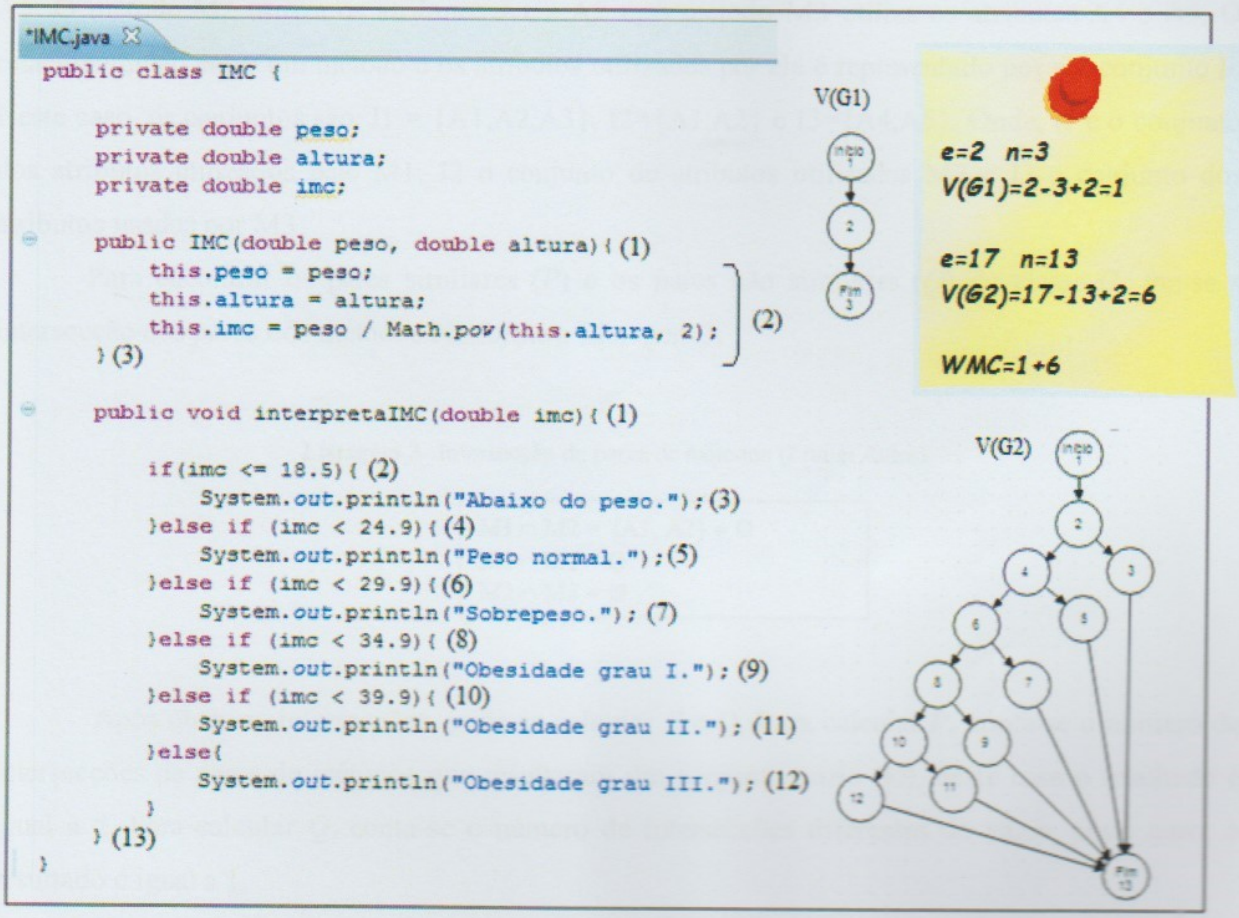


**Figura 16**- Exemplo de cálculo da WMC utilizando CCM (Fonte: Autor).

A aplicação da fórmula no código-fonte da Figura 16, resulta um valor de WMC igual a 7.

### A.3. Exemplo de Cálculo da LCOM*:

A LCOM* apresentada neste trabalho é uma revisão da LCOM de Chimdaber e Kemerer [CHI91]. Para um melhor entendimento da métrica, será apresentado um exemplo adaptado da versão LCOM de Chimdaber e Kemerer [CHI91]. Para os autores, LCOM é dado pelo número de pares de métodos com similaridade igual à zero ($P$), menos o número de pares de métodos cuja similaridade é diferente de zero ($Q$) (Equação 2.6):

$$LCOM = P - Q, quando P > Q, senão LCOM = 0 \quad (2.6)$$

Onde, $P$ é o número de pares de métodos que não compartilham atributos; $Q$ é o número de pares de métodos que compartilham atributos.

Considere uma classe C com os métodos M1, M2 e M3 e com os atributos A1, A2, A3, A4 e A5. O método M2 utiliza os atributos A1 e A2 e, o método M3 utiliza os atributos A4 e A5. O relacionamento entre um método e os atributos utilizados por ele é representado por um conjunto Ii. Neste caso, os conjuntos são: I1 = {A1,A2,A3}, I2={A1,A2} e I3={A4,A5}. Onde, I1 é o conjunto dos atributos utilizados pelo M1, I2 o conjunto de atributos utilizados M2 e I3 o conjunto dos atributos usados por M3.

Para encontrar os pares similares ($P$) e os pares não similares ($Q$) da classe C, faz-se a intersecção dos pares dos métodos conforme a Listagem 3.

**Listagem 3**- Intersecção de pares de métodos (Fonte: Autor).

```
M1 ∩ M2 = {A1, A2} ≠ Ø
M1 ∩ M3 = Ø
M2 ∩ M3 = Ø
```

Após obter as intersecções, pode-se calcular $P$ e $Q$, Para calcular $P$, conta-se o número de intersecções de pares de métodos que resultaram um conjunto vazio (Ø), neste caso o resultado é igual a 2. Para calcular $Q$, conta-se o número de intersecções diferentes de vazio, neste caso, o resultado é igual a 1.

Com $P$ e $Q$ calculados, pode-se aplicar a Equação 2.6 para encontrar a LCOM. Aplicando $P$ e $Q$ deste exemplo, obtém LCOM = 1.

A métrica LCOM passou por revisões para corrigir algumas falhas, uma das revisões é a de [HEN96] chamada de LCOM* apresentada no Capítulo 2 deste trabalho. A nova revisão foi criada porque, segundo o autor, existiam muitos casos de classes diferentes onde o valor da LCOM era o mesmo, mas, não representavam a mesma coesão. Um exemplo pode ser visto na Figura 17, onde duas classes (a) e (b) apresentam o mesmo valor de LCOM, porém a classe (a) tem baixa coesão e a classe (b) tem alta coesão.

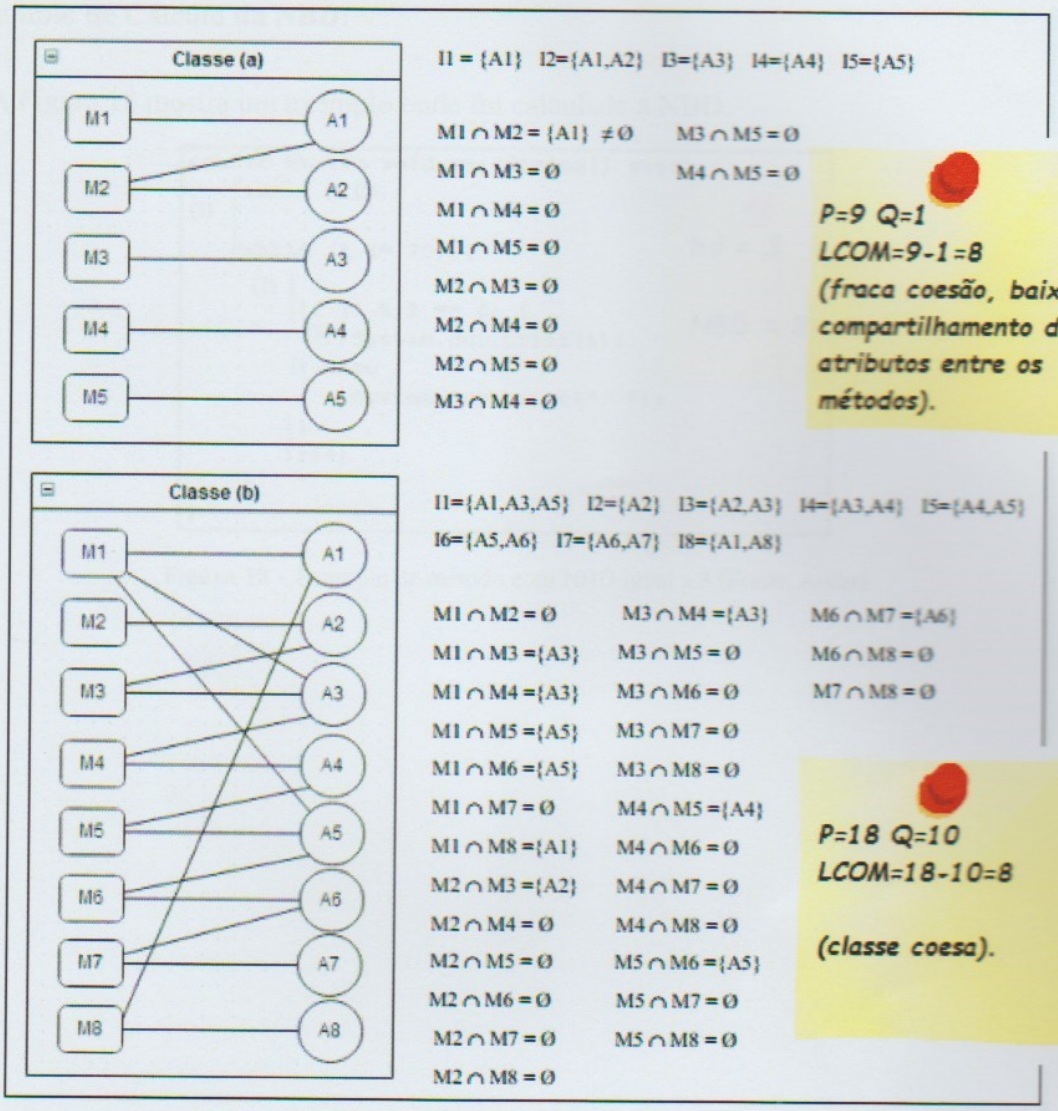


**Figura 17**- Exemplo de classes com o mesmo valor de LCOM (Adaptado de [HEN96] (p. 145))

Outro problema levantado por Henderson-Sellers [HEN96] é que nem sempre um valor de LCOM igual à zero é um indicador de boa coesão. Desta forma o autor propôs a LCOM*, onde uma classe é considerada coesa quando todos os seus métodos acessam todos os atributos, resultando em LCOM igual à zero. Valores baixos, próximos de zero, são indicativos de coesão. Valores próximos de 1 indicam falta de coesão e, valores maiores que um indicam problemas sérios de coesão, onde certamente a classe deverá ser subdivida em outras classes.

A LCOM* é calculada da seguinte forma: dada uma classe C, com um conjunto de métodos {Mi} (M1, M2,...Mn) que acessam um conjunto de atributos {Ai}(A1,A2,...An), calcula-se o número de métodos que acessam cada atributo, representado por m(Ai), na sequência faz-se a média de m(Ai) para todos os atributos e subtrai-se o número de métodos (m), o resultado é dividido por (1-m).

### A.4. Exemplo de Cálculo da NBD:

A Figura 18 mostra um exemplo onde foi calculada a NBD.

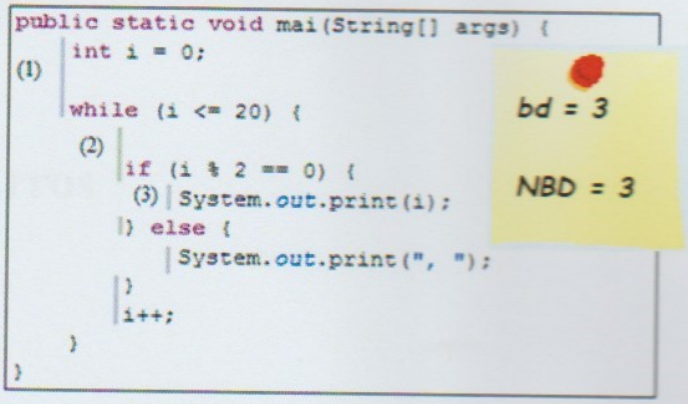


**Figura 18** - Exemplo de método com NBD igual a 3 (Fonte: Autor).

# Apêndice C

## Grupos de Erros

Neste Apêndice são apresentados os grupos de erros levantados por um especialista em linguagem de programação Java ao analisar os erros coletados pelo sensor HACKYSTAT (Tabela 23).

**Tabela 23**- Grupos de Erros (Fonte: Autor).

| Código do Grupo | Descrição |
|---|---|
| 1 | Erro de sintaxe |
| 2 | Elemento não declarado |
| 3 | Problema com referência estática para elemento não estático |
| 4 | Problema com construtor |
| 5 | Elemento duplicado |
| 6 | Problema com retornos |
| 7 | Problema com métodos |
| 8 | Problema com tipo de dado |
| 9 | Problema com modificadores de acesso |
| 10 | Problema de atribuição |
| 11 | Código inacessível |
| 12 | Problema com operadores |

O Código do grupo é utilizado na característica *codErro* do perfil dinâmico do desenvolvedor. Os grupos foram criados para simplificar a grande quantidade de erros que são cometidos pelos desenvolvedores enquanto codificam.

# Apêndice D

## Cenário : MyBMI

A mudança nos hábitos de vida e, consequentemente, das questões alimentares são o centro da atenção do cotidiano de muitas pessoas. Estratégias alternativas a formas de tratamento não farmacológico da obesidade são necessárias nos dias atuais. Como apontado pela pesquisa VIGITEL 2012 [POR12] o número de pessoas com excesso de peso e obesas vem crescendo muito, o percentual de pessoas com excesso de peso supera mais da metade da população brasileira. Cresce também a preocupação, por parte de muitas pessoas, para manter um peso saudável. Uma prática conhecida e adequada é a vigilância do peso e da ingestão diária de calorias, além da prática de exercícios físicos e alimentação balanceada. A motivação para a escolha do tema vem da publicação dos resultados da pesquisa de Vigilância de Fatores de Risco e Proteção para Doenças Crônicas por Inquérito Telefônico - VIGITEL 2012, promovida pelo Ministério da saúde, divulgada em agosto deste ano. A pesquisa aponta que a proporção de pessoas (acima de 18 anos) acima do peso no Brasil avançou de 42,7%, em 2006, para 51% em 2012. Entre os homens, o excesso de peso atinge 54% e entre as mulheres, 48%. Em Curitiba (PR), o percentual de obesos passou de 12,3%, em 2006, para 16,3%, em 2012. Com relação ao excesso de peso, os números passaram de 43,7% para 51,6%. Para a pesquisa, foram entrevistados 45,4 mil pessoas em todas as capitais e também no Distrito Federal, entre julho de 2012 a fevereiro de 2013. O objetivo deste estudo foi retratar os hábitos da população brasileira para apoiar o desenvolvimento de políticas públicas de saúde preventiva [POR12].

Desta forma, é proposto a implementação de uma aplicação em Java para auxiliar pessoas interessadas em manter um peso saudável através de um programa que exibe informações relevantes e sintetizadas para acompanhamento e monitoramento de peso e de ingestão diária de calorias. O sistema deve registrar dados como: sexo, altura, peso, idade, peso e calorias ingeridas um cada uma das refeições do dia, por um usuário. Além disso, deve registrar o peso meta que o usuário pretende atingir e a quantidade de calorias meta que o usuário deseja ingerir por dia para

perder, manter ou ganhar peso. O sistema deve reproduzir as histórias da Tabela 24 a seguir:

**Tabela 24**- Histórias de Usuário (Fonte: Autor).

| HISTÓRIAS DE USUÁRIO |
|---|
| **História 1:** o sistema deve registrar logo na abertura o sexo do usuário, a altura em metros, o peso em kg, a idade em anos, o peso meta em kg e a quantidade de calorias meta. O peso meta é o peso que o usuário deseja atingir seguindo uma dieta e controlando as calorias ingeridas diariamente. A quantidade de calorias meta é a quantidade de calorias que o usuário pretende ingerir diariamente para perder, ou ganhar peso e atingir o peso meta. |
| **História 2:** após obter os dados da História 1, o sistema deve apresentar um menu para o usuário selecionar as seguintes opções: calcular IMC (Índice de Massa Corporal), calcular GEB (Gasto Energético Basal), lançar calorias ingeridas em cada uma das refeições (Café, Lanche da Manhã, Almoço, Lanche da Tarde, Janta e Ceia) do dia, e visualizar informações para acompanhamento de peso e de ingestão de calorias. Cada uma das opções do menu serão descritas nas histórias a seguir. |
| **História 3:** calcular o IMC (Índice de Massa Corporal) utilizando a seguinte fórmula:<br>$$IMC = \frac{peso}{altura^2}$$<br>Onde, peso é em quilogramas e altura em metros.<br>O resultado do IMC deve ser apresentado para o usuário e seu resultado deve ser interpretado utilizando a Tabela Comparativa da ABESO [ABE12]:<br>**Categoria** \| **IMC**<br>Abaixo do peso \| Abaixo de 18,5<br>Peso Normal \| 18,5 – 24,9<br>Sobrepeso \| 25,0 – 29,9<br>Obesidade Grau 1 \| 30,0 – 34,9<br>Obesidade Grau 2 \| 35,0 – 39,9<br>Obesidade Grau 3 \| 40,0 e acima<br>Adaptado de [ABE12].<br>A interpretação deve ser apresentada na forma de mensagem. |
| **História 4:** calcular o GEB (Gasto Energético Basal) utilizando a fórmula de Harris-Benedict, que leva em conta o sexo, a altura, peso e idade do usuário [GAS13]. As fórmulas para calcular o GEB para homens e mulheres são apresentadas a seguir:<br>$$\text{Homens} = 66{,}5 + (13{,}7 * \text{Peso}) + (5 * \text{Altura}) - (6{,}8 * \text{Idade})$$<br>$$\text{Mulheres} = 665 + (9{,}6 * \text{Peso}) + (1{,}7 * \text{Altura}) - (4{,}7 * \text{Idade})$$<br>Onde peso é em quilogramas, a altura em centímetros e a idade em anos.<br>O valor do GEB deve ser apresentado ao usuário juntamente com a mensagem a seguir:<br>*GEB é a quantidade de calorias gasta pelo organismo para manter as funções vitais em funcionamento, estando o indivíduo acordado e em repouso completo. São exemplos de funções vitais: circulação sanguínea, respiração e manutenção da temperatura corporal.*<br>*O GEB é a quantidade mínima de calorias que o organismo necessita. Se desejar emagrecer, procure ingerir uma quantidade de calorias menor que seu GEB. Lembrando que toda dieta deve ter acompanhamento médico, por isso, antes iniciá-la.* |
| **História 5:** o lançamento de calorias ingeridas em uma refeição consiste em informar a refeição (café da manhã, lanche da manhã, almoço, lanche da tarde, janta ou ceia) e quantidade de calorias ingeridas nela. O sistema deve permitir ao usuário informar as calorias para as refeições que ele fez num dia. Os valores não precisam ser armazenados em banco de dados. Podem ser armazenados em uma lista de refeições do dia, por exemplo. Não deve ser permitido lançar a mesma refeição duas vezes. |
| **História 6:** permitir ao usuário visualizar informações para acompanhamento de peso e de ingestão de calorias. Visualizar o peso atual do usuário, o peso meta e mostrar a diferença. De acordo com a diferença, mostrar uma mensagem de advertência caso ainda não tenha atingido a meta, ou mensagem de |

felicitação caso contrário. Em relação às informações de calorias ingeridas, calcular as calorias ingeridas em cada uma das refeições e totalizar. A totalização deve ser apresentada juntamente com a quantidade de calorias meta. Realizar a diferença entre elas e mostrar uma mensagem de advertência ou felicitação, seguindo o exemplo do peso.

Para ajudar a entender o que se pede nas histórias, a seguir são apresentadas protótipos de telas.

A Figura 19 mostra dois protótipos de telas diferentes para lançamento dos dados do usuário. A tela (a) mostra dados de um usuário que não é médico, já a tela (b) mostra dados de um usuário que é médico.

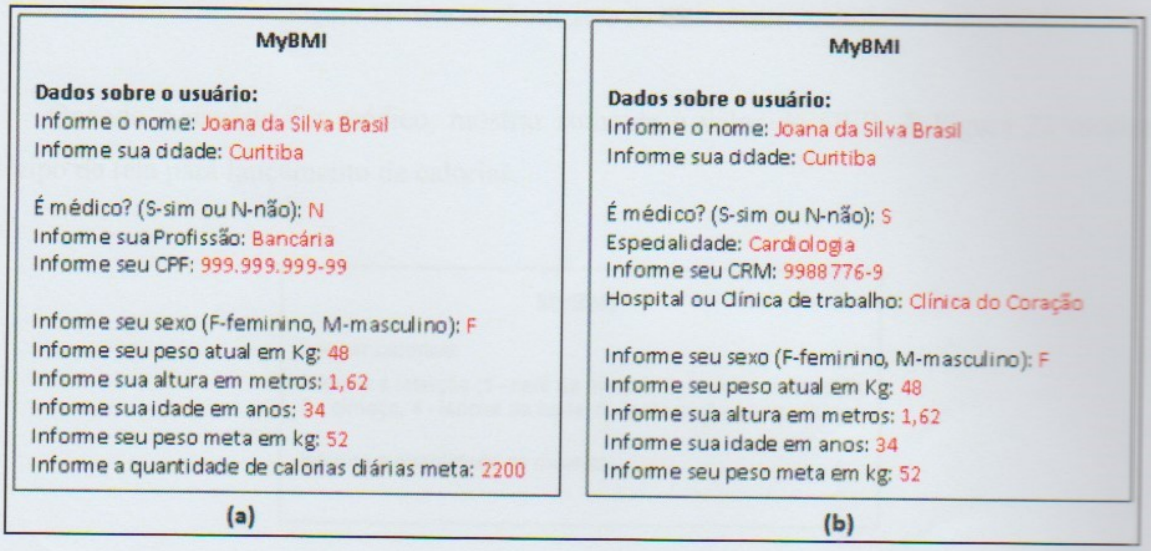


**Figura 19**- Protótipos de Telas para Lançamento de Dados de Usuário (Fonte: Autor).

A Figura 20 mostra protótipos de telas que mostram o IMC calculado com base nos dados do usuário de acordo com o tipo de usuário: não médico (a) e médico (b).

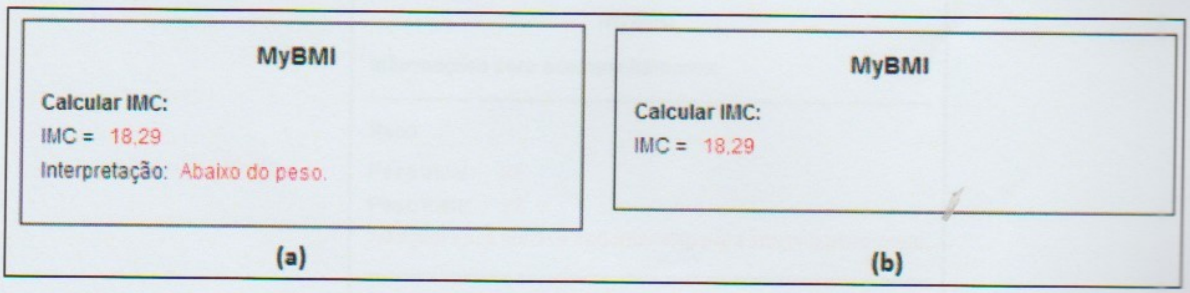


**Figura 20**- Telas de Visualização do IMC (Fonte: Autor).

A Figura 21 mostra o GEB calculado com base nos dados de um usuário não médico.

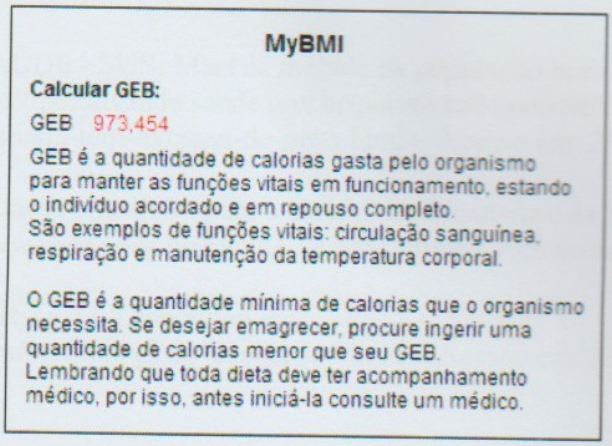


**Figura 21**- Tela de visualização do GEB (Fonte: Autor).

Quando o usuário for médico, mostrar somente o valor do GEB. A Figura 22 mostra um protótipo de tela para lançamento de calorias.

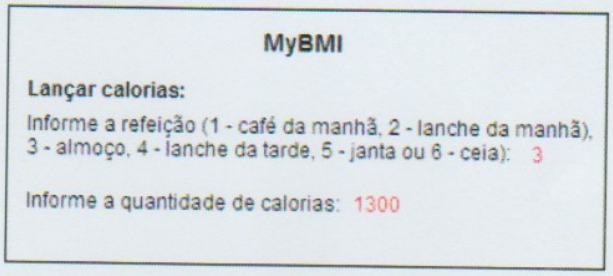


**Figura 22**- Tela de lançamento de calorias (Fonte: Autor).

A Figura 23 mostra um protótipo de tela das informações de acompanhamento, tanto de peso quanto de calorias.

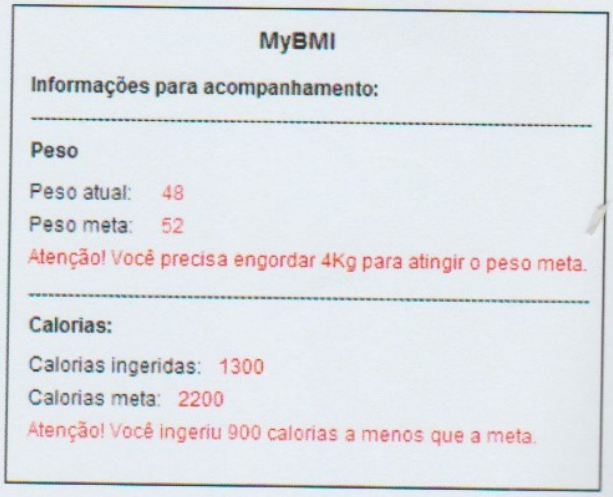


**Figura 23**- Tela de informações de Acompanhamento (Fonte: Autor).

**Referência do Cenário:**

[POR12] PORTAL DA SAÚDE - SUS. Mais da metade da população brasileira tem excesso de peso. Disponível em: <http://portalsaude.saude.gov.br/portalsaude/noticia/12926/162/mais-da-metade-da-populacao-brasileira-tem-excesso-de-peso.html>. Acesso em: 23/09/2013.

[ABE12] ABESO – Associação Brasileira para o estudo da obesidade e da Síndrome metabólica. Disponível em: <http://www.abeso.org.br/calcule-seuimc.shtml>. Acesso em: 19/09/2012.

[GAS13] GASTO ENERGÉTICO. Disponível em: <http://www.uftm.edu.br/nutro/html/gasto_energetico.html>. Acesso em: 23/09/2013.

# Apêndice E

## Documentos Comitê de Ética da PUCPR

O projeto foi aprovado pelo Comitê de Ética da PUCPR. Segue o Comprovante de Envio, o Termo de Consentimento Livre e Esclarecido (TCLE) e o Parecer Consubstanciado do CEP.

## COMPROVANTE DE ENVIO DO PROJETO

**DADOS DO PROJETO DE PESQUISA**

**Título da Pesquisa:** DevMODEL: Uma Ferramenta para a Construção do Modelo Dinâmico do Desenvolvedor

**Pesquisador:** Franciele Beal

**Versão:** 1

**CAAE:** 18200313.6.0000.0020

**Instituição Proponente:** Pontifícia Universidade Católica do Parana - PUCPR

**DADOS DO COMPROVANTE**

**Número do Comprovante:** 044094/2013

**Patrocionador Principal:** Financiamento Próprio

**Endereço:** Rua Imaculada Conceição 1155
**Bairro:** Prado Velho **CEP:** 80.215-901
**UF:** PR **Município:** CURITIBA
**Telefone:** (41)3271-2292 **Fax:** (41)3271-2292 **E-mail:** nep@pucpr.br

## TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu, nome: _____, nacionalidade: _____, idade: ___ anos, estado civil: _____, profissão: _____, endereço (rua, nº, bairro, cidade/UF e complemento) :_____, RG: _____, estou sendo convidado a participar de um estudo denominado DevMODEL: Uma Ferramenta para a Construção do Modelo Dinâmico do Desenvolvedor, cujos objetivos e justificativas são: identificar e classificar desenvolvedores de software em diferentes perfis (Júnior, Pleno e Sênior) com base em características obtidas do código-fonte produzido por eles. Para isso, se faz necessária a construção de uma base com dados históricos coletados (de forma automática e não invasiva) durante o trabalho de codificação dos desenvolvedores. Esta base será utilizada no treinamento de um algoritmo de classificação. A classificação é importante para auxiliar a gerência de projetos e as equipes de desenvolvimento na identificação dos perfis de seus desenvolvedores e assim fornecer treinamentos e suportes aos seus membros.

A minha participação no referido estudo será no sentido de executar (implementar) um cenário fornecido pelo pesquisador em um ambiente de desenvolvimento de software. Este ambiente é constituído de um IDE Eclipse, da linguagem de programação Java e de dois sensores de coleta de dados. Este ambiente será instalado e configurado pelo pesquisador sendo de minha responsabilidade somente executar o cenário fornecido. Os sensores irão coletar dados enquanto eu escrevo o código-fonte. Não serão coletados dados pessoais que possam revelar minha identidade. Os dados coletados serão referentes às medidas de qualidade de código-fonte (por exemplo, número de linhas de código, número de classes construídas, número de atributos, número de métodos, complexidade ciclomática de McCabe, etc.), informações sobre a compilação (sucesso ou com fracasso) do programa, a utilização (ou não) do depurador de código e os erros de sintaxe cometidos. Estou ciente que os dados coletados serão utilizados para constituir uma base que será disponibilizada para a comunidade de pesquisadores da área. Tal base não contém qualquer referência que possa identificar os voluntários da pesquisa.

Fui alertado de que, da pesquisa a se realizar, posso esperar alguns benefícios indiretos, tais como: a definição de um método capaz de classificar de forma

automática os desenvolvedores, e assim poder receber conselhos (por parte do software) de boas práticas de programação.

Recebi, por outro lado, os esclarecimentos necessários sobre os possíveis desconfortos e riscos decorrentes do estudo, levando-se em conta que é uma pesquisa, e os resultados positivos ou negativos somente serão obtidos após a sua realização. Assim, estou ciente que o tempo despendido neste experimento poderá ser em vão, visto que a pesquisa pode não revelar resultados significativos.

Estou ciente de que minha privacidade será respeitada, ou seja, meu nome ou qualquer outro dado ou elemento que possa, de qualquer forma, me identificar, será mantido em sigilo.

Também fui informado de que posso me recusar a participar do estudo, ou retirar meu consentimento a qualquer momento, sem precisar justificar, e de, por desejar sair da pesquisa, não sofrerei qualquer prejuízo à assistência que venho recebendo.

Os pesquisadores envolvidos com o referido projeto são Prof. Dr. Emerson Cabrera Paraiso professor responsável pela pesquisa e membro permanente do Programa de Pós-Graduação em Informática da PUCPR e Franciele Beal aluna de mestrado do Programa de Pós-Graduação em Informática da PUCPR, e com eles poderei manter contato pelo telefone (41) 3271-1676 ou (46) 9921-8259.

É assegurada a assistência durante toda pesquisa, bem como me é garantido o livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências, enfim, tudo o que eu queira saber antes, durante e depois da minha participação.

Enfim, tendo sido orientado quanto ao teor de todo o aqui mencionado e compreendido a natureza e o objetivo do já referido estudo, manifesto meu livre consentimento em participar, estando totalmente ciente de que não há nenhum valor econômico, a receber ou a pagar, por minha participação.

Em caso de reclamação ou qualquer tipo de denúncia sobre este estudo devo ligar para o CEP PUCPR (41) 3271-2292 ou mandar um email para nep@pucpr.br

Curitiba, ____ de _____ de 2013.

_____ _____

*Participante* *Franciele Beal*

_____

*Prof. Dr. Emerson Cabrera Paraiso*

## PARECER CONSUBSTANCIADO DO CEP

### DADOS DO PROJETO DE PESQUISA

**Título da Pesquisa:** DevMODEL: Uma Ferramenta para a Construção do Modelo Dinâmico do Desenvolvedor

**Pesquisador:** Franciele Beal

**Área Temática:**

**Versão:** 1

**CAAE:** 18200313.6.0000.0020

**Instituição Proponente:** Pontifícia Universidade Católica do Parana - PUCPR

**Patrocinador Principal:** Financiamento Próprio

### DADOS DA NOTIFICAÇÃO

**Tipo de Notificação:** Envio de Relatório Final

**Detalhe:**

**Justificativa:**

**Data do Envio:** 22/05/2014

**Situação da Notificação:** Parecer Consubstanciado Emitido

### DADOS DO PARECER

**Número do Parecer:** 675.150

**Data da Relatoria:** 05/06/2014

**Apresentação da Notificação:**

Este relatório apresenta o resultado da participação voluntária de desenvolvedores JAVA para a construção de uma base com dados históricos sobre o trabalho dos mesmos.

**Objetivo da Notificação:**

Apresentar relatório final.

**Avaliação dos Riscos e Benefícios:**

Contemplados.

**Comentários e Considerações sobre a Notificação:**

Dos 64 desenvolvedores que formam a Base Histórica, foram selecionados 57desenvolvedores para a base de treinamento, sendo 19 desenvolvedores de cada uma das classes. Como na classe

P2000 existem 26 desenvolvedores, foram selecionados aleatoriamente 19. O resultado final foi uma base de treinamento com 2191 registros.O conjunto de dados de Treinamento foi utilizado para treinar os classificadores indicados no método do projeto: k-NN (IBK), Naive Bayes, Árvore de Decisão (J48), Rede Neural Artificial (Multilayer Perceptron) e SVM (libSVM).O classificador que apresentou melhor desempenho foi a Árvore de Decisão com
90,96% de Acurácia, seguida pelos classificadores Rede Neural Artificial do tipo Multilayer Perceptron com uma Acurácia de 86,44%, k-NN 85,08%, SVM 79,37% e o Naive Bayes com 65,91%.

**Considerações sobre os Termos de apresentação obrigatória:**
Relatório contempla todo período da pesquisa e esta de acordo com a resolução 466/12

**Recomendações:**
Sem recomendações

**Conclusões ou Pendências e Lista de Inadequações:**
Aprovado

**Situação do Parecer:**
Aprovado

**Necessita Apreciação da CONEP:**
Não

**Considerações Finais a critério do CEP:**

CURITIBA, 05 de Junho de 2014

**Assinado por:**
**NAIM AKEL FILHO**
**(Coordenador)**