COPPE UFRJ
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

INTELIGÊNCIA COMPUTACIONAL NA AVALIAÇÃO DE CÓDIGOS EM UM SISTEMA COMPLEXO DE DETECÇÃO COM DESENVOLVIMENTO COLABORATIVO

Andressa A. Sivolella Gomes

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: José Manoel de Seixas

Rio de Janeiro
Março de 2016

INTELIGÊNCIA COMPUTACIONAL NA AVALIAÇÃO DE CÓDIGOS EM UM SISTEMA COMPLEXO DE DETECÇÃO COM DESENVOLVIMENTO COLABORATIVO

Andressa A. Sivolella Gomes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Aluízio Fausto Ribeiro Araújo, D.Sc.

Prof. Afonso de Bediaga e Hickman, D.Sc.

Pesquisadora Carmen Lúcia Lodi Maidantchik, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2016

Sivolella Gomes, Andressa A.

Inteligência Computacional na Avaliação de Códigos em um Sistema Complexo de Detecção com Desenvolvimento Colaborativo /Andressa A. Sivolella Gomes. – Rio de Janeiro: UFRJ/COPPE, 2016.

XI, 28 p.: il.; 29, 7cm.

Orientador: José Manoel de Seixas

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2016.

Referências Bibliográficas: p. 23 – 27.

1. Mineração de códigos. 2. Métodos ensemble com árvores. 3. Plataforma colaborativa. I. Manoel de Seixas, José. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

A todo mundo, geralzão.

Agradecimentos

Gostaria de agradecer a todos.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

INTELIGÊNCIA COMPUTACIONAL NA AVALIAÇÃO DE CÓDIGOS EM UM SISTEMA COMPLEXO DE DETECÇÃO COM DESENVOLVIMENTO COLABORATIVO

Andressa A. Sivolella Gomes

Março/2016

Orientador: José Manoel de Seixas

Programa: Engenharia Elétrica

Apresenta-se, nesta tese, ...

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

COMPUTATIONAL INTELLIGENCE IN SOURCE CODE ASSERTION IN A COMPLEX SYSTEM IN A COLLABORATIVE DEVELOPMENT ENVIROMENT

Andressa A. Sivolella Gomes

March/2016

Advisor: José Manoel de Seixas

Department: Electrical Engineering

In this work, we present ...

Sumário

Lista de Figuras **x**

Lista de Tabelas **xi**

1 Introdução **1**
1.1 Motivação 1
1.2 Objetivos 1
1.3 Organização do documento 1

2 A Física de Altas Energias **2**
2.1 CERN e LHC 2
2.2 Experimento ATLAS 3
2.3 Calorímetro Hadrônico de Telhas (TileCal) 4
2.3.1 Análise *Online* e *Offline* 5
2.3.2 A colaboração TileCal 7

3 Plataforma web Tile-in-ONE **8**
3.1 Fluxo de dados 9
3.2 Novo desenvolvimento 11

4 Mineração de códigos fonte para identificação de falhas **13**
4.1 Seleção de atributos 15
4.1.1 Analisadores Estáticos 15
4.1.2 Medidas estatísticas e de qualidade 16
4.2 Analisadores estáticos e Mineração de códigos no CERN 18

5 Metodologia **20**
5.1 Aquisição de dados 20
5.2 Seleção de categorias 20
5.3 Métodos *ensemble* em Árvores de Decisão 20

6 Resultados **21**
6.1 AvaliaÃ§Ã£o dos Classificadores 21

7 Conclusões **22**

Referências Bibliográficas **23**

A Alguns casos **28**

Lista de Figuras

2.1 Representação aérea do LHC e seus detectores no CERN. Extraído de [1]. 3
2.2 Esquema do detector ATLAS. Adaptado de [2] 4
2.3 Calorímetro de Telhas do ATLAS (com suas 4 partições na cor verde) envolvendo o calorímetro de argônio líquido. O Sistema Magnético e a Câmara de Múons não estão ilustrados nesta figura. Adaptado de [3]. 5
2.4 Estrutura de absorção e amostragem do TileCal. Extraído de [4] . . . 6
2.5 Esboço das células do TileCal. Extraído de [3]. 6

3.1 *Dashboard Web System*, desenvolvido em 2010 para integrar as os dados adquiridos e as análises do grupo de QD do TileCal. Mais detalhes em [5] 9
3.2 Esquemático representando a infraestrutura da plataforma Tile-in-ONE. Em vermelho, a nova camada a ser desenvolvida. 11
3.3 Fluxo de dados da plataforma web Tile-in-ONE. 12

4.1 Estimativa do percentual de erros inseridos durante fases do SDLC. (Fonte: *Computer Finance Magazine*. Dados extraídos de [6]) 14

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

1.2 Objetivos

1.3 Organização do documento

Capítulo 2

A Física de Altas Energias

A Física de Partículas estuda os constituintes da matéria e interações entre elas. Esse estudo também é referenciado como Física de Altas Energias, devido à grande quantidade de energia necessária para a observação das estruturas básicas da matéria.

Este capítulo aborda o ambiente da física de partículas no CERN, bem como o colisor de partículas atualmente em operação, o LHC, com destaque para o experimento ATLAS e um de seus calorímetros (TileCal).

2.1 CERN e LHC

Fundado em 1954, o CERN (em francês *Centre Européen pour la Recherché Nucléaire* [7]) é o maior centro de pesquisa na área de física de partículas de altas energias no mundo. Atualmente, conta com a participação de 38 países membros e outros países colaboradores, entre eles o Brasil.

O LHC (em inglês *Large Hadron Collider* [8]) é o maior colisor de partículas já construído e encontra-se atualmente em operação no CERN. Instalado a 175 metros abaixo do solo, consiste em um grande túnel em formato anelar, com 27 km de perímetro. Quando partículas são eletricamente carregadas e submetidas a pulsos eletromagnéticos no interior de tubos tem-se o processo de aceleração de partículas. Neste processo, as partículas adquirem aceleração ao serem envolvidas a campos eletromagnéticos variantes [9]. Em aceleradores circulares, partículas são injetadas e circulam no anel até atingirem a energia desejada. Diversos experimentos podem ser realizados em determinados pontos de colisão ao longo da circunferência. O perímetro do acelerador circular é diretamente proporcional a energia necessária na colisão. Ao atingir a energia desejada, os feixes de partículas aceleradas em sentidos opostos colidem e, nesse caso, detectores em formato cilíndrico são os mais utilizados. Cada colisão gera informações que devem ser analisadas por especialistas, com objetivos distintos, tais como: identificação de sub-partículas, observação de fenômenos

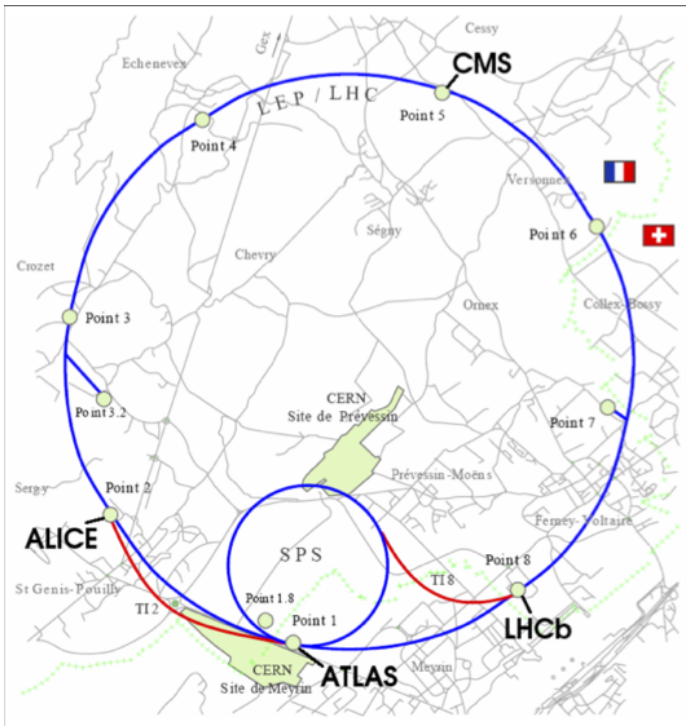


Figura 2.1: Representação aérea do LHC e seus detectores no CERN. Extraído de [1].

e a comprovação ou a eliminação de teorias físicas. Nos pontos de colisão, detectores são montados para observação e coleta de dados. Existem, por exemplo, detectores de calorimetria, para identificar a energia depositada pelas partículas colisionadas; detectores de traços, para observar trajetórias após colisões; e detectores de múons, responsáveis por absorver tais partículas.

A figura 2.1 ilustra uma representação aérea do LHC. Existem quatro detectores de partículas instalados em pontos de colisão ao longo da extensão do LHC, altamente especializados: ALICE [10], ATLAS [3], CMS [11] e LHCb [12]. Os detectores ilustrados permitem obter informações detalhadas sobre a trajetória das partículas resultantes das colisões ocorridas e características energéticas, sendo possível identificar partículas. O ATLAS é o maior dos detectores do LHC.

2.2 Experimento ATLAS

O ATLAS (em inglês *A Toroidal LHC ApparatuS*) [3] é um detector em formato cilíndrico. O detector pesa aproximadamente 7.000 toneladas, com 44 metros de comprimento por 24 metros de altura. A figura 2.2 ilustra os diversos subsistemas que compõem o detector ATLAS.

Cada subsistema apresenta uma característica específica de acordo com suas funcionalidades. São eles, da parte interna para a externa:

- Detector Interno (ID), responsável por registrar as trajetórias das partículas resultantes da colisão, bem como o momento e o sinal da carga, se for o caso;
- Sistema de Calorimetria, composto pelos calorímetros eletromagnético (LAr) e hadrônico de telhas (TileCal). Os calorímetros citados são responsáveis por medirem a energia das partículas resultantes;
- Sistema Magnético, responsável por desviar partículas carregadas para medir o momento;
- o Espectrômetro de Múons, responsável por medir a trajetória de uma determinada partícula denominada Múon.

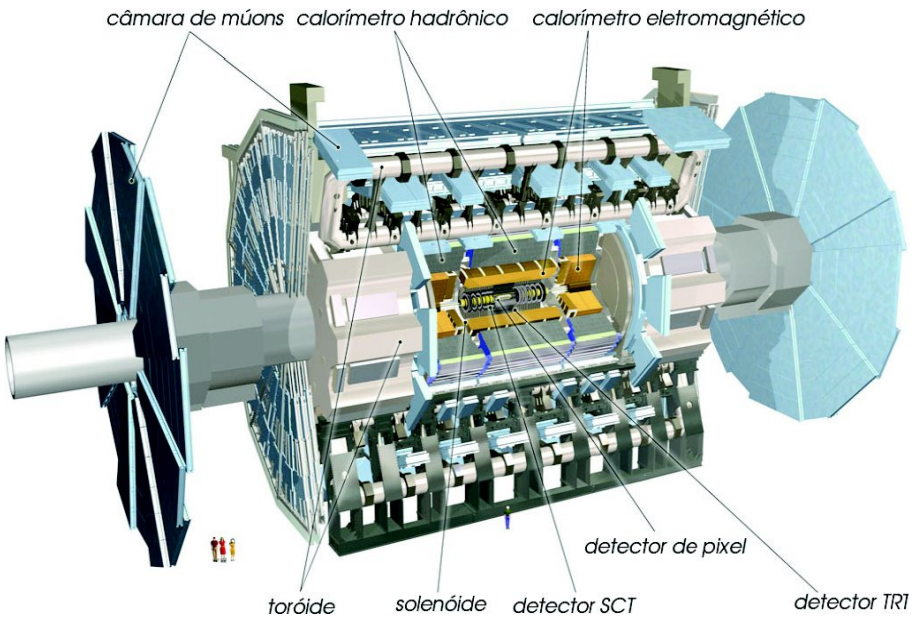


Figura 2.2: Esquema do detector ATLAS. Adaptado de [2]

O sistema de calorimetria do LHC foi projetado para absorver a energia das partículas que cruzam o detector, sendo o calorímetro hadrônico de telhas (em inglês *Tile Calorimeter* ou TileCal) é o foco deste mestrado.

2.3 Calorímetro Hadrônico de Telhas (TileCal)

O TileCal [4] é composto por três barris: um central (que se divide em duas partições: LBA e LBC) e dois estendidos (cada um correspondendo a mais duas partições, EBA e EBC, respectivamente), como ilustrado na figura 2.3. Cada partição é igualmente dividida em 64 partes (módulos). Os módulos do TileCal são compostos de ferro como mediadores passivos e telhas cintilantes como material ativo. A estrutura absorvedora é coberta de placas de aço de várias dimensões,

conectadas a uma estrutura maciça de suporte. A principal inovação deste detector é a posição perpendicular das telhas em relação aos feixes do LHC.

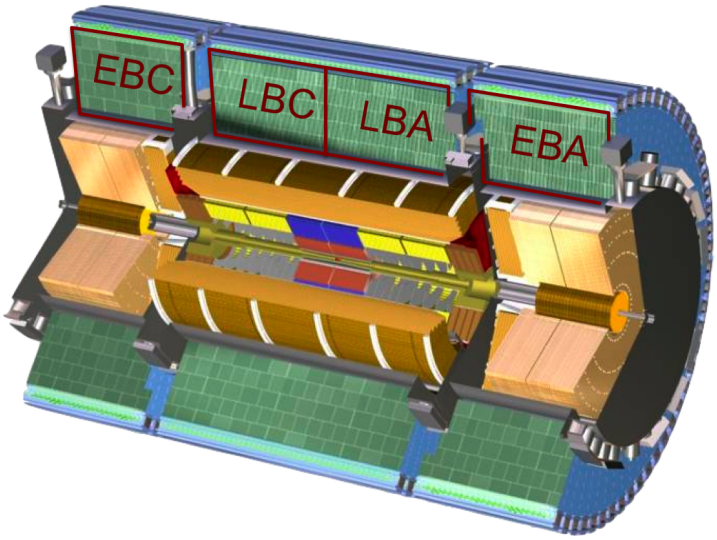


Figura 2.3: Calorímetro de Telhas do ATLAS (com suas 4 partições na cor verde) envolvendo o calorímetro de argônio líquido. O Sistema Magnético e a Câmara de Múons não estão ilustrados nesta figura. Adaptado de [3].

A passagem de partículas ionizadas pelo TileCal produz luz no espectro. Sua intensidade é proporcional à energia depositada pela partícula. A luz produzida se propaga através das telhas para suas bordas onde é absorvida por fibras óticas e deslocada, por reflexão interna total, até os canais de leitura eletrônica ou fotomultiplicadores (em inglês *PhotoMulTipliers Tubes* ou PMTs), onde é convertido em um sinal de corrente. Um PMT pode ser idealizado como um pulso de corrente. A figura 2.4 ilustra a estrutura de absorção e amostragem do TileCal sendo possível identificar tubos fotomultiplicadores.

Os módulos do barril central do detector comportam 45 canais cada, enquanto os módulos dos barris estendidos comportam 32 canais. Radialmente, cada módulo do TileCal ainda é segmentado em três camadas (A, BC e D), conhecidas como células. Estas são formadas pelo agrupamento de fibras até cada PMT. Um esboço da geometria das células é apresentado na figura 2.5. Cada módulo contém 11 linhas transversas de telhas.

2.3.1 Análise *Online* e *Offline*

Existem dois tipos de análises, comuns a todos os experimentos do LHC. O experimento ATLAS, sozinho, gera um volume de dados na ordem de 3,2 PetaBytes ao ano [13], que precisam ser processados e analisados. Nesse caso, é impraticável

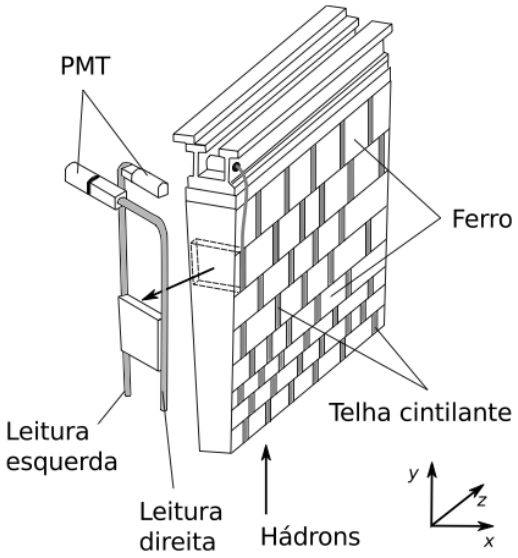


Figura 2.4: Estrutura de absorção e amostragem do TileCal. Extraído de [4]

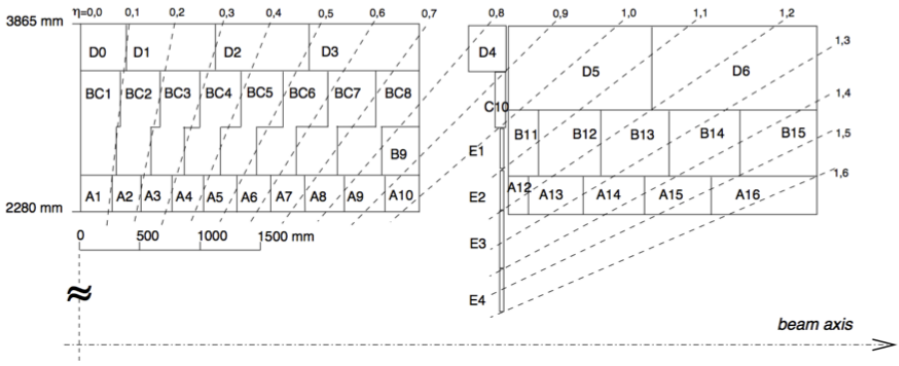


Figura 2.5: Esboço das células do TileCal. Extraído de [3].

armazenar os dados adquiridos pelo experimento em sua totalidade para posterior análise. Por esse motivo, análises referentes a atividades anteriores ao armazenamento em mídia permanente são definidas como *online*. Já as análises que ocorrem a partir do acesso de informações em bases de dados são definidas como sendo *offline*.

As soluções abordadas neste mestrado estão contextualizadas no processo de análise *offline* do TileCal.

As análises *online* e *offline* serão descritas detalhadamente na seção 3.

2.3.2 A colaboração TileCal

O TileCal é composto ainda por diferentes grupos, responsáveis por atividades específicas no experimento. Cada grupo desempenha uma etapa no processo de trabalho do calorímetro. São eles:

DAQ (do inglês, *Data Acquisition* ou aquisição de dados) responsável pela aquisição dos sinais resultantes das interações físicas no detector;

DCS (do inglês, *Detector Control System*, ou sistema de controle do detector) responsável pela manutenção das fontes de alta e baixa tensão do TileCal, placas mãe e sistema de refrigeração;

DQ (do inglês, *Data Quality*, ou qualidade de dados) responsável por avaliar o funcionamento do detector garantindo confiabilidade nos dados adquiridos que serão analisados;

Calibração (do inglês, *Calibration*) responsável pela calibração das células do calorímetro e validação da escala eletromagnética, a fim de corrigir desvios que ocorrem ao longo do tempo devido a desgastes ocorridos pela exposição do detector a altos níveis de radiação;

Computação e Software responsável por realizar a organização e desenvolvimento da simulação, reconstrução, digitalização e desenvolvimento de filtros de alto nível;

Dados e Processamento (do inglês, *Data and Processing*) responsável por coletar informações dos grupos de qualidade de dados e calibração a fim de decidir que alterações nos dados de condições serão realizadas. Responsável também por analisar os dados físicos.

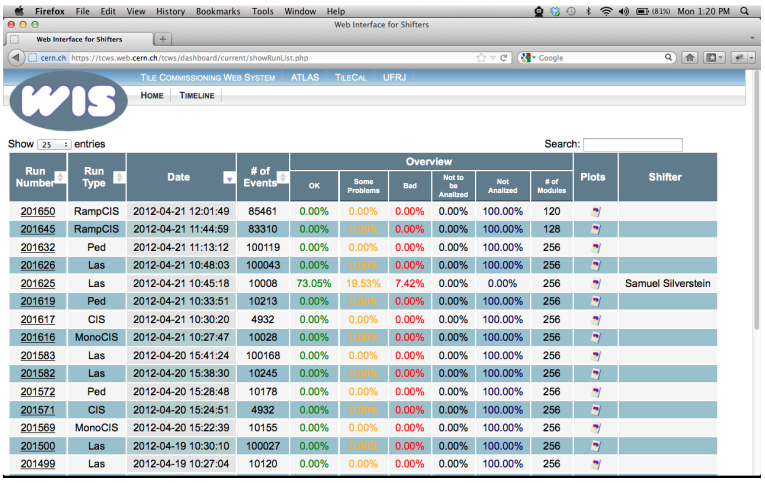
Existem, portanto, seis grupos no TileCal onde o compartilhamento de informações entre eles faz-se necessário: se um PMT não está sendo corretamente alimentado, a qualidade dos dados adquiridos pode ficar comprometida. Tal situação exige o compartilhamento de informações do grupo e DQ, por exemplo.

Capítulo 3

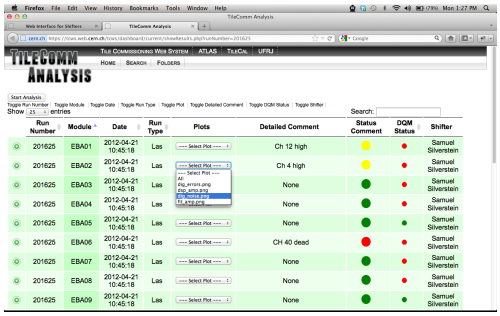
Plataforma web Tile-in-ONE

Diversos sistemas foram desenvolvidos ao longo das diferentes fases do experimento, cada um com um propósito específico [5]. A análise *online* tem início com a aquisição e decisão acerca do armazenamento dos dados adquiridos, que é responsabilidade do grupo TDAQ (do inglês, *Trigger and Data Acquisition*) [14]. A reconstrução dos dados pelo software para análise *offline* ATHENA [15] marca o início da análise *offline*. Em seguida, o DQMF (do inglês, *Data Quality Monitoring System*) [16] é aplicado e gera automaticamente estados para os PMTs, indicando a qualidade do tubo fotomultiplicador e reduzindo a quantidade de canais que precisam ser analisados. Nesta etapa, gráficos são gerados para auxiliar o grupo de qualidade de dados do TileCal em sua análise *offline*. Um sistema web foi desenvolvido para integrar os resultados gráficos gerados durante a etapa de reconstrução, guiando o grupo de qualidade de dados em suas análises. A figura 3.1 ilustra sistemas integrados por uma única interface web: a figura 3.1(a) corresponde a lista de dados reconstruídos e quais gráficos já estão disponíveis para a colaboração, indicando que a atuação do ATHENA foi concluída; a figura 3.1(b) exibe informação detalhada sobre um determinado módulo, acessado pela interface ilustrada anteriormente. Através deste sistema web é possível inserir comentários relacionados a performance do detector; e a figura 3.1(c) ilustra alguns gráficos gerados durante a etapa de reconstrução dos dados.

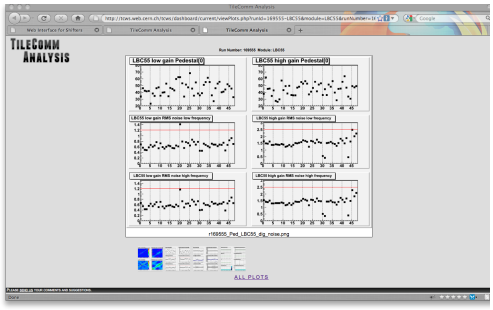
Para uma referência histórica, a lista de canais definidos como defeituosos pela colaboração (*Bad Channels List*) é armazenada no banco de dados de condicionamento, comum a todos os experimentos que compõem o LHC, o COOL DB [17]. Os sistemas MCWS (do inglês, *Monitoring & Calibration Web System*) [18] e DCS (do inglês, *Detector Control System*) [19] apoiam o grupo de qualidade de dados na análise dos quase 10.000 PMTs do calorímetro de telhas. O MCWS foi desenvolvido para exibir a lista de canais defeituosos de maneira gráfica e permite que comentários sobre o estado do detector sejam compartilhados. O sistema web DCS foi desenvolvido para monitorar as fontes de alta tensão que alimentam os PMTs.



(a) Lista dos dados adquiridos.



(b) Estados gerados automaticamente pelo DQMF e acesso a informações detalhadas.



(c) Gráficos gerados para um determinado módulo.

Figura 3.1: *Dashboard Web System*, desenvolvido em 2010 para integrar as os dados adquiridos e as análises do grupo de QD do TileCal. Mais detalhes em [5]

É evidente que o cenário descrito é composto por diversas ferramentas (web) para apoiar em diferentes aspectos as análises e operação de forma mais eficiente do TileCal. Tais sistemas foram desenvolvidos em diferentes fases do detector, envolvendo diferentes colaboradores, que não necessariamente ainda encontram-se envolvidos em atividades da colaboração. Além disso, a manutenção de tais ferramentas não é garantida pelos mesmos colaboradores contribuintes. Para a colaboração, seria mais fácil reunir todas as ferramentas existentes em uma única interface, que utilizasse preferencialmente a mesma tecnologia, permitindo inclusive, reutilização de códigos fonte. Tais requisitos foram contemplados com o projeto e desenvolvimento da Plataforma web Tile-in-ONE.

3.1 Fluxo de dados

A Plataforma Tile-in-ONE foi projetada para integrar as diferentes análises realizadas pela colaboração TileCal. Ao fornecer uma estrutura onde os colaboradores podem desenvolver códigos fonte diretamente através da web, ela integra diferen-

tes ferramentas em um único lugar. Isso garante a escolha da mesma tecnologia para todas as ferramentas desenvolvidas (linguagem Python [20]), o que torna a manutenção da plataforma mais fácil. Além disso, existe o incentivo natural para reutilização de códigos, uma vez que os desenvolvimentos encontram-se disponíveis para toda colaboração. Outra funcionalidade oferecida pela plataforma é o encapsulamento em pacotes de configurações para acessar bases de dados importantes para as análises. Isso permite que colaboradores novos não percam tempo aprendendo a acessar a informação, mas foquem na análise propriamente dita.

A figura 3.3 ilustra o fluxo de dados do Tile-in-ONE. A plataforma oferece a funcionalidade de desenvolvimento de códigos pela qual os colaboradores podem escrever seus próprios programas. Também é permitida a edição de códigos salvos ou escritos por outros usuários. Uma vez que o desenvolvimento do código é concluído, o desenvolvedor pode submeter seu código que será manipulado no lado do servidor. Este, por sua vez, irá direcionar o código em questão para uma outra máquina (nesse contexto, máquinas *slaves*), dependendo dos dados identificados pela plataforma que o código fonte a ser executado deseja acessar. Cada máquina *slave* está configurada para acessar um determinado repositório de dados utilizado para as análises da colaboração. Isso permite que os desenvolvedores se abstraiam de configurações de acesso a tais repositórios. A motivação para efetuar a execução do código fonte em outra máquina que não no servidor é evitar que ocorra sobrecarga no servidor principal e o mais importante: caso o código fonte falhe durante a execução, o servidor onde a plataforma está hospedado não fica comprometido. A figura 3.3 ilustra ainda, em vermelho, onde esta tese de mestrado irá atuar. O objetivo é criar uma etapa antes da submissão de novos códigos para máquinas *slave*, atuando, desta forma, como uma camada de segurança cujo intuito é prever quais códigos vão falhar sem a necessidade de executá-los. Esta camada consiste em um novo desenvolvimento e portanto, no momento, qualquer código pode ser submetido para ser executado na máquina *slave*.

Uma vez que o servidor web escolhe uma máquina *slave* esta tenta executar o código fonte submetido e retorna para o servidor web, de maneira estruturada, o que ocorreu durante a execução. Neste momento, o desenvolvedor do código fonte pode utilizar objetos gráficos também fornecidos pela plataforma para exibir os resultados recuperados através de seus códigos fonte. Enfim, o código fonte, a resposta obtida com a execução na máquina *slave* e os objetos gráficos são encapsulados em *Plugins* e disponibilizados em *Dashboards* para toda colaboração.

A figura 3.2 ilustra a infraestrutura atual da plataforma web Tile-in-ONE. Destaca-se em vermelho onde este mestrado pretende atuar.

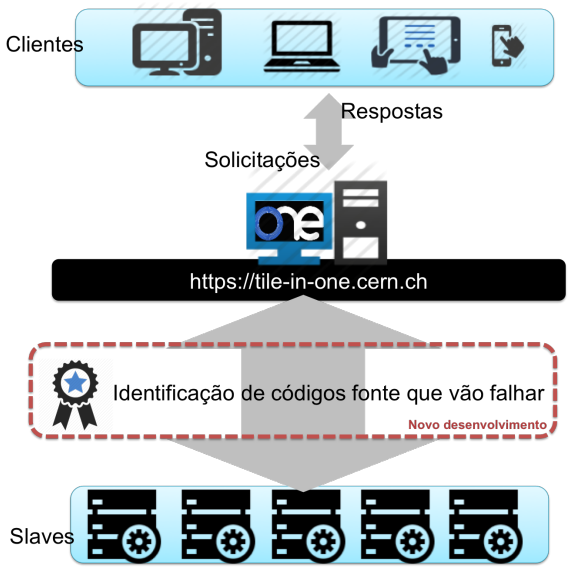


Figura 3.2: Esquemático representando a infraestrutura da plataforma Tile-in-ONE. Em vermelho, a nova camada a ser desenvolvida.

3.2 Novo desenvolvimento

A plataforma Tile-in-ONE está atualmente sendo utilizada pelos grupos de calibração e qualidade de dados do TileCal. Uma preocupação que surgiu ao longo do ano de 2014 foi a execução bem sucedida de códigos fonte sem avaliação prévia. Cada código que enfrenta falhas durante sua execução na máquina *slave* acaba consumindo recursos computacionais que podem comprometer o fluxo de dados da plataforma como um todo. A solução para tal questão levantada é estudada neste mestrado, fazendo uso de técnicas de mineração de códigos fonte, como abordado no capítulo 4.

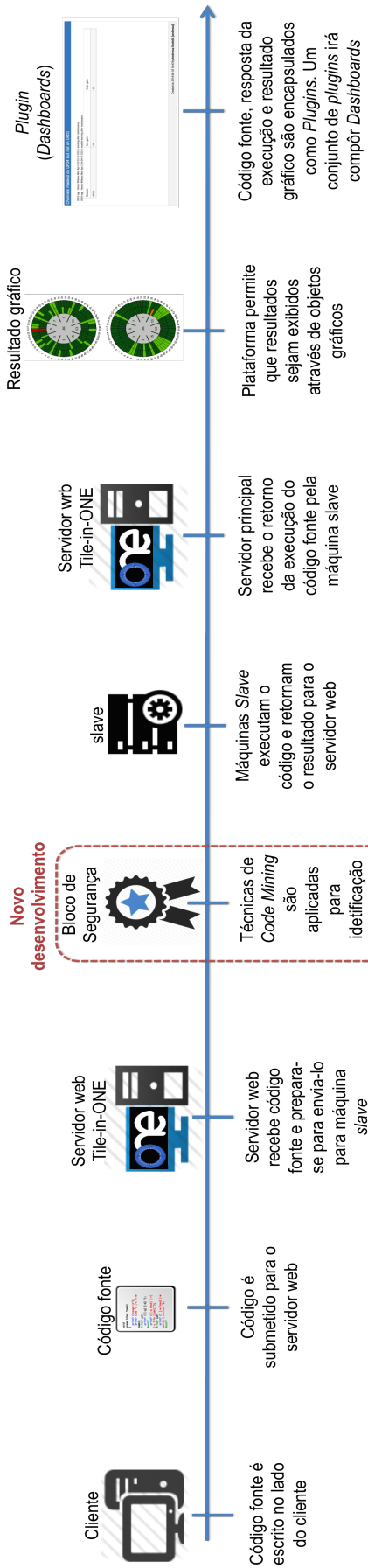


Figura 3.3: Fluxo de dados da plataforma web Tile-in-ONE.

Capítulo 4

Mineração de códigos fonte para identificação de falhas

Em desenvolvimento de software é comum observar que quanto maior for o tempo de permanência de um erro no código mais custosa torna-se a sua correção e manutenção de um projeto [21]. Falhas em códigos fonte são definidas como "Imperfeições durante o desenvolvimento de software que como consequência impedem que o programa atinja as expectativas desejadas." [22]. Entende-se ainda que muitos defeitos são introduzidos durante o processo de desenvolvimento como um todo, não somente no seu início ou fim. Assim sendo, identificar falhas antecipadamente torna-se parte essencial em desenvolvimento de software, que significa qualidade na entrega final do produto. [23].

O SDLC (do inglês, *software development life cycle* ou ciclo de vida do desenvolvimento de software) [24] indica que o desenvolvimento de software pode ser descrito em cinco etapas:

1. Levantamento de Requisitos
2. Projeto
3. Implementação
4. Teste
5. Implantação

A figura 4.1 ilustra em quais etapas do ciclo de vida de desenvolvimento do software são inseridos erros em projetos. Entidades presentes na área de engenharia de computação (tais como a IBM [25] e HCL [26]) confirmam que cerca de 35% das falhas identificadas em códigos fonte são inseridas na fase de implementação [6].

Atualmente, diferentes técnicas tem sido utilizadas para identificação antecipada de erros em códigos fonte. A análise estática, por exemplo, é bastante utilizada na

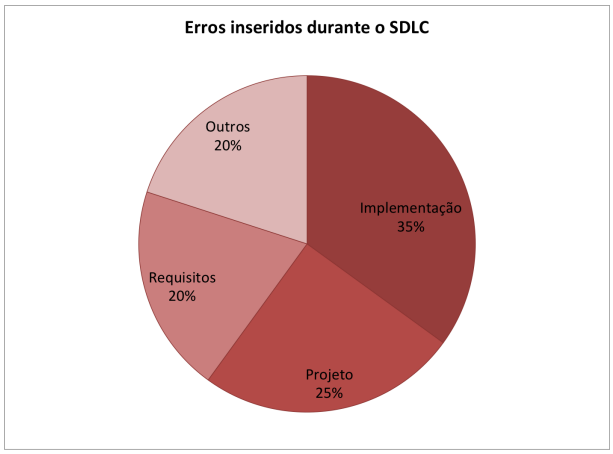


Figura 4.1: Estimativa do percentual de erros inseridos durante fases do SDLC. (Fonte: *Computer Finance Magazine*. Dados extraídos de [6])

área de programação desde a década de 80. As características desta análise são abordadas em 4.1.1. Em conjunto com analisadores estáticos, pode-se aplicar técnicas de mineração de dados, fazendo uma combinação entre a área de engenharia de software e inteligência computacional. No artigo [27], publicado em 2007, os autores abordam como a mineração de dados pode contribuir com a área de engenharia de software, na medida que técnicas de inteligência computacional, quando aplicadas, melhoram a qualidade de projetos. Nesta mesma direção, a MSR (em inglês, *Mining Software Repositories* ou mineração de repositórios de códigos) passa a ser amplamente abordada devido a disponibilidade gratuita de inúmeros repositórios públicos de controle de versionamento, rastreamento de erros inseridos e até mesmo documentações [28].

Aplicar mineração de dados em códigos fonte desperta interesse na comunidade científica, uma vez que códigos são tipicamente estruturados e semanticamente ricos em termos de construtores de programação (tais como variáveis, funções, objetos estruturados e rotinas bem definidas). Os objetivos são diversos: vão desde a tentativa de descobrir o que um determinado software faz (sem necessariamente executá-lo) até manutenção de projetos ou análises de desenvolvimento [29].

Como citado, a identificação antecipada de erros inseridos durante a implementação de códigos fonte é atraente do ponto de vista de qualidade de produto. A utilização de mineração de códigos é uma das aplicações mais ativas em engenharia de software atualmente. Neste caso, o objetivo é obter ferramentas que não só identifiquem falhas (ou *bugs*) mas também sua localização exata nas linhas de códigos do projeto, de tal forma que o seu conserto seja facilitado [29]. Para alcançar esse objetivo, uma direção de bastante destaque consiste na mineração de padrões, que uma vez pré-definidos, são aplicadas em diversos repositórios de projetos já existentes com o objetivo de encontrar anomalias comuns que violam regras ([30], [31], [32]).

Outro objetivo dominante na mineração de códigos é a tentativa de identificar

fragmentos clonados. A não reutilização de códigos dificulta a manutenção do software e ainda, pode proliferar *bugs* em diferentes partes de um projeto.

A seção 4.1 aborda quais são os atributos comumente utilizados quando mineração de dados é aplicada a códigos fonte. Esta seção é composta por duas sub-seções (4.1.1 e 4.1.2) que abordam em que situações tais características foram utilizadas na aplicação de mineração de códigos neste mestrado; A seção 4.2 investiga a aplicação de mineração de dados em códigos fonte no CERN.

4.1 Seleção de atributos

Uma revisão feita por Heckman [33] com 21 artigos sobre mineração de dados em códigos fonte, identifica cinco categorias comumente utilizadas como entradas:

1. *Alert Characteristics* (AC): abordagem que leva em consideração os alertas gerados por analisadores estáticos (sub-seção 4.1.1);

2. *Code Characteristics* (CC): abordagem considera medidas estatísticas e de qualidade do código fonte atributos de entrada (sub-seção 4.1.2);

3. *Source code repository metrics* (SCR): atributos retirados de repositórios de códigos. Nesta abordagem, leva-se em consideração versionamentos (histórico de *bugs* inseridos no projeto), comentários de atualizações e documentação sobre requisitos por exemplo;

4. *Bug database metrics* (BDM): Neste caso, existe uma base de dados para o projeto contendo todos as falhas já surgidas e consertadas. O objetivo dessa abordagem é identificar padrões que podem voltar a acontecer no projeto;

5. *Dynamic analyses metric* (DA): esta abordagem armazena resultados obtidos com análise dinâmica de códigos, mediante diversas execuções (ver seção 4.1.1).

Tais categorias são descritas detalhadamente em [34].

Este mestrado utiliza as duas primeiras abordagens, descritas nas próximas subseções.

4.1.1 Analisadores Estáticos

Em engenharia de software existem dois tipos de análise (complementares) que podem ser feitas para validar e testar o produto em concepção [35]:

- **Estática**: neste caso apenas a estrutura do código é analisada, mas o código não é executado;

- **Dinâmica**: para esta análise, é preciso levantar um plano de testes, executá-los e avaliar os resultados. Ou seja, é necessário executar o código diversas vezes;

Como o objetivo deste mestrado é analisar códigos fonte sem executá-los, utilizar analisadores estáticos torna-se atraente, por definição.

Na década de 80, diversas ferramentas que aplicam análise estática de código começaram a ser desenvolvidas para auxiliar desenvolvedores na implementação de códigos com menos *bugs* [36]. Mas logo um inconveniente surge [37]: a quantidade de alertas gerados pelas ferramentas de análise estática de código (SCAT, do inglês, *Static Code Analysis Tools*) é grande e muitos deles não são acionáveis. Diz-se acionável um alerta que realmente vai impedir o sucesso na execução de um determinado código. Essa classificação é feita manualmente pelo desenvolvedor (que neste caso é o especialista): se o especialista entende que determinado alerta é importante e precisa ser consertado antes da execução do programa, tal alerta é definido como *acionável*. Caso contrário, ele é definido como *não acionável*([38], [39]).

Empiricamente, existem cerca de 40 alertas para cada mil linhas de código (KLOC, do inglês, *Kilo Lines of Code*) [39]. Desta estimativa, podem ser não acionáveis 30-100%, como observado em [40], [41], [39], [42], [43], [44]. Tais números desestimulam desenvolvedores e projetistas a aplicar ferramentas de análise estática de código no processo de desenvolvimento de software, embora entende-se a sua importância no quesito qualidade adquirida. Por outro lado, um alerta definido pelo especialista como *acionável* nem sempre vai impedir a execução bem sucedida do código. Ou seja, se por um lado os analisadores estáticos geram muitos alertas (*não acionáveis*), nada garante que o alerta *acionável* indica que o programa contém *bugs* e vai falhar durante a tentativa de execução. Em outras palavras, um analisador estático pode não gerar nenhum alerta e mesmo assim, durante a tentativa de execução, o código ainda pode falhar.

Para este mestrado, analisadores estáticos serão utilizados em uma segunda etapa, já que a classificação dos alertas gerados não significa exatamente que o código não pode ser executado. Uma vez que um código for classificado como *não executável*, o objetivo seguinte será identificar quais alertas gerados por um analisador estático são acionáveis, para guiar o desenvolvedor no seu conserto, antes da execução de tal código.

4.1.2 Medidas estatísticas e de qualidade

Em computação, "Halstead Metrics" [45] foram medidas de qualidade e estatísticas definidas por Maurice Halstead e publicadas em 1977.

As medidas estatísticas (de Halstead) que podem ser extraídas de um código fonte são:

- η_1: quantidade de operadores distintos no código;
- η_2: quantidade de operandos distintos no código;
- N_1: total de operadores;
- N_2: total de operandos;

Destas, outras medidas estatísticas são também definidas por Halstead:

- Vocabulário: $\eta = \eta_1 + \eta_2$;
- Tamanho: $N = N_1 + N_2$;
- Volume: $V = Nlog_2\eta$;
- Dificuldade: $D = \frac{\eta_1}{2}.\frac{N_2}{\eta_2}$;
- Esforço: $E = D.V$;
- Estimativa de tempo de execução (em segundos): $T = \frac{E}{18}$;
- Estimativa de potenciais *bugs*: $B = \frac{V}{3000}$;

Outra característica que pode ser retirada de códigos fonte é a sua complexidade. Define-se como "Cyclomatic Complexity" (do inglês, complexidade ciclomática) [46] o número de decisões que um determinado bloco de código contém mais uma unidade. Este número (também conhecido como *McCabe number*) representa o número de caminhos independentes percorridos durante a execução de um código, quando se monta uma árvore abstrata de sintaxe (AST, do inglês, *abstract syntax tree*) [47].

O índice de manutenabilidade (ou MI, do inglês *Maintainability Index*) é uma medida em software que indica o quão fácil é a manutenção de um determinado código. O MI é calculado de diferentes maneiras, dependendo da abordagem que se deseja utilizar. Entretanto, a fórmula clássica [48] envolve o número de linhas totais em um código fonte (SLOC, do inglês, *source lines of code*), a complexidade ciclomática (neste contexto, CC) e o volume de Halstead (HV):
$MI = 171 - 5.2ln_{HV} - 0.23_{CC} - 16.2ln_{SLOC}$.
Os autores chegaram nesta fórmula através de um número de códigos fornecidos pela HP, escritos em linguagens de programação C e Pascal. Para cada código, um especialista (engenheiro de software) atribuiu uma nota (de 1 a 100) indicando o quão fácil seria fazer alterações ou correções posteriores ao determinado código (quanto maior a nota, mais fácil é sua manutenção). Consequentemente, 40 faixas

de valores foram identificadas e através de uma regressão estatística, eventualmente, a fórmula exposta foi encontrada como um índice de manutenabilidade.

As medidas descritas nesta sub-seção serão utilizadas neste mestrado, mediante análise de quais seriam mais importantes para o problema que se deseja resolver.

4.2 Analisadores estáticos e Mineração de códigos no CERN

O CERN oferece um ambiente para desenvolvimento de softwares bastante particular. Estima-se que centenas de projetos de softwares estão sendo desenvolvidos, com propósitos específicos, desde programas que auxiliam o trabalho da equipe de recursos humanos até os que monitoram o desempenho dos diversos experimentos do colisor de partículas. Apesar do setor de TI do CERN ter estipulado algumas regras para a política de segurança, muitos projetos de software pequenos não as seguem a risca. Isso implica em vulnerabilidades de programas utilizados em ambientes de produção no CERN.

As regras de segurança estipuladas não são tão rígidas para estimular o meio acadêmico no avança das pesquisas científicas. Da mesma maneira, analisadores estáticos para assegurar a qualidade no desenvolvimento de códigos livres não são impostos para a comunidade presente. Existem ferramentas disponíveis e com documentação e suporte da equipe de TI, mas nenhum projeto é obrigado a utilizá-las. No caso deste projeto de mestrado, a linguagem computacional utilizada é o Python, sendo o *PyLint* [49] uma ferramenta de análise estática cuja utilização é incentivada pelo CERN [50].

Recentemente, o setor de segurança em computação do CERN contratou um serviço (*Coverity* [51]) para garantir qualidade no desenvolvimento de novas funcionalidades no seu principal produto de análises físicas (o ROOT [52]). O ROOT é utilizado por cerca de 10.000 pesquisadores e a integridade do software passou a ser vista como algo valioso. Um *bug* perpetuado no ROOT pode ter um impacto muito negativo nas análises de resultados dos dados gerados pelo LHC. Assim que incorporado, o *Coverity* já foi capaz de identificar mais de 40.000 falhas em aproximadamente 50 milhões de linhas de códigos [53]. Um ponto negativo observado por desenvolvedores do ROOT é o tempo necessário para executar o analisador estático: são 28 horas para cada 2 milhões de linhas de código. É válido observar que a ferramenta escolhida para o ROOT não atende as necessidades deste projeto de mestrado: o *Coverity* atende códigos escritos em linguagem C/C++ enquanto os códigos do Tile-in-ONE (seção 3) são implementados em Python. Além disso, deseja-se aplicar mineração de dados em códigos fonte em um ambiente web.

Em física de altas energias não existem artigos que indiquem a aplicação de mineração de dados em códigos fonte.

Capítulo 5

Metodologia

Este capítulo descreve o processo de estudo utilizado para este mestrado.

As seções 5.1 e 5.2 descrevem, respectivamente, o conjunto de dados adquirido e os atributos que foram previamente selecionados; A seção 5.3 descreve os algorítimos de máquina de aprendizado utilizados;

5.1 Aquisição de dados

5.2 Seleção de categorias

5.3 Métodos *ensemble* em Árvores de Decisão

Capítulo 6

Resultados

6.1 AvaliaÃ§Ã£o dos Classificadores

Capítulo 7

Conclusões

Referências Bibliográficas

[1] "CDS - CERN Document Server",
Acessado em janeiro de 2016. `cds.cern.ch`.

[2] "ATLAS Photos",
Acessado em janeiro de 2016. `http://www.atlas.ch/photos/`.

[3] ATLAS COLLABORATION, *ATLAS: Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*, Tech. rep., CERN, 1994, CERN/LHCC 94–43.

[4] ATLAS/TILE CALORIMETER COLLABORATION, *Tile Calorimeter Calorimeter Technical Design Report*, Tech. rep., CERN, 1996, CERN/LHCC 96–42.

[5] BALABRAM, C. M. F. F. F. G. A. S. L., "Web System for Data Quality Assessment of Tile Calorimeter During the ATLAS Operation", *Journal of Physics: Conference Series (JPCS)*, v. 331 - part 4, 2011.

[6] PRAKASH, A., ASHOKA, D., ARADYA, V., "Application of Data Mining Techniques for Defect Detection and Classification". In: *Proceeding of 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications. FICTA 2014. Advances in Intelligent Systems and Computing 327.*, v. 1, pp. 387–395, 2015.

[7] "The European Laboratory for Particle Physics",
Acessado em janeiro de 2016. `http://www.cern.ch`.

[8] "The Large Hadron Collider",
Acessado em janeiro de 2016. `http://lhc.web.cern.ch`.

[9] GRIFFTHS, D., *Introduction to Elementary Particles*. John Wiley & Sons, 1987.

[10] THE ALICE COLLABORATION, "The ALICE experiment at the CERN LHC", *Journal of Instrumentation*, v. 3, n. 08, Aug. 2008.

[11] THE CMS COLLABORATION, "The CMS experiment at the CERN LHC", *Journal of Instrumentation*, v. 3, n. 08, Ago 2008.

[12] SZUMLAK, T., "The LHCb experiment", *Acta Physica Polonica. Series B: Elementary Particle Physics, Nuclear Physics, Statistical Physics, Theory of Relativity, Field Theory*, v. 41, n. 7, pp. 1661–1668, 2010.

[13] "ATLAS *Factsheet*",
Acessado em janeiro de 2016.
`http://www.atlas.ch/pdf/fact_sheet_1page.pdf`.

[14] JENNI, P., NESSI, M., NORDBERG, M., et al., "ATLAS high-level trigger, data-acquisition and controls: Technical Design Report", *Technical Design Report ATLAS. CERN, Genebra*, v. 42, 2003.

[15] P. CALAFIURA AND W. LAVRIJSEN AND OTHERS, "The athena control framework in production, new developments and lessons learned", *Journal of Physics: Conference Series*, pp. 456–458, 2004.

[16] CORSO-RADU, A., HADAVAND, H., HAUSCHILD, M., et al., "Data Quality Monitoring Framework for the ATLAS Experiment at the LHC", *IEEE Xplore Digital Library*, v. 55, 2008.

[17] VERDUCCI, M., "ATLAS conditions database experience with the LCG COOL conditions database project", *Journal of Physics: Conference Series (JPCS)*, v. 119 - part 4, 2008.

[18] SIVOLELLA, A., MAIDANTCHIK, C., "The Monitoring and Calibration Web Systems for the ATLAS Tile Calorimeter Data Quality Analysis", *Journal of Physics: Conference Series 052037, Computing High Energy Physics, New York, USA*, v. 396, pp. 8, May 2012.

[19] MAIDANTCHIK, C., FERREIRA, F., GRAEL, F., "DCS Web System", *Journal of Physics: Conference Series, Computing High Energy Physics, Praga, Rep. Tcheca*, pp. 8, March 2009.

[20] "Python Programming Language",
Acessado em janeiro de 2016.
`http://www.python.org/`.

[21] MCCONNEL, S., "An ounce of prevention", *IEEE software*, May/June 2001.

[22] KUMARESH, S., BASKARAN, R., "Defect Analysis and Prevention for Software Process Quality Improvement", *International Journal of Computer Applications (0975 - 8887)*, v. 8, n. 7, October 2010.

[23] CHILLAREGE, R., BHANDARI, I., CHAAR, J., et al., "Orthogonal Defect Classification - A Concept for In-Process Measurements", *IEEE Transactions on Software Engineering*, v. 18, n. 11, November 1992.

[24] VELMOUROUGAN, S., DHAVACHELVAN, P., BASKARAN, R., et al., "Software Development Life Cycle Model to Improve Maintainability of Software Applications". In: *Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on*, pp. 270–273, Aug 2014.

[25] "IBM",
Acessado em janeiro de 2016.
`http://www.ibm.com/us-en/`.

[26] "HCL",
Acessado em janeiro de 2016.
`http://www.hcl.com/`.

[27] XIE, T., PEI, J., HASSAN, A., "Mining Software Engineering Data". In: *Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th International Conference on*, pp. 172–173, May 2007.

[28] HASSAN, A., "The road ahead for Mining Software Repositories". In: *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pp. 48–57, Sept 2008.

[29] KHATOON, S., LI, G., MAHMOOD, A., "Comparison and evaluation of source code mining tools and techniques: A qualitative approach". v. 17, pp. 459–484, May 2013.

[30] LI, Z., ZHOU, Y., "PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code". In: *In Proc. 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pp. 306–315, ACM Press, 2005.

[31] KRISHNA RAMANATHAN, M., GRAMA, A., JAGANNATHAN, S., "Path-Sensitive Inference of Function Precedence Protocols". In: *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pp. 240–250, May 2007.

[32] CHANG, R.-Y., PODGURSKI, A., YANG, J., "Finding What's Not There: A New Approach to Revealing Neglected Conditions in Software". In: *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, *ISSTA '07*, pp. 163–173, ACM: New York, NY, USA, 2007.

[33] HECKMAN, S., WILLIAMS, L., "A Systematic Literature Review of Actionable Alert Identification Techniques for Automated Static Code Analysis", *Inf. Softw. Technol.*, v. 53, n. 4, pp. 363–387, April 2011.

[34] HECKMAN, S., WILLIAMS, L., "A measurement framework of alert characteristics for false positive mitigation models", *North Carolina State University TR-2008- 23*, October 2008.

[35] FAIRLEY, R., "Tutorial: Static Analysis and Dynamic Testing of Computer Software", *Computer*, v. 11, n. 4, pp. 14–23, April 1978.

[36] TISCHLER, R., SCHAUFLER, R., PAYNE, C., "Static Analysis of Programs As an Aid to Debugging", *SIGSOFT Softw. Eng. Notes*, v. 8, n. 4, pp. 155–158, March 1983.

[37] YUKSEL, U., SÃ¶ZER, H., "Automated Classification of Static Code Analysis Alerts: A Case Study." In: *ICSM*, pp. 532–535, IEEE Computer Society, 2013.

[38] HECKMAN, S., WILLIAMS, L., "A Model Building Process for Identifying Actionable Static Analysis Alerts". In: *Software Testing Verification and Validation, 2009. ICST '09. International Conference on*, pp. 161–170, April 2009.

[39] HECKMAN, S., WILLIAMS, L., "On Establishing a Benchmark for Evaluating Static Analysis Alert Prioritization and Classification Techniques". In: *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, *ESEM '08*, pp. 41–50, ACM: New York, NY, USA, 2008.

[40] AGGARWAL, A., JALOTE, P., "Integrating Static and Dynamic Analysis for Detecting Vulnerabilities". In: *Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01*, *COMPSAC '06*, pp. 343–350, IEEE Computer Society: Washington, DC, USA, 2006.

[41] BOOGERD, C., MOONEN, L., "Prioritizing Software Inspection Results using Static Profiling". In: *Source Code Analysis and Manipulation, 2006. SCAM '06. Sixth IEEE International Workshop on*, pp. 149–160, Sept 2006.

[42] KIM, S., ERNST, M., "Prioritizing Warning Categories by Analyzing Software History". In: *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, pp. 27–27, May 2007.

[43] KIM, S., ERNST, M. D., "Which Warnings Should I Fix First?" In: *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, *ESEC-FSE '07*, pp. 45–54, ACM: New York, NY, USA, 2007.

[44] KREMENEK, T., ENGLER, D., "Z-ranking: Using Statistical Analysis to Counter the Impact of Static Analysis Approximations". In: *Proceedings of the 10th International Conference on Static Analysis*, *SAS'03*, pp. 295–315, Springer-Verlag: Berlin, Heidelberg, 2003.

[45] HALSTEAD, M. H., *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc.: New York, NY, USA, 1977.

[46] MACCABE, T., MCCABE, ASSOCIATES, et al., *Structured testing*. *Tutorial Texts Series*, IEEE Computer Society Press, 1983.

[47] JONES, J., "Abstract Syntax Tree Implementation Idioms". In: *Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003)*, 2003.

[48] OMAN, P., HAGEMEISTER, J., "Metrics for assessing a software system's maintainability". In: *Software Maintenance, 1992. Proceerdings., Conference on*, pp. 337–344, Nov 1992.

[49] "PyLint - Static Analysis Tool for Python",
Acessado em janeiro de 2016. `http://www.pylint.org/`.

[50] "CERN IT Security Recomended Tools",
Acessado em janeiro de 2016. `https://security.web.cern.ch/security/recommendations/en/code_tools.shtml`.

[51] BESSEY, A., BLOCK, K., CHELF, B., et al., "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World", *Commun. ACM*, v. 53, n. 2, pp. 66–75, Feb. 2010.

[52] BRUN, R., RADEMAKERS, F., "ROOT-an object oriented data analysis framework". In: *World Scientific Publishing*, 2005.

[53] "Coverity efects at CERN",
Acessado em janeiro de 2016. `http://www.coverity.com/press-releases/cern-chooses-coverity-to-ensure-accuracy-of-large-hadron-co`

Apêndice A

Alguns casos