UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA CONTRIBUIÇÃO PARA A AVALIAÇÃO DA QUALIDADE DE MODELOS NA FASE DE PROJETO DE SOFTWARE

Odair Moreira de Souza

Orientador: Prof. Dr. Fabiano Cutigi Ferrari

São Carlos – SP

Julho de 2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA CONTRIBUIÇÃO PARA A AVALIAÇÃO DA QUALIDADE DE MODELOS NA FASE DE PROJETO DE SOFTWARE

Odair Moreira de Souza

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador: Prof. Dr. Fabiano Cutigi Ferrari

São Carlos – SP

Julho de 2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Souza, Odair Moreira de
S729c Uma contribuição para a avaliação da qualidade de modelos na fase de projeto de software / Odair Moreira de Souza. -- São Carlos : UFSCar, 2016.
165 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2015.

1. Qualidade de software. 2. Métrica de software. 3. Atributo de qualidade. 4. Avaliação de modelos. I. Título.

UNIVERSIDADE FEDERAL DE SÀO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado do candidato Odair Moreira de Souza, realizada em 13/07/2015.

Prof. Dr. Fabiano Cutigi Ferrari
(UFSCar)

Profa. Dra. Elisa Yumi Nakagawa
(ICMC/USP)

Prof. Dr. Edson Alves de Oliveira Junior
(UEM)

Certifico que a sessão de defesa foi realizada com a participação à distância dos membros Profa. Dra. Elisa Yumi Nakagawa e Prof. Dr. Edson Alves de Oliveira Junior, depois das arguições e deliberações realizadas, os participantes à distância estão de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Odair Moreira de Souza.

Prof. Dr. Fabiano Cutigi Ferrari
Presidente da Comissão Examinadora
(UFSCar)

Dedico este trabalho aos meus familiares, amigos e professores, a todas as pessoas que participaram diretamente e indiretamente dessa etapa da minha vida.

Agradecimentos

Chegou o momento de expressar meus agradecimentos a muitos e tantos adorados familiares, amigos, companheiros e professores.

Primeiramente, quero agradecer a Deus pela oportunidade concedida em adquirir novos conhecimentos, por ter proporcionado saúde para viver tudo que vivi, por ter iluminado o meu caminho, me dado força, paciência, compreensão e por estar sempre ao meu lado, me auxiliando a superar os obstáculos e dificuldades encontrados na vida.

Agradeço ao Prof. Dr. Fabiano Ferrari, pela amizade, pelo apoio e paciência durante a elaboração deste trabalho, pelas orientações, pela confiança, pelo entendimento dos meus limites e a cada incentivo, sempre demonstrando muita dedicação e profissionalismo.

A todos os professores do DC–UFSCar pelas lições aprendidas, em especial a Profa. Dra. Sandra Fabbri, pelo conhecimento transmitido durante as disciplinas, seminários e conversas no LaPES, além do apoio e motivação em momentos difíceis.

Com muito amor, sou eternamente grato à minha avó, a Sra. Izaura Moreira, por estar sempre ao meu lado, orientando-me e apoiando-me em todos os momentos. Agradeço ao amor dedicado, aos conselhos e explicações sobre a vida. Não possuo palavras para agradecer por tudo que me fez.

Um agradecimento muito especial à minha namorada, amiga, companheira. Jaqueline agradeço pela compreensão, alegria, atenção, amor, carinho, encorajamento e pelo apoio de todos os momentos. Sem você eu não sei se teria conseguido.

Agradeço aos meus familiares, tia Lourdes e primas Aline e Ana, especialmente ao meu irmão Tiago e sua família, Carol e Allyne. A toda a família Karvat.

Ao amigo Allysson, pelo incentivo, motivação e amizade.

A todos os amigos do LaPES que dividiram bons momentos: Thiago, Gaspar, Bento, Ana, Cleiton, Elis, Ivan, Gastaldi, Matheus, Fábio, Erik, Anderson, Sandra, Fabiano, Kamila e Augusto. Ao André DT pela motivação em momentos de dificuldades e ao

Abade pelo companheirismo e motivação.

Aos Amigos: Lucas Porto, Elias, Stéfano, Victor, Rafael Sanches, Tomé, Kanegae, Bruno, Mirela, Fernando, Guido e João. E a todos os meus amigos do mestrado, pelas horas e horas de estudos compartilhados e pelos vários momentos de lazer.

A todos os amigos que conviveram na "Rep.BigTable": Rafael, Thiago, Guilherme, Afonso e Rodrigo. E ao pessoal do futebol e dos churrascos.

Agradeço aos membros da banca examinadora pela disposição em analisar esta dissertação e pelas valiosas contribuições para tornar este trabalho melhor.

À Universidade Federal de São Carlos – UFSCar, ao Programa de Pós–Graduação em Ciência da Computação e ao LaPES – Laboratório de Pesquisa em Engenharia de Software pela infraestrutura e suporte para a realização deste trabalho.

Agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Instituto Nacional de Ciência e Tecnologia – Sistemas Embarcados Críticos (INCT–SEC), pelo apoio financeiro parcial em bolsa durante esse trabalho.

A todos o meu sincero agradecimento, muitíssimo obrigado!

"O futuro não é um lugar onde estamos indo, mas um lugar que estamos criando. O caminho para ele não é encontrado, mas construído, e o ato de fazê-lo muda tanto o realizador quanto o destino."

Antoine de Saint-Exupéry

Resumo

Contexto: A engenharia de software tem por finalidade proporcionar o desenvolvimento de software com alto nível de qualidade em curto tempo e baixo custo. Nesse contexto, a garantia de qualidade de produto passa por todas as etapas do processo de desenvolvimento, evitando gerar alto custo em manutenção corretiva em fases avançadas do desenvolvimento. Na etapa de projeto (do inglês, *design*), uma das maneiras de antecipar a detecção de problemas estruturais, comportamentais e reduzir o retrabalho, é por meio da aplicação de métricas em modelos de projeto. **Objetivo:** Este trabalho teve como objetivo prover um processo sistemático e automatizado para a avaliação de modelos de projeto de software baseada em métricas internas e atributos externos de qualidade. Um *framework* conceitual, intitulado QM^2, e ferramentas automatizadas foram propostos, os quais incluem as principais atividades e recursos necessários para a avaliação do nível de qualidade dos projetos durante a etapa de *design*. **Metodologia:** Para a definição do QM^2, foram realizados estudos para a identificação das atividades essenciais para a avaliação de *design*, abordando modelos de qualidade, métricas internas e recursos reutilizáveis de software. Para a avaliação dos conceitos e ferramentas desenvolvidas, foram avaliados modelos de projeto de sistemas embarcados robóticos representados em UML. **Resultados:** Estabeleceu-se uma sequência de passos para a avaliação do nível de qualidade dos modelos de software. Realizou-se a automatização do QM^2, incluindo uma ferramenta de extração de métricas e gestão de projetos de avaliação. **Conclusão:** A simples definição e extração de métricas não assegura a qualidade de um projeto de software, pois a interpretação dos valores deve estar associada a atributos externos de qualidade. Nesse sentido, neste trabalho a avaliação de *design* é realizada por meio das extrações e cálculos das métricas dos modelos. O nível de qualidade de um modelo é obtido por meio da associação entre medidas internas computadas e atributos externos de qualidade.

Palavras-chave: Qualidade de Software, Métrica de Software, Atributo de Qualidade, Avaliação de Modelos

Abstract

Context: Software engineering aims to provide software development with high quality in short time and low cost. In this context, product quality assurance goes through all phases of development process, avoiding the generation of high cost in corrective maintenance in advanced phases of development. In the design phase, a mean of anticipating the detection of structural and behavioural problems and reduce rework is by applying metrics in design models. **Objective**: This work aimed to provide a systematic and automated process for the evaluation of software design models based on internal metrics and external quality attributes. A conceptual framework, entitled QM^2, and automated tools have been proposed, which include the main activities and resources needed for assessing the quality level of models during the design phase. **Method:** For the definition of QM^2, studies have been conducted to identify the essential activities for design evaluation, addressing quality models, internal metrics and reusable software resources. For the evaluation of the the framework and tools developed, were addressed models of a robotic embedded system represented in UML. **Results:** Was established a sequence of steps to evaluate the quality level of software models. Was performed the automation of the QM^2 framework, including tool for metrics extraction and the management of assessment projects. **Conclusion:** The simple definition and extraction of metrics do not ensure the quality of a software project, because the interpretation of values should be associated with external quality attributes. In that sense, in this work the assessment of design is performed by means of extraction and calculation of model metrics. The quality level of a model is obtained through the association between internal measures computed and external quality attributes.

Keywords: Software Quality, Software Metric, Quality Attribute, Model Evaluation

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

API – *Application Programming Interface*

BDD – *Block Definition Diagram*

BI – *Business Intelligence*

BPD – *Business Process Diagram*

BPMN – *Business Process Model and Notation*

CBSE – *Component-Based Software Engineering*

CCF – *Component Complexity Factor*

DAO – *Data Access Object*

EPL – *Eclipse Public License*

IBD – *Internal Block Diagram*

JSF – *Java Server Faces*

KLOC – *Kilo Lines of Code*

KPI – *Key Performance Indicator*

LOC – *Line of Code*

LOV – *List Of Values*

MBD – *Model Based Design*

MBSE – *Model Based Systems Engineering*

MDA – *Model-Driven Architecture*

MDD – *Model-Driven Development*

MOF – *Meta Object Facility*

MOOD – *Metrics for Object Oriented Design*

MS – *Mapeamento Sistemático*

MVC – *Model-View-Control*

NAS – *Number of Associations*

OMG – *Object Management Group*

OOSEM – *Object-Oriented Systems Engineering Method*

OOSE – *Object-Oriented Software Engineering*

OO – *Orientado a Objeto*

OPF – *Open Process Framework*

ORM – *Object/Relational Mapping – Mapeamento Objeto-Relacional*

PU – *Processo Unificado*

QM2 – *Quality of Models based on Metrics*

QMOOD – *Quality Model for Object-Oriented Design*

RUP SE – *Rational Unified Process for Systems Engineering*

RUP – *Rational Unified Process*

SCQM – *Samsung software Component Quality evaluation Model*

SGBD – *Gerenciador de Banco de Dados*

SQuaRE – *Software product Quality Requirements and Evaluation*

SYSMOD – *System Modelling Process*

SaaS – *Software as a Service*

SysML – *Systems Modeling Language*

UML – *Unified Modeling Language*

V&V – *Verificação e Validação*

XMI – *XML Metadata Interchange*

XML – *eXtensible Markup Language*

Sumário

Capítulo 1

Introdução

1.1 Contextualização

A necessidade de obtenção de altos níveis de qualidade tem concentrado esforços da academia e da indústria de software, em relação à qualidade do processo de desenvolvimento e qualidade dos produtos criados (JABANGWE et al., 2015). Nesse contexto, a qualidade de produto se define como a conformidade entre o software desenvolvido e as características esperadas (PRESSMAN, 2010). Essas características podem ser vistas como os pontos a serem atingidos em um determinado nível de qualidade, para que o produto atenda às necessidades dos usuários (ROCHA et al., 2001). Com respeito à qualidade de processo, ela pode ser vista como um indicativo de que a organização que o executa é capaz de produzir consistentemente software de qualidade (CMMI Product Team, 2010).

De acordo com Rocha et al. (2001), o nível de qualidade de um produto ou de um processo de software pode ser obtido por meio da utilização de métricas, que auxiliam na organização, no monitoramento, na identificação e na prevenção de falhas. Ressalta-se que essas métricas podem auxiliar na avaliação de processos e produtos finais.

Destaca-se que a qualidade dos produtos de software é um dos principais requisitos considerados durante o processo de desenvolvimento. Com a finalidade de alcançar um nível de qualidade aceitável, utilizam-se as atividades complementares às do processo de desenvolvimento, agrupadas sob o nome de Garantia de Qualidade de Software. Nesse grupo, citam-se as atividades de Verificação e Validação (V&V), assim como a avaliação dos artefatos intermediários desenvolvidos por meio de métricas, normalmente para medir o tamanho, complexidade e facilidade de compreensão.

Diante dos vários modelos de processos de desenvolvimento de software, até então propostos na literatura, destacam-se os que têm como principal recurso modelos de projeto (do inglês, modelos de *design*). Por exemplo, o *Object Management Group* (OMG) apresenta um conjunto de técnicas para Engenharia de Sistema Baseada em Modelos (do inglês, *Model Based Systems Engineering* – MBSE), tal como o Desenvolvimento Dirigido por Modelos (do inglês, *Model-Driven Development* – MDD), o qual propõe uma abordagem que utiliza modelos para o desenvolvimento. Uma outra abordagem, proposta por Larman (2004), utiliza o Processo Unificado (PU), enfatizando a fase de projeto por meio da modelagem em UML com a aplicação de padrões de projeto. No contexto de Sistemas Embarcados, cita-se ainda o processo *System Modelling Process* (SYSMOD) (WEILKIENS, 2008) que utiliza como base os diagramas fundamentais da SysML/UML.

Tendo em vista o alto custo da correção de defeitos de software em fases avançadas do desenvolvimento ou após o software ser colocado em operação, é clara a importância de realizar avaliações de qualidade desde o início do ciclo de desenvolvimento (DUGERDIL; NICULESCU, 2014). Uma das formas de antecipar a detecção de problemas estruturais e comportamentais do software consiste na aplicação de métricas nos artefatos – ou seja, modelos – produzidos durante a etapa de projeto (LIU et al., 2000). Os valores obtidos para as métricas podem ser ponderados e associados a atributos externos de qualidade, que compõem modelos de qualidade que são empregados como referência na avaliação de processos e produtos de software (JABANGWE et al., 2015).

Na literatura são encontradas algumas propostas de modelos de qualidade e definições de métricas para avaliação da qualidade de modelos em nível de projeto. Citam-se, como exemplos, o *Metrics for Object-Oriented Design* (MOOD) que é um conjunto de métricas para abordar herança, polimorfismo, acoplamento e coesão definido por (ABREU; CARAPUCA, 1994), o *Quality Model for Object-Oriented Design* (QMOOD) proposto por Bansiya e Davis (2002) e o modelo de Ahrens et al. (2011). O QMOOD baseia-se na norma de qualidade de produto ISO/IEC 9126-1 (ISO/IEC, 2001). O modelo de Ahrens et al. (2011), por sua vez, contempla um conjunto de características de qualidade e as correlaciona com atributos e métricas internas.

Para possibilitar a avaliação quantitativa de níveis de qualidade para um determinado artefato de software tendo como base métricas, existe a necessidade de se definir um modelo de qualidade factível para um determinado domínio e de se associar seus atributos de qualidade com métricas computáveis (BRIAND; WÜST, 2002). Nesse contexto, definem-se nas próximas seções o problema a ser tratado neste trabalho e o objetivo pretendido.

1.2 Motivação e Definição do Problema

A qualidade não é um problema que se restringe ao domínio de engenharia de software, estende-se para além dos limites de qualquer produto ou processo de software (GOYAL; JOSHI, 2014). De acordo com Kitchenham e Pfleeger (1996) qualidade pode ter definição diferente para diferentes pessoas, pois é altamente dependente do contexto e não existe uma definição universal aceitável, ou conjunto de medidas de qualidade comum para todos os contextos. Medir a qualidade é a chave para o desenvolvimento de software, pois possibilita a avaliação dos produtos de software e a identificação dos pontos de melhoria da qualidade dos mesmos (KUMAR et al., 2012).

Nesse contexto, métricas têm sido amplamente utilizadas para avaliar e gerenciar a qualidade do software (BRIAND et al., 2000). As métricas podem medir cada fase do desenvolvimento e muitos aspectos referentes ao produto. As métricas relacionadas a qualidade propõem estratégias sobre como realizar análises de projetos e podem ser integradas de maneira continua em projetos e desenvolvimentos, assim como as métricas de software, que são utilizadas para auxiliar na construção de arquiteturas de sistemas (THIRUGNANAM; SWATHI.J.N, 2010). As métricas também podem auxiliar na compreensão dos artefatos e implementação do software (GEETIKA; SINGH, 2014).

Entretanto, a simples definição de métricas de qualidade não assegura a sua aplicação. Isto deve-se à dificuldade de se coletar e apresentar os resultados referentes aos diferentes projetos de desenvolvimento (ROCHA et al., 2001). Além disso, de acordo com Lanza et al. (2010), os resultados obtidos são difíceis de serem interpretados a partir de medição dos artefatos produzidos, em particular na etapa de projeto (*design*). Esta dificuldade ocorre devido a ausência de referências para os valores obtidos como resultados.

Ressalta-se a importância de avaliar a qualidade em nível de projeto, sendo que esta atividade auxilia na antecipação e correções de defeitos (LIU et al., 2000). Os resultados das métricas, quando combinados, podem indicar problemas no projeto. Para isso, devem ser interpretados com base em valores de referência, também chamados de limiares (do inglês, *thresholds*) (ROCHA et al., 2001). Os limiares são intervalos predeterminados que são normalmente utilizados para representar o nível de qualidade de um atributo. Uma das maneiras de estabelecer interpretações precisas é por meio dos atributos de um modelo de qualidade, com os quais as métricas e limiares são associados, permitindo a obtenção de níveis de qualidade com respeito aos atributos considerados (JABANGWE et al., 2015).

Neste cenário, identificou-se a carência de um processo para apoiar desde a definição de

um modelo de qualidade adequado para um contexto específico, passando pela associação desse modelo com métricas internas e chegando à atribuição de níveis de qualidade para artefatos de nível de projeto. Além disso, considera-se relevante um apoio automatizado às diversas etapas desse processo, como forma de viabilizar a medição dos artefatos, o processamento e as interpretações dos valores obtidos e a produção dos resultados da avaliação (LINCKE et al., 2008). Portanto, com base nos pontos levantados define-se os objetivos deste trabalho na próxima seção.

1.3 Objetivo

O objetivo principal deste trabalho é prover um processo sistemático para avaliar o nível de qualidade dos modelos de software na fase de projeto[1]. A avaliação de qualidade é realizada com o apoio de métricas internas, que são extraídas diretamente de artefatos em nível de projeto, e as suas associações com atributos externos de qualidade. Relacionado ao objetivo principal, apresenta-se a proposta de um *framework* conceitual para o processo de avaliação, indicando quais são as principais atividades e recursos prioritários para a avaliação dos artefatos. O *framework* visa a apoiar a avaliação de artefatos de projeto independentemente da definição do modelo de qualidade, da definição de métricas internas, da personalização dos intervalos para os níveis de qualidade e da geração de relatórios dinâmicos. O trabalho também inclui a criação e reutilização de recursos automatizados para apoiar atividades presentes no *framework* proposto.

1.4 Metodologia e Desenvolvimento do Trabalho

Para atingir o objetivo proposto, foi necessário realizar uma sequência de atividades, para a identificação dos passos essenciais para a avaliação de modelos. Dentre essas atividades, realizou-se um estudo comparativo de modelos de qualidade de software para identificar as similaridades dos atributos do modelo de Ahrens et al. (2011) com os modelos tradicionais e amplamente utilizados. Essa etapa do trabalho levou à conclusão de que os modelos de qualidade devem ser personalizados para serem aplicados em domínios específicos, tais como, o domínio de aplicação desse estudo.

Com essa preocupação em mente, a sequência do trabalho consistiu em uma série de discussões que levou à definição de um conjunto de atividades que devem ser realizadas,

[1] *design* ou diagramas da fase de projeto.

para que a avaliação de qualidade almejada pudesse se concretizar. Atividades essenciais foram identificadas, tais como: (i) a análise do domínio de aplicação; (ii) a definição (ou revisão) do modelo de qualidade adotado; (iii) a análise da linguagem de modelagem empregada na criação dos artefatos[2] a serem avaliados; e (iv) a definição (ou revisão) de métricas internas e seu respectivo mapeamento com os atributos de qualidade. Com base nesse estudo elaborou-se a proposta de um *framework* conceitual para orientar avaliações do nível de qualidade dos modelos.

Uma vez definidas as atividades essenciais, foram implementados recursos automatizados para viabilizar a medição dos modelos e atribuição de níveis de qualidade. A implementação incluiu a reutilização de APIs (do inglês, *Application Programming Interface)* e de uma plataforma de código livre para geração de relatórios de qualidade de projetos de software.

Para a avaliação da viabilidade de aplicação do *framework* proposto e do ambiente automatizado, elaborou-se um estudo no domínio de Sistemas Embarcados robóticos, modelado em UML, para a especificação de uma arquitetura de referência. Nesse estudo avaliou-se modelos (diagramas) por meio de métricas internas relacionadas ao atributo de qualidade de compreensibilidade estrutural, com base no modelo de qualidade de Ahrens et al. (2011). Um conjunto de estudos complementares foi realizado para avaliar o funcionamento do ambiente automatizado desenvolvido.

1.5 Organização da Dissertação

Neste capítulo foram apresentadas a contextualização, a motivação, a definição do problema, o objetivo e a metodologia para o desenvolvimento do trabalho. Os demais capítulos desta dissertação estão estruturados conforme descrito a seguir.

No Capítulo 2 é descrita a fundamentação teórica, que abrange conceitos de processo de desenvolvimento de software. Também são apresentados conceitos de qualidade de software para a garantia de qualidade em vários níveis. Nesse contexto, apresenta-se um conjunto de modelos de qualidade e métricas para avaliação de software amplamente utilizados e adaptados para domínios específicos.

No Capítulo 3, apresenta-se a definição de um *framework* conceitual, proposto para a avaliação de modelos de projetos de software. O *framework* visa orientar as definições e

[2]Deste ponto em diante, os termos *modelo* e *artefatos em nível de projeto* serão usados como sinônimos. Quando o termo *modelo* estiver associado a *modelo de qualidade*, a distinção ficará explícita no texto.

as execuções das atividades diante dos propósitos estabelecidos para a avaliação.

No Capítulo 4 são descritos os estudos realizados para a identificação e implementação dos mecanismos que automatizam os processos definidos pelo *framework* proposto. Esse ambiente foi desenvolvido contando com o apoio de várias tecnologias e bibliotecas de desenvolvimento, dentre as quais destacam-se as API *OpenCore* para a extração de métricas diretamente dos modelos e a plataforma de avaliação de qualidade *Spago4Q* para avaliação e síntese de resultados.

No Capítulo 5 são apresentados dois estudos: (i) estudo de viabilidade da aplicação do *framework* proposto e dos mecanismos automatizados; e (ii) estudo complementar para avaliação piloto do ambiente desenvolvido e das definições realizadas, nesses estudos analisaram-se as ferramentas *MMTool* e a utilização a plataforma *Spago4Q*.

Por fim, no Capítulo 6 apresentam-se as conclusões obtidas no trabalho, assim como suas contribuições, limitações e as propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Considerações Iniciais

Conforme definido no Capítulo 1, os objetivos deste trabalho envolvem os conceitos de processo de desenvolvimento de software, com ênfase na etapa de projeto de software, métricas internas e atributos externos de qualidade. Sendo assim, neste capítulo são apresentados os principais conceitos relacionados a esses três tópicos. A finalidade desse capítulo é familiarizar o leitor com estes conceitos.

Na Seção 2.2, apresenta-se uma visão geral sobre processos de desenvolvimento de software, destacando abordagens para projetos de software orientado a objetos (OO) e projetos de sistemas. Na Seção 2.3 apresentam-se os conceitos relacionados à qualidade de software e um conjunto de modelos de qualidade de produtos de software. Por fim, a Seção 2.4 traz conceitos relacionados às métricas de software, tais como métricas tradicionais, métricas para projetos OO e métricas para sistemas embarcados.

2.2 Processo de Desenvolvimento de Software

De acordo com Sommerville (2011), o processo de desenvolvimento de software é um conjunto completo de atividades necessárias para transformar os requisitos dos usuários em um produto de software. Para Fuggetta (2000), os processos de desenvolvimento de software são definidos como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para compreender, desenvolver e manter um produto de software.

Dentre as inúmeras atividades relacionadas com a Engenharia de Software, cita-se a criação de técnicas, elaboração de processos, Verificação e Validação (V&V), confiabilidade dos modelos de projeto e métodos de estimativas de índice de qualidade (PRESSMAN, 2010).

Pressman (2010) destaca que um processo de desenvolvimento é um arcabouço estrutural para a realização de tarefas necessárias na construção de um software de alta qualidade. Trata-se de um processo que define a estratégia para controlar e gerenciar os projetos de software e seus artefatos, tais como diagramas, documentos, dados, formulários e programas produzidos.

O conceito de modelo de ciclo de vida tem por finalidade definir as principais fases da vida do produto de software. Tipicamente, o ciclo de vida abrange as seguintes atividades (FUGGETTA, 2000; LARMAN, 2004; KRUCHTEN, 2004): (i) especificação e análise dos requisitos, (ii) projeto, (iii) implementação, (iv) V&V, (v) implantação e manutenção.

Ressalta-se que existem várias propostas para a definição de um processo de desenvolvimento de software. Essas propostas abrangem os modelos tradicionais, tais como, o modelo de Royce (1970), o modelo baseado em prototipação proposto por Gomaa e Scott (1981) e o modelo espiral apresentado por Boehm (1989).

2.2.1 Projeto de Software – *Design*

Os projetos de software são o núcleo técnico do processo de desenvolvimento de software, pois a documentação e os artefatos elaborados nessa fase são aplicados independentemente do modelo de ciclo de vida e do paradigma de programação adotado no desenvolvimento. Em processos tradicionais essa fase se inicia assim que os requisitos do software são modelados e especificados, sendo que a fase de projeto é a primeira dentre as três atividades técnicas (projeto, implementação e testes) requeridas para se desenvolver um sistema de software (PRESSMAN, 2010).

A fase de projeto tem por objetivo elaborar a modelagem de como o sistema deve ser implementado, com a adição de requisitos não funcionais aos modelos elaborados na fase de análise (WAZLAWICK, 2010).

Durante a fase de projeto, utilizam-se linguagens de modelagem para descrever os componentes do software e os relacionamentos entre esses componentes, além de seus respectivos comportamentos em termos do fluxo de execução e de dados. Considerando que a avaliação de modelos (diagramas) é o objetivo desse trabalho, a seguir são apresentadas

abordagens para projeto e modelagem de software OO e de sistemas.

2.2.2 Projeto de Software Orientado a Objetos

A modelagem de projetos Orientados a Objetos (OO) é uma das maneiras de mapear problemas, por meio da utilização de modelos fundamentados em conceitos do mundo real. Durante a análise do problema, a modelagem é realizada identificando-se e analisando-se os objetos e os eventos que interagem com esses objetos para solucionar o problema em questão.

Um projeto OO é realizado reutilizando-se as classes de objetos existentes e, se necessário, definindo-se novas classes. Ao modelar um sistema, deve-se identificar os tipos de objetos e as operações que representam o seu comportamento.

Dentre as diversas abordagens para a elaboração de projetos de software OO, cita-se: a técnica *Object-Oriented Software Engineering* (OOSE) proposta por Jacobson et al. (1992); o Processo Unificado (JACOBSON et al., 1999); o *Open Process Framework* (OPF) proposto por Firesmith e Henderson-Sellers (2002); e a proposta de Larman (2004), a qual propõe práticas para projetos OO, modelados com UML e baseadas no Processo Unificado, com a aplicação de padrões de projeto. Em seguida, apresenta-se uma visão geral do Processo Unificado.

O Processo Unificado (PU)

O Processo Unificado (PU) de Jacobson et al. (1999) originou-se como um processo de desenvolvimento iterativo e incremental, visando à construção de software OO. Devido à evolução dos processos de Engenharia de Software, os conceitos do PU foram revisados pela empresa Rational, e o PU passou a ser o Processo Unificado Rational (RUP, do inglês, *Rational Unified Process*).

De acordo com Larman (2004), nesse processo o desenvolvimento é organizado em subprojetos com durações fixas, as chamadas *iterações*. Ao final de cada iteração tem-se um produto testado, integrado e executável. Durante essas iterações, realizam-se as atividades de análise, projeto, implementação e testes.

Na Figura 2.1, apresenta-se uma visão geral do RUP. Observa-se as atividades (disciplinas) ao longo das fases. São seis atividades de Engenharia do Software (modelagem de negócios, requisitos, análise e design, implementação, teste e implantação) e atividades de apoio (gerenciamento de configurações e mudanças, gerenciamento de projetos e

ambiente).

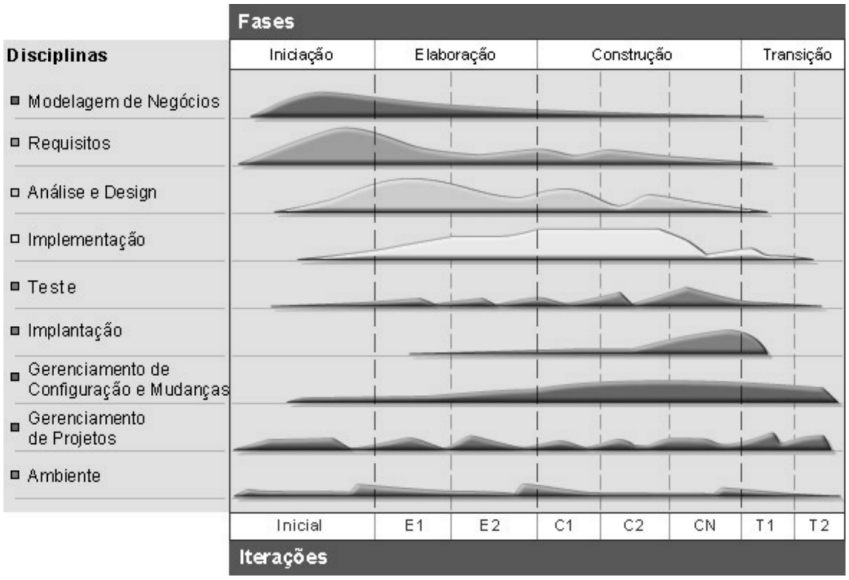


Figura 2.1: Processo Unificado – adaptado de Kruchten (2004).

Cada uma das atividades do RUP passa pelo ciclo de vida, que é composto por quatro fases. São elas:

- **Iniciação** – A fase de iniciação tem como propósito estabelecer uma visão inicial com os objetivos do projeto (LARMAN, 2004). Durante a iniciação criam-se as regras de negócio para a construção do software. A visão é o artefato essencial durante a fase de concepção, praticamente sendo uma descrição em alto nível do software (KRUCHTEN, 2004).

- **Elaboração** – A finalidade da fase de elaboração é estabelecer uma arquitetura do software, fornecendo uma base estável para as principais atividades de design e implementação da fase de construção (KRUCHTEN, 2004). De acordo com Larman (2004), a elaboração é composta de duas a quatro iterações, na qual cada iteração dura entre duas a seis semanas, com datas previamente definidas.

- **Construção** – Durante a fase de construção é finalizado o desenvolvimento do software. Nessa fase se realiza o gerenciamento de recursos, o controle das operações, a otimização dos custos, a programação e a garantia de qualidade (KRUCHTEN, 2004). As primeiras iterações dessa fase focam no refinamento dos artefatos do *design* e no inicio da implementação, já nas iterações finais o foco se volta para as atividades de testes e implantação.

- **Transição** – A fase de transição tem o proposito de garantir que o software esteja disponível para o usuário final. Nessa fase são executadas as atividades de testes finais para que ocorra a liberação dos produtos e com base no *feedback* do usuário, pequenos ajustes são realizados (KRUCHTEN, 2004).

Larman (2004) apresenta uma abordagem do Processo Unificado que utiliza a linguagem de modelagem UML para a elaboração dos artefatos do projeto de software. O autor argumenta que em um processo de desenvolvimento de software, as fases de análise (especificação do problema e domínio) e de projeto (elaboração da solução) abrangem a maior parte do tempo.

As soluções investem em etapas que antecipam a implementação, ou seja, visam em uma especificação capaz de representar o software por meio de diagramas e modelos. Esse é um exemplo de processo apoiado por modelagem em nível de projeto, que embasa a relevância de avaliar o nível de qualidade dos diagramas.

2.2.3 Projetos de Sistemas

Os métodos definidos para a engenharia de sistemas utilizam modelos, geralmente criados em uma linguagem padrão, para representar os artefatos projetados nas atividades de desenvolvimento do sistema. Essa abordagem é chamada de Engenharia de Sistemas Baseadas em Modelos (do inglês, *Model-Based Systems Engineering* – (MBSE)).

Algumas das principais abordagens e métodos padrão para engenharia de sistemas são: *Harmony* de DOUGLASS (2005); *Object-Oriented Systems Engineering Method (OOSEM)* proposto por LYKINS (2000); *Rational Unified Process for Systems Engineering (RUP SE)* apresentado por MURRAY (2003); *Vitech Model-Based Systems Engineering Method* definido por LONG (2000); e, o processo SYSMOD proposto por Weilkiens (2008).

A MBSE é a aplicação formalizada de modelagem para a especificação de requisitos, a análise, o projeto, a verificação e validação das atividades de concepção do sistema. O objetivo é auxiliar e facilitar a realização das atividades definidas para os processos de engenharia de sistemas (INCOSE, 2007).

Nesse contexto, o *Object Management Group* (OMG) apresenta um conjunto de técnicas para a MBSE de acordo com os princípios da Arquitetura Dirigida a Modelos (do inglês, *Model-Driven Architecture* – MDA) (Object Management Group, 2003). O MDA embasa o Desenvolvimento Dirigido por Modelos (do inglês, *Model-Driven Development* -

MDD), o qual se trata de um paradigma que utiliza modelos para o processo de desenvolvimento, ou seja, a implementação pode ser gerada automaticamente a partir dos modelos. Ao invés de implementar um software utilizando linguagens de programação, o MDD permite projetar as funcionalidades e recursos desejados utilizando um conjunto de modelos (ATKINSON; KUHNE, 2003; SCHMIDT, 2006).

Em linhas gerais, o MDD aplica o conceito de separar a especificação do sistema dos detalhes de sua implementação. Os detalhes de como o sistema irá utilizar as capacidades de uma plataforma especifica para operar, são definidos em uma etapa posterior do desenvolvimento (MARTíNEZ et al., 2009).

A utilização de modelos resulta em um aumento na qualidade de comunicação, especificação e precisão do projeto, integração e reúso dos artefatos desenvolvidos (WEILKIENS, 2008). Esses artefatos representam, por exemplo, uma modelagem coerente do sistema, em que o esforço é concentrado no refinamento dos modelos inicialmente desenvolvidos, utilizando-se ferramentas e métodos indicados (MARTíNEZ et al., 2009).

No contexto de sistemas embarcados, de acordo Liggesmeyer e Trapp (2009), nota-se que são várias as linhas de pesquisas que visam aplicar a MBSE e MDD. O domínio de sistemas embarcados é amplo e pode ser subdividido em vários subdomínios, o que limita a criação de uma única plataforma ou técnica de projetos de software para esses tipos de sistemas (LIGGESMEYER; TRAPP, 2009).

Em seguida, apresenta-se uma síntese do processo de desenvolvimento para sistemas denominado *System Modelling Process (SYSMOD)*. Tal processo tem como base a especificação completa de sistemas por meio de diagramas da linguagem de modelagem SysML/UML.

Processo SYSMOD

O SYSMOD (WEILKIENS, 2008) é um processo de desenvolvimento *top-down* proposto com base nos diagramas fundamentais da linguagem de modelagem SysML/UML. O processo inclui etapas e atividades bem definidas para a especificação de sistemas de maneira geral. Além disso, o SYSMOD apresenta aspectos de rastreabilidade por meio do uso de anotações em diversos modelos.

O SYSMOD inclui cinco fases, sendo elas: (i) Requisito; (ii) Contexto do Sistema; (iii) Caso de Uso; (iv) Conhecimento de Domínio; e, (v) Estrutura do Sistema. Em cada fase está inserida a abstração da análise e projeto do sistema. Em seguida descrevem-se, resumidamente, as fases do SYSMOD:

- **(i) Requisitos** – é a fase dedicada à coleta e à análise dos requisitos essenciais e técnicos. Os requisitos essenciais descrevem o sistema independentemente da sua solução técnica, possibilitando a reutilização dos requisitos em nível mais alto de abstração e a aplicação desses requisitos em diferentes tipos de soluções técnicas específicas. O diagrama de requisitos elaborado nessa fase se articula com as demais fases do processo.

- **(ii) Contexto do Sistema** – é a fase na qual se realiza uma sequência de refinamento dos diagramas *Block Definition Diagram* (BDD) e *Internal Block Diagram* (IBD). Esses diagramas são elaborados com base nos requisitos especificados. A cada refinamento adicionam-se elementos aos diagramas, tais como a definição dos limites (do inglês, *boundarys*) do sistema, o fluxo das informações com os respectivos atores do sistema e o ponto de interação entre os atores do sistema. A modelagem do sistema descreve de forma geral o fluxo global do sistema por meio de diagramas de atividades.

- **(iii) Casos de Uso** – nessa fase realiza-se a especificação do sistema por meio dos casos de uso, na qual são estabelecidos aspectos importantes entre os requisitos e os casos de uso essenciais do sistema. Para manter a consistência do sistema, deve-se realizar a remoção de redundâncias nos casos de uso, identificando os fluxos e os casos equivalentes durante a modelagem. Por exemplo, em um sistema de computador de bordo, os casos de uso "rota" e o "modificar rota" possuem interesses semelhantes. Isso deve ser gerenciado por meio de um glossário, identificando a similaridade e o respectivo processo de reorganização (do inglês, *refactoring*) dos casos de uso.

- **(iv) Conhecimento de Domínio** – é a fase na qual são especificados os principais blocos e seus respectivos relacionamentos, proporcionando uma visão geral da estrutura do sistema. Essa especificação é realizada via diagramas de definição de blocos (BDD), identificando as interfaces operacionais essenciais do sistema. Nessa etapa leva-se em consideração os aspectos de restrição temporal (do inglês, *clock*) e hardware para o sistema. Tais aspectos são estritamente relacionados com os requisitos não funcionais de um sistema e são representados pelos diagramas paramétricos.

- **(v) Estrutura do Sistema** – é a última fase do processo, na qual define-se o domínio da aplicação por meio de diagramas de sequência, os quais representam os aspectos e interesses comportamentais do sistema. Essa fase trata de questões como, por exemplo, o tipo do evento que irá acionar uma transição no sistema.

Esse comportamento é representado por uma máquina de estados, que fornece uma abstração do sistema em relação à visão comportamental.

Ressalta-se, que o processo SYSMOD não cobre aspectos particulares de projetos envolvendo empresas de domínios específicos. Desse modo, deve-se considerar os diferentes tipos de áreas e domínios para a elaboração dos artefatos.

A seguir, apresenta-se uma visão geral das linguagens de modelagem UML e SysML, que são empregadas para apoiar os processos aqui descritos. Salienta-se que esses são apenas dois exemplos de linguagem de modelagem, mas que o *framework* e as ferramentas propostas nesta dissertação não são limitadas a elas.

Linguagens de Modelagem UML e SysML

A *Unified Modeling Language (UML)* é uma linguagem visual genérica de modelagem que pode ser usada para especificar, visualizar, construir e documentar os componentes de um sistema de software (BOOCH et al., 2005). Essa linguagem pode ser associada a um amplo conjunto de modelos de processo da Engenharia de Software e pode cobrir todo o ciclo de desenvolvimento (GUEDES, 2011).

O conjunto de diagramas é classificado em estrutural e comportamental. Os diagramas estruturais, tais como, os diagramas de classes, pacotes e componentes, descrevem os elementos do sistema e os seus relacionamentos. Por outro lado, os diagramas comportamentais englobam, por exemplo, os diagramas de atividades, de sequência e de comunicação. Os elementos estruturais são a base para a visão dinâmica descrita pelos elementos comportamentais do sistema ao logo do tempo (LAVAGNO et al., 2010).

Fowler (2003) apresenta três visões pelas quais aplicam a UML: (i) Perspectiva Conceitual ou UML como rascunho – são diagramas elaborados informalmente e incompletos para explorar partes difíceis do problema; (ii) Perspectiva de Especificação ou UML como planta de software – são diagramas de projetos detalhados e construídos para realizar a engenharia reversa (visualizar e entender o código) e engenharia avante (geração de códigos); e (iii) Perspectiva de Implementação ou UML como linguagem de programação – é a especificação completa e executável do software em UML, sendo que o código executável é gerado automaticamente, porém dependendo do nível de diagramação abordada.

No contexto de sistemas, o *Object Management Group* (OMG) propôs uma derivação à UML 2.0, resultando-se na *Systems Modeling Language (SysML)*. A primeira especificação da linguagem SysML foi publicada em setembro de 2006, atualmente está na versão 1.4, lançada pelo OMG em dezembro de 2013 (Object Management Group, 2014).

Friedenthal et al. (2008) referem-se à SysML como uma linguagem de modelagem gráfica baseada em UML de propósito geral, que pode ser utilizada para especificar, analisar e projetar sistemas incluindo aspectos de hardware, software, pessoas e processos. A SysML facilita a integração entre sistemas e desenvolvimento de software. Com isso, os engenheiros podem descrever como o sistema interage com seu ambiente e como os módulos devem interagir para atingir o comportamento e desempenho desejado.

Para modelar um sistema os engenheiros e arquitetos devem compreender como integrar todos os subsistemas, para formar um sistema que satisfaça os requisitos solicitados. Um modelo SysML fornece uma visão integrada do sistema, permitindo que a equipe de projeto identifique pontos de possíveis problemas e evite-os o mais cedo possível (Object Management Group, 2014).

Parte da linguagem original UML é reutilizada para a especificação da SysML. Nessa especificação definem-se nove diagramas, entre diagramas novos e os diagramas modificados em relação a UML. Os diagramas da SysML suportam a especificação, a análise, o projeto, a verificação e validação de sistemas. Esses diagramas permitem que em uma única especificação seja construída uma visão integrada do sistema, abrangendo hardware, software, dados, pessoas, processos, infraestrutura e subsistemas eletromecânicos (FRIEDENTHAL et al., 2008).

O simples uso de uma linguagem de modelagem bem estruturada não garante que os artefatos criados pelos projetistas de software terão a qualidade desejada (BECKER; RAUBER, 2011). Sendo assim, atividades de Garantia de Qualidade devem ser realizadas também durante a fase de projeto (ROCHA et al., 2001). Nesse contexto, as próximas seções contemplam o conceito de qualidade de software e formas de realizar sua medição por meio de métricas e modelos de qualidade.

2.3 Qualidade de Software

A qualidade de software pode ser considerada como um conjunto de características (ou *atributos de qualidade*) que devem ser alcançadas em um determinado nível, para que o produto atenda às necessidades de seus usuários (GOYAL; JOSHI, 2014). Essa é a área da Engenharia de Software com a finalidade de garantir a qualidade, por meio de definições e padronizações de processos de desenvolvimento e estratégias para melhorar os produtos de software. Por intermédio desse conjunto de características e definições que a qualidade de um produto pode ser definida e avaliada (ROCHA et al., 2001).

Cada um dos atributos de qualidade pode ser detalhado em vários níveis de subatributos (PETRINJA et al., 2009). Com esse detalhamento pode-se obter um amplo conjunto de atributos, que se associam para estabelecer a qualidade de um produto de software (BUGLIONE et al., 2011). Rocha et al. (2001) salientam que embora os modelos aplicados na garantia da qualidade de software foquem principalmente no processo, o principal objetivo é assegurar a máxima qualidade possível ao produto final, atendendo às expectativas dos envolvidos de acordo com o planejado.

Milicic (2005) define que software de qualidade é aquele que cumpre seus objetivos, é gerenciável, com facilidade de manutenção, além de ter longa duração, de fácil operacionalização e aprendizagem. Conforme Fenton e Bieman (2014), um projeto de software com alta qualidade deve ter características que levam à qualidade dos produtos, tais como, a facilidade de compreensão, facilidade de implementação, facilidade para a aplicação dos testes e a facilidade de modificação. Bartié (1992) entende que qualidade de software é um processo sistemático, que abrange todas as etapas e artefatos produzidos no processo de desenvolvimento de software, com o objetivo de garantir a conformidade de processo e produto, prevenindo e eliminando defeitos.

De acordo com Jones (2008) o objetivo quando se desenvolve um produto de software, não é atingir a qualidade perfeita, mas sim a qualidade necessária e suficiente para o uso. Os estudos de Putnam e Myers (2003) complementam que não é possível obter qualidade máxima em todas as características de qualidade de um produto de software.

Controle e Garantia de Qualidade

Ao longo de um processo de desenvolvimento de software podem ocorrer enganos e interpretações equivocadas, resultando em defeitos nos produtos finais. Os principais provocadores desses problemas são a comunicação e transformação das informações, que acarretam no mau funcionamento do software. Sendo assim, é muito importante identificar esses defeitos o quanto antes no ciclo de desenvolvimento, preferencialmente na atividade em que ocorre o problema, para diminuir o trabalho de correção e custos com a manutenção. As atividades que se preocupam com essa questão são denominadas de atividades de garantia de qualidade de software, e devem ser realizadas durante todo o processo de desenvolvimento (MALDONADO; FABBRI, 2001).

De acordo com Maldonado e Fabbri (2001), as atividades de Verificação e Validação (V&V) são utilizadas para controlar e garantir a qualidade. As atividades que envolvem a verificação, tem por finalidade assegurar que o software esteja sendo desenvolvido de forma correta. A verificação deve inspecionar se os artefatos produzidos estão atendendo

aos requisitos estabelecidos e se os padrões organizacionais, de produto e processo, estão sendo aplicados adequadamente. A validação tem como objetivo garantir que o software esteja correto, ou seja, se o conjunto de requisitos atende ao uso específico planejado. Por fim, as atividades de testes são importantes atividades de validação, que consistem na análise dinâmica do software, isto é, a execução das partes que integram um produto de software. Devido à importância das atividade de V&V, elas devem ser planejadas, originando um Plano de Garantia de Qualidade.

Os artefatos de um projeto de software são as entradas para realizar as atividades de garantia da qualidade. Nas atividades de verificação, os modelos, documentação e artefatos são inspecionados quanto à conformidade em relação aos padrões de documentação estabelecidos pelas organizações. A inspeção dos artefatos é imprescindível para garantir a qualidade, tanto do produto final quanto das partes do projeto de software. Esses artefatos também devem ser validados em relação aos propósitos e aos requisitos que o projeto se propõe a realizar (LAVAZZA et al., 2012).

A despeito das atividades de garantia de qualidade, que podem ser realizadas durante o processo de desenvolvimento, torna-se necessário escolher um modelo que organize o conjunto de atributos de qualidade e proporcione a realização de avaliações de qualidade de software. Em seu estudo Pfleeger (1998) destaca que esses modelos auxiliam a compreender como as diferentes facetas contribuem para a qualidade do produto. Uma das principais estratégias para orientar a garantia de qualidade é por meio de modelos de qualidade.

Os modelos de qualidade se apresentam como referência para a qualidade de produtos de software. De maneira geral, esses modelos abrangem as características que um software deve apresentar, organizadas de forma hierárquica, relacionando atributos e métricas que permitem avaliar as características. Sendo assim, a seguir apresenta-se uma discussão sobre os modelos de qualidade considerados referência na área de Engenharia de Software.

2.3.1 Modelos de Qualidade de Software

Para realizar a avaliação de qualidade do software, deve-se estabelecer de maneira sistemática um conjunto de diretrizes para um determinado domínio de aplicação do software (GOYAL; JOSHI, 2014). Em outras palavras, um modelo de qualidade adequado deve ser definido para auxiliar no processo de avaliação de um produto de software específico.

Um modelo de qualidade é formado por vários atributos de qualidade, que são utili-

zados como um conjunto de verificação para determinar a qualidade do software (ABRAN, 2010). Além disso, para avaliar corretamente um software, precisa-se de um *framework* de avaliação, tipicamente composto por métricas e, se possível, do apoio de ferramentas de software para facilitar o processo de avaliação (BEUS-DUKIC; BØEGH, 2003). Ressalta-se que os modelos de qualidade de software são independentes do domínio de aplicação (BECKER; RAUBER, 2011).

A seguir, são apresentados alguns modelos de qualidade encontrados na literatura. Esses modelos influenciaram a definição de modelos específicos, auxiliando nas propostas de outros autores.

- **Modelo de Qualidade – McCall**

De acordo com Kitchenham e Pfleeger (1996), um dos modelos de qualidade de software mais reconhecidos é o modelo de qualidade apresentado por McCall et al. (1977). Possivelmente, é um dos primeiros modelos teóricos encontrados na literatura científica. Esse modelo teve origem nos meios militares americanos e foi utilizado para o processo de desenvolvimento de sistemas.

O modelo de qualidade de McCall tenta estabelecer uma estreita relação entre os usuários e desenvolvedores, focando nos fatores de qualidade de software, que refletem tanto as opiniões dos usuários quanto às prioridades dos desenvolvedores (COLOMBO; GUERRA, 2014).

Na Figura 2.2 apresentam-se as três grandes perspectivas para definir e identificar a qualidade de um produto de software, são elas: *Operação do Produto*, que são as características de operação do software; *Revisão do Produto*, que considera a capacidade de submeter-se a mudanças; e *Transição do Produto*, que se refere à adaptabilidade a novos ambientes.

Figura 2.2: Modelo de McCall – adaptado de McCall et al. (1977).

McCall et al. (1977) definem o modelo de qualidade de forma hierárquica em fatores [1], critérios e métricas. Ele possui 11 fatores de qualidade (para especificar), que descrevem a visão externa do software, conforme a visão dos usuários. Além disso, o modelo tem 23 critérios de qualidade (para construir), que descrevem a visão interna do software conforme visão dos desenvolvedores. Por fim, podem ser subdivididos em métricas (para controlar), que são definidas e usadas para prover uma escala e um método para a medição.

Os fatores de qualidade descrevem diferentes tipos de características comportamentais do sistema, e os critérios de qualidade são atributos para um ou mais fatores de qualidade. As métricas de qualidade, por sua vez, objetivam capturar alguns dos aspectos dos critérios de qualidade.

A abordagem de McCall é definir um conjunto de métricas e desenvolver expressões para cada fator de qualidade de acordo com:

$$F_q = c_1 m_1 + c_2 m_2 + ... + c_n m_n \tag{2.1}$$

Onde F_q é um fator de qualidade, c_n são coeficientes de regressão e m_n são as métricas que afetam esse fator qualidade. Observa-se que em muitas das métricas definidas para McCall, esses efeitos só podem ser definidos de forma subjetiva. As métricas podem ser utilizadas sob a forma de uma lista de verificação, usada para analisar os aspectos subjetivos do software. As métricas propostas por McCall podem ser classificadas numa escala de 0 (baixo) a 10 (alta).

- **Modelo de Qualidade – Boehm**

O modelo de qualidade proposto por Boehm et al. (1978) consiste em um conjunto de atributos e métricas para avaliar quantitativamente a qualidade de software. O modelo de Bohem é estruturado similarmente ao modelo de qualidade de McCall, pois apresenta-se de maneira hierárquica, com os atributos de nível alto, intermediários e primitivos. Cada um deles contribui para o nível total de qualidade (MILICIC, 2005).

De acordo com Milicic (2005), o modelo de Boehm foca seus atributos em três questões principais: (i) Como é a utilidade: "O quanto (facilmente, confiavelmente, eficientemente) posso usar o software como ele é?"; (ii) Manutenibilidade: "O quanto é fácil entender, modificar e retestar o software?"; e, (iii) Portabilidade: "Posso continuar usando o software

[1] Para simplificar a terminologia, neste trabalho, utiliza-se os termos "atributo de qualidade" e "subatributo de qualidade" com o mesmo sentido de "fatores" e "critérios" adotado por McCall et al. (1977).

se eu mudar o meu ambiente?".

Embora os modelos de McCall e de Bohem possam parecer muito similares, existem diferenças com relação à hierarquia dos atributos. Pode-se afirmar que foram esses modelos que inspiraram os atuais modelos de qualidade, apresentados na norma ISO/IEC 9126, e atualmente na ISO/IEC 25010 (MILICIC, 2005).

- **Modelo de Qualidade – ISO/IEC 9126**

A norma ISO/IEC 9126 (ISO/IEC, 2001) é um padrão internacional, desenvolvido pelo Subcomitê de Software (SC7) do Comitê Técnico Conjunto (JTC1) da ISO e IEC. A norma divide-se em quatro relatórios técnicos, no qual cada um assume responsabilidades sobre a avaliação do produto de software. São eles:

- Parte 1: ISO/IEC 9126-1 – Modelo de Qualidade;
- Parte 2: ISO/IEC 9126-2 – Métricas Externas;
- Parte 3: ISO/IEC 9126-3 – Métricas Internas;
- Parte 4: ISO/IEC 9126-4 – Métricas de Qualidade em Uso;

A primeira parte da norma estabelece um modelo de qualidade de produtos de software, composto de: (i) qualidade externa e interna; e (ii) qualidade de uso.

Os atributos de qualidade externa são identificados quando os projetos de software são executados, sendo a avaliação realizada quando os produtos são testados. Dessa forma, resulta-se em uma visão dinâmica do software. As propriedades internas de qualidade são avaliadas por meio da verificação dos artefatos do projeto, em geral documentação, modelagem e códigos fontes de software. Essa inspeção é uma visão analítica estática do software.

A qualidade em uso é a capacidade do produto de software de permitir que usuários atinjam as metas especificadas com eficácia, produtividade, segurança e satisfação em contextos de uso definidos. Em outras palavras, a qualidade em uso é a visão da qualidade sob a perspectiva do usuário.

Conforme ilustrado na Figura 2.3, a qualidade em uso é dependente da obtenção da qualidade externa, a qual é estritamente dependente da qualidade interna. Os atributos internos adequados são pré-requisitos para estabelecer comportamentos externos definidos

e, por outro lado, o comportamento externo correto é um pré-requisito para obter a qualidade em uso.

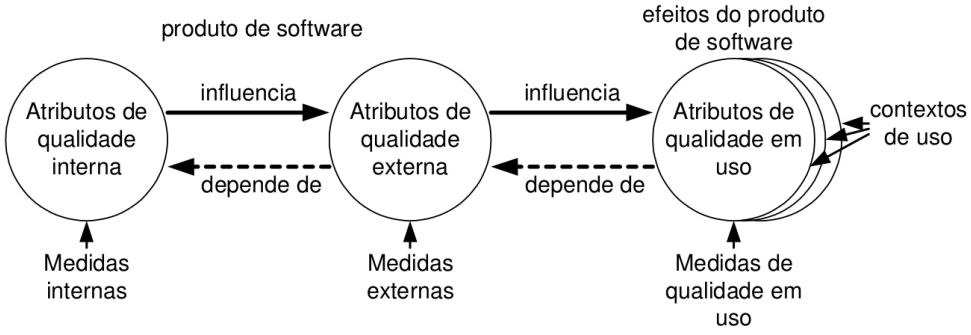


Figura 2.3: Níveis de Qualidade ISO/IEC 9126 – adaptado de ISO/IEC (2001).

A qualidade interna e externa do modelo de qualidade divide-se em seis atributos, sendo que um atributo pode ter vários níveis de subatributos. O nível mais alto dessa estrutura consiste em atributos de qualidade, e o nível mais baixo são subatributos mensuráveis. Na Figura 2.4 apresenta-se o modelo de qualidade para atributos externos e internos.

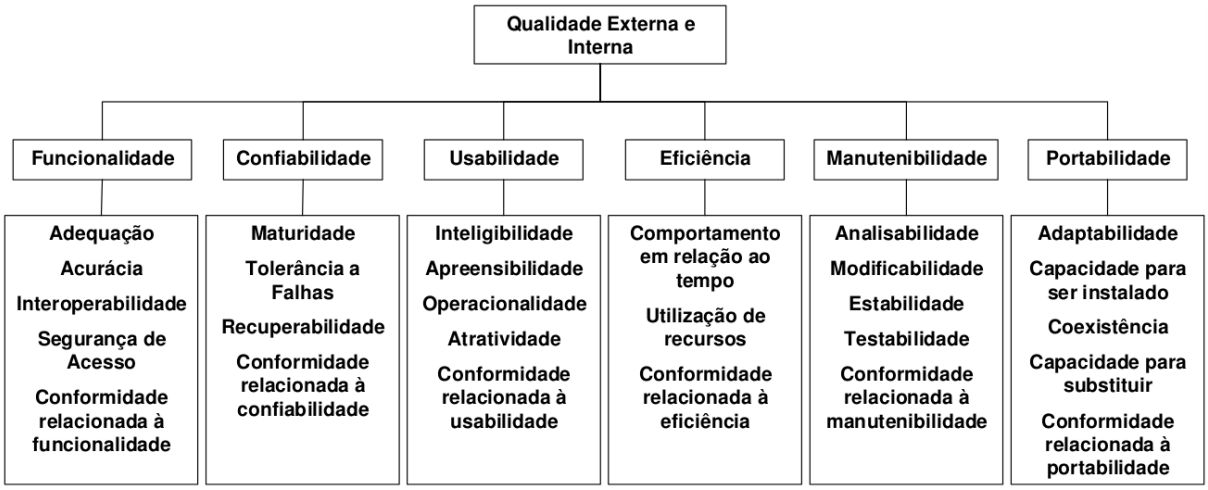


Figura 2.4: Modelo de Qualidade ISO/IEC 9126 – adaptado de ISO/IEC (2001).

Os atributos desse modelo de qualidade de produto de software são classificados numa estrutura hierárquica, atendendo características e subcaracterísticas. Sobre os atributos de qualidade de software são definidas métricas internas e externas, tais métricas são apresentadas nas partes 2 e 3 da respectiva norma em (ISO/IEC, 2001).

A aplicação da norma ISO/IEC 9126-1 permite que a avaliação da qualidade do produto de software seja especificada e avaliada pelos envolvidos com aquisição, requisitos, projeto, desenvolvimento, uso, avaliação, apoio, manutenção, garantia de qualidade e auditoria de software. Essa norma pode, por exemplo, ser aplicada por desenvolvedores,

adquirentes, equipe de garantia de qualidade e avaliadores independentes, ou seja, os responsáveis por especificar e avaliar qualidade do produto de software.

- **Modelo de Qualidade – ISO/IEC 25010**

A série de normas ISO/IEC 25000 – *Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE)*, na sua divisão ISO/IEC 25010, trata do modelo de qualidade para produtos de software. É uma reestruturação da norma ISO/IEC 9126. Ela fornece um modelo baseado em um conjunto de atributos de qualidade que um produto de software deve apresentar, o qual também é organizado hierarquicamente em subatributos (ISO/IEC, 2011).

Na Figura 2.5, apresentam-se os oito atributos (que são subdivididos em subatributos) do modelo de qualidade. Nota-se que os atributos de qualidade se relacionam com propriedades tanto estáticas quanto dinâmicas do software (ISO/IEC, 2011).

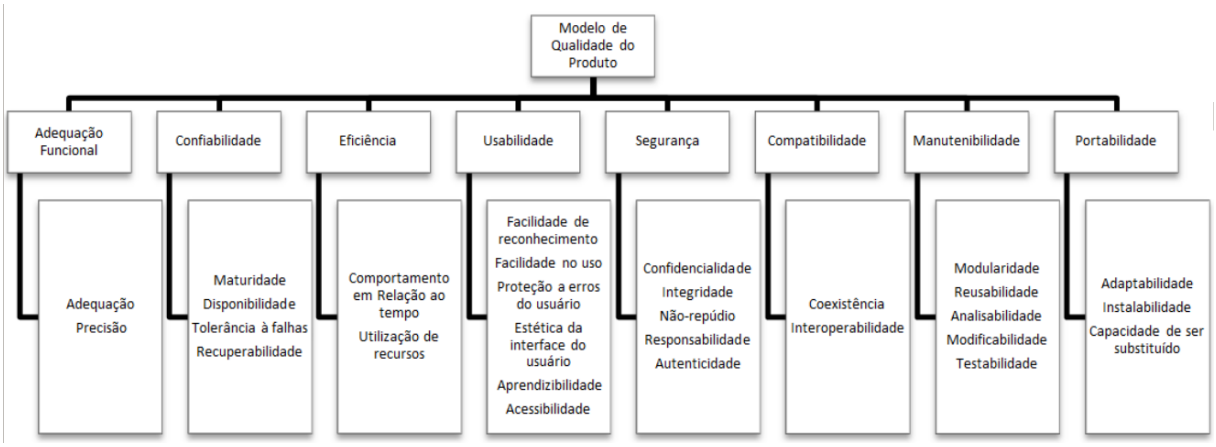


Figura 2.5: Modelo de Qualidade ISO/IEC 25010 – adaptado de (ISO/IEC, 2011).

Colombo e Guerra (2014) referenciam que esse modelo de qualidade pode ser aplicado para a definição de requisitos de qualidade de um produto de software, avaliação da especificação de software, para verificar se ele irá satisfazer os atributos de qualidade durante o desenvolvimento, descrição das particularidades e atributos implementados.

- **Modelos de Qualidade Específicos**

Os modelos de qualidade têm sido um amplo tema de pesquisa por várias décadas. Em consequência disso, há um grande número de modelos de qualidade propostos (KLÄS et al., 2009). Complementando, Wagner (2013) destaca que a partir dos anos 90, os pesquisadores vêm propondo formas elaboradas para decompor as características de qualidade

e, assim, construir modelos de qualidade mais ricos, com base em metamodelos explícitos. Esses modelos descrevem características de qualidade para a validação das estruturas.

Uma das primeiras tentativas de fazer uma ligação clara entre as medidas e os fatores de qualidade descritos em modelos de qualidade hierárquicos, foi elaborada no projeto ESPRIT REQUEST (KITCHENHAM; PICKARD, 1987), o qual denominou-se modelo de qualidade construtivo. Os autores destacam que o fator de qualidade é essencial em um modelo de qualidade, e argumentam que cada fator deve ser avaliado de maneira diferente, ao longo de diferentes fases de desenvolvimento. Portanto, avalia-se por diferentes métricas, cada fator em sua respectiva fase de desenvolvimento.

Os modelos de qualidade apresentados anteriormente são propostos para atender aos critérios de qualidade de âmbito geral. Esses modelos de qualidade tornam-se o ponto de partida para pesquisas que definem modelos de qualidade específicos.

Nota-se na literatura um amplo conjunto de modelos de qualidade para aplicação de avaliações específicas, tais como modelos de qualidade para projetos OO, modelos de qualidade para componentes de software e modelos de qualidade para Sistemas Embarcados. Todos são modelos de qualidade adaptados ou elaborados com base nos modelos apresentados. Essas adaptações são realizadas de acordo com as necessidades dos domínios e linguagem a serem aplicadas na avaliação (DELONE et al., 2003).

Modelo de Qualidade para Projetos OO

O Modelo de Qualidade QMOOD *(Quality Model for Object-Oriented Design)*, apresentado por Bansiya e Davis (2002), é um modelo com a proposta de estimar os atributos de qualidade que são de interesse para os usuários. Esse modelo tem formato hierárquico, o qual é definido com base no modelo de qualidade de ISO/IEC 9126-1.

A proposta é realizada em quatro etapas: (i) identificação dos atributos de qualidade, na qual são selecionados os atributos de qualidade presentes no modelo ISO/IEC 9126-1, (por exemplo, reusabilidade, flexibilidade, compreensibilidade, funcionalidade, extensibilidade e eficiência); (ii) identificação das propriedades dos projetos OO (por exemplo, abstração, encapsulamento, acoplamento e coesão); (iii) identificação das métricas de projeto OO (por exemplo, número total de classes no projeto, número de métodos abstratos e número de interface da classe); e (iv) identificação dos componentes (por exemplo, objetos, classes, relacionamentos e métodos).

A seguir apresenta-se um exemplo de uma fórmula do QMOOD para um dos atributos de qualidade externa, a reusabilidade:

$$Reusabilidade = -0,25 * acoplamento + 0,25 * coesao + 0,50 * mensagens + 0,50 * tamanhodoprojeto \quad (2.2)$$

Bansiya e Davis (2002) destacam que a identificação de um conjunto de atributos de qualidade, que representa completamente a avaliação da qualidade, não é uma atividade trivial e depende de vários fatores, incluindo os objetivos de gestão, os objetivos de negócio e tempo destinado para o desenvolvimento. Portanto, as métricas de projeto em nível mais baixo, definido em termos de características de *design* e qualidade, são avaliadas com a agregação dos atributos de qualidade de alto nível individual dos modelos.

Modelos de Qualidade para Componentes de Software

O modelo de qualidade para componentes proposto por Kumar et al. (2012) é um modelo elaborado com base no modelo da Norma ISO/IEC 9126, e nos modelos de Boehm et al. (1978), McCall et al. (1977) e Dromey (1995). O intuito desse modelo é realizar a avaliação de qualquer tipo de componente. Além disso, o modelo apresenta um conjunto de métricas para realizar a avaliação de cada subatributo.

O modelo de qualidade proposto por Rawashdeh e Matalkah (2008) é uma adaptação de vários modelos, tais como o modelo da Norma ISO/IEC 9126 e modelo de qualidade de Dromey (1995).

Modelos de Qualidade para Sistemas Embarcados

Pode-se encontrar propostas e adaptações de modelos de qualidade para a avaliação dos projetos de sistemas embarcados, tais como:

- O modelo de Wei et al. (2012), que estabelece um modelo de qualidade elaborado com base na ISO/IEC–9126, visando avaliar as qualidades internas de modelos de sistemas embarcados modelados em Matlab/Simulink. Esse modelo de qualidade contém seis atributos para a avaliação interna de modelos Simulink. Além disso, define um conjunto de métricas internas para cada atributo.

- Jeong e Kim (2011) adaptaram o modelo de qualidade para sistemas de informação de DeLone&McLean (DELONE et al., 2003) para o domínio de sistemas embarcados. Além disso, apresentaram um conjunto de atributos de qualidade para componentes de sistemas embarcados. Em outra pesquisa, Jeong (2013) realiza estudos adaptativos para a definição de um modelo de qualidade prático para sistemas e software

embarcados, identificando atributos de qualidade para SaaS (do inglês, *Software as a Service*) com base no modelo de qualidade ISO/IEC–9126 (ISO/IEC, 2001).

- Choi et al. (2008) definem um modelo de qualidade para a avaliação de componentes de Sistemas Embarcados. Nesse trabalho, os autores realizam um estudo analítico dos atributos de qualidade dos modelos *Samsung software Component Quality evaluation Model (SCQM)* com o modelo de qualidade da Norma ISO/IEC-9126. Dessa análise, foram extraídos oito atributos de qualidade e, por meio de uma matriz de impacto, definiu-se a correlação entre esses atributos.

- Carvalho et al. (2009) apresentam um modelo de qualidade para a avaliação e validação de componentes de software embarcados, com base no modelo ISO/IEC (2011). Nesse trabalho é elaborada uma metodologia sob diferentes aspectos de pesquisa, seleção e avaliação da qualidade de componentes, pois são considerados pontos chave na adoção da abordagem Engenharia de Software Baseada em Componentes (CBSE). Além disso, nesse modelo é estabelecido um conjunto de métricas para todos os atributos de qualidade selecionados e adaptados para o modelo.

Assim como os modelos de qualidade estão relacionados com qualidade de software, as métricas estão incluídas em modelos de qualidades para a medição e associação com os atributos de qualidade. Em seguida, apresentam-se conjuntos de métricas geralmente utilizados para a avaliação do software em diferentes domínios e fase de desenvolvimento.

2.4 Métricas de Software

Segundo Pressman (2010), as métricas auxiliam os engenheiros de software a ganhar profundidade na visão sobre os projetos e na construção. De acordo com Sommerville (2011), a medição de software concentra-se em obter valores numéricos para alguns atributos de qualidade. Então, sobre esses valores é possível tirar conclusões referentes à qualidade do software. Uma métrica é qualquer tipo de medição que se pode ter de um sistema.

De acordo com o seu conceito original, as métricas de software têm por finalidade a identificação e medição dos principais parâmetros que afetam o desenvolvimento do software (PUTNAM; MYERS, 2003). De acordo com Jones (2008), a necessidade das métricas teve maior atenção quando se constatou a chamada crise do software. Nesse período, detectou-se problemas relacionados à gerência ineficaz durante o desenvolvimento e à

falta de medidas, bem definidas, que fossem viáveis para a avaliação do software e do seu desenvolvimento.

Porém, estimativas precisas e eficazes, planejamento e controle são aspectos difíceis de se concretizar em um conjunto. Assim, com o estabelecimento de métricas de software, propõe-se melhorias no processo de gestão com a identificação, medição e controle dos parâmetros essenciais do software (MILLS, 1999). Fenton e Bieman (2014) destacam que as métricas surgiram com a finalidade de auxiliar na tomada de decisões, ajudando a identificar as possíveis melhorias nos artefatos do software.

Em seu estudo Kan (2014) afirma que as métricas podem ser divididas em três domínios diferentes: (i) métricas de produto, que descrevem as características dos produtos, tais como, tamanho e complexidade; (ii) métricas de processo, podem ser usadas para melhorar a manutenção e desenvolvimento do software, por exemplo, a métrica eficiência de remoção de defeitos; e, (iii) métricas de projeto, que descrevem as características e a execução do projeto. Além disso, métricas de qualidade de software são um subconjunto das métricas de software, que focam em aspectos de qualidade de processos, produto e projetos.

Conforme Jones (2008), as métricas para avaliação de projetos OO são classificadas em três categorias: métricas de análise, métricas de projeto e métricas de construção. As métricas de projeto e de construção são as mais indicadas na obtenção da análise de código fonte e artefatos, pois a maioria das informações para o cálculo dessas métricas pode ser extraído de análises automáticas(ABRAN, 2010). Além disso, Olague et al. (2007) listam um conjunto de métricas para projetos orientados a objetos, no qual muitas dessas métricas são possíveis de serem aplicadas na fase de projeto.

Existem várias maneiras de se classificar as principais questões que impactam na qualidade do software. A classificação adotada neste trabalho inclui as categorias *atributos externos de qualidade* e *métricas internas*. As métricas internas relacionam-se com propriedades que podem ser diretamente mensuradas como, por exemplo, número de linha de código, número de classes em um projeto e complexidade entre as entidades. Os atributos externos como, por exemplo, a manutenibilidade, extensibilidade e reusabilidade, tipicamente são medidos indiretamente por meio da combinação de métricas internas (JONES, 2008).

Abuasad e Alsmadi (2012) destacam a dificuldade de se medir diretamente os atributos externos de qualidade de software. Atributos como a facilidade de manutenção, a complexidade e a facilidade de compreensão são afetados por um conjunto de diversos

fatores e não existem métricas diretas e simples para avaliá-los Sommerville (2011). Para isso, deve ser usado um conjunto de métricas internas, permitindo que sejam inferidos os atributos de qualidade externos, identificando-se uma relação entre as informações necessárias com ao que pode ser avaliado (ALSHAMMARI et al., 2010). Conforme Goyal e Joshi (2014), apesar de apenas os atributos externos terem importância no final, a chave para assegurar que eles serão satisfatórios são as métricas internas, as quais representam um meio para atingir a qualidade de software externa.

As métricas são classificadas em dinâmicas e estáticas. As métricas dinâmicas são extraídas por medições realizadas em um programa em execução e auxiliam na avaliação da eficiência e na confiabilidade. Por outro lado, as métricas estáticas são coletadas por meio dos artefatos do sistema como, por exemplo, projeto, documentação e diagramas. Essas métricas estáticas ajudam a avaliar a complexidade, a facilidade de compreensão e a facilidade de manutenção (KUMAR et al., 2012).

- **Métricas Tradicionais de Software**

Existe um amplo número de métricas de software que foram testadas, aplicadas e aceitas, tais como número de linhas de código, métricas de Halstead e complexidade ciclomática, mas essas não são medidas universalmente convencionadas. Estudos relatam experiências que tentam correlacionar as métricas com um número de propriedades de software (isto é, atributos de qualidade), incluindo-se complexidade e confiabilidade, propensão a erros e manutenibilidade (BRIAND et al., 2000).

Ao aplicar métodos baseados em métricas de software em um ambiente limitado ou domínio definido, é possível auxiliar na melhoria da qualidade do software e da produtividade (BASILI; ROMBACH, 1987). Em alguns casos, métricas relativamente simples, como linhas de código e complexidade ciclomática, são estratégias relevantes para predição de outras características, como número de erros, esforço total de desenvolvimento e manutenibilidade (JONES, 2008). Briand e Wüst (2002) ressalta que essas métricas podem obter resultados úteis se forem aplicadas em ambientes e contextos específicos.

Conforme Abdellatief et al. (2013), uma série de trabalhos são os precursores das abordagens com métricas para tamanho e complexidade de software. Nota-se que existe uma ampla quantidade de métricas para esse tipo de medição, conforme descrito no trabalho de Jabangwe et al. (2015). Esses autores identificaram e compararam 31 métricas de tamanho e complexidade. A seguir apresentam-se alguns exemplos de métricas tradicionais para mensurar o tamanho e complexidade do software.

Métricas de Tamanho

As métricas de tamanho são definidas com a finalidade de quantificar o tamanho do software e auxiliar nas medições na fase de concepção, partindo do princípio de desenvolvimento tradicional. Possivelmente, a métrica LOC *(Line of code)* é a mais usada para medir o tamanho do software e baseia-se no princípio de contar todas as linhas que não sejam linhas em branco ou comentários, isso independentemente do número de declarações por linha. Essa métrica foi evoluída para KLOC *(Kilo Lines of Code)* (JONES, 2008).

De acordo com Jones (2008), os resultados dessa métrica somente podem ser comparados quando se referirem à mesma linguagem e se a programação for normalizada. LOC é uma métrica para predição da complexidade do software, esforço total de desenvolvimento e desempenho do software, conforme descrito no estudo comparativo de Woodfield et al. (1981). Esse estudo aborda métricas de LOC, complexidade ciclomática e métricas Halstead, para indicar o esforço de desenvolvimento e desempenho do programador.

Métricas de Halstead (1977)

As métricas de complexidade de Halstead (1977) são baseadas na teoria da informação. Essas métricas são as primeiras com fundamentação teórica e são chamadas de Ciência do Software. As métricas foram elaboradas com o intuito de quantificar a complexidade do software a partir dos operadores e operandos em um módulo do sistema, essa medição é feita no código fonte. Além disso, as métricas de Halstead podem ser úteis durante o desenvolvimento para avaliar a qualidade do código e acompanhar as tendências de complexidade.

Métricas de McCabe (1976)

Para McCabe (1976), a complexidade de um software depende do número de condições (Caminhos), correspondendo ao número máximo de percursos linearmente independentes. Essa métrica pode ser representada por meio de grafos de fluxo de controle, no qual os nós indicam uma ou mais instruções sequenciais, e os arcos orientados representam o sentido do fluxo de controle.

A métrica proposta tem o intuito de medir a complexidade de um software e orientar durante o desenvolvimento e na execução dos testes. A complexidade ciclomática também pode ser utilizada para verificar o esforço de depuração e manutenção (BRIAND et al., 2000). Essa métrica foi estendida e alterada, pois não realizava a cobertura dos casos especiais com apenas uma única condição. Além disso, o próprio autor estende esse conjunto de métricas para *Complexidade da unidade real, Complexidade essencial,*

Complexidade de projeto de módulos, Complexidade total do projeto e Complexidade de integração (MCCABE; BUTLER, 1989).

- **Métricas de Software Orientado a Objetos**

De acordo com Sharma et al. (2012), existe uma ampla gama de estudos sobre métricas voltados para os aspectos relevantes no contexto do paradigma OO. As métricas apresentadas a seguir abrangem métricas de classes, métricas de métodos, métricas de herança, métricas de acompanhamento e métricas de sistemas (características gerais do software OO).

Métricas CK de Chidamber e Kemerer (1994): As métricas CK têm por finalidade medir a complexidade de um projeto em relação a atributos externos de qualidade, tais como a manutenibilidade, a testabilidade, a compreensibilidade e a reusabilidade.

Métricas de Li e Henry (1993): As métricas definidas por Li e Henry (1993) têm o objetivo de medir propriedades internas, tais como, acoplamento, complexidade e tamanho.

Métricas MOOD de Abreu e Carapuca (1994): Abreu e Carapuca (1994) apresentam o conjunto de métricas MOOD *(Metrics for Object Oriented Design)*. Essas métricas são definidas para realizar a avaliação em termos de produtividade no desenvolvimento e qualidade de software. O conjunto inclui métricas relacionadas a herança, encapsulamento, acoplamento e polimorfismo.

Métricas Lorenz e Kidd (1994): Lorenz e Kidd (1994) definem um conjunto de métricas com o objetivo de avaliar as informações estáticas de um projeto de software como a herança, tamanho, responsabilidades atribuídas às classes e aspectos internos da classe.

Métricas de Marchesi (1998): Com o objetivo de medir a complexidade do software e equilibrar as responsabilidades entre pacotes e classes, Marchesi (1998) elaborou um conjunto de métricas para classes, pacotes e métricas de complexidade global. Além disso, define que as responsabilidades são informações contidas e processadas pela classe, e que todos os relacionamentos são dependência de informação.

Métricas de Bansiya e Davis (2002): Além do modelo de qualidade QMOOD, Bansiya e Davis (2002) definem métricas para avaliar as propriedades de projetos, tais como encapsulamento, acoplamento, coesão, composição, polimorfismo e herança.

Métricas de Briand e Wüst (2002): Briand et al. (1997) apresentam um conjunto de métricas para medir o nível de acoplamento das classes durante o projeto de um software OO, tendo como base as métricas CK.

Métricas de Genero et al. (2004): O conjunto de métricas definido por Genero et al. (2004) foca na avaliação de atributos de qualidade externos, tais como a manutenibilidade, por meio dos relacionamentos de associação, generalização, agregação e dependência.

Métricas de Dallal e Briand (2010): Definem métrica de coesão em classes para design em alto nível(do inglês, *Higt-level design (HLD)*), baseiam-se em pressupostos realistas em conformidade com propriedades matemáticas esperados. A métrica considera interações entre método-método, atributo, atributo-método e suas relações transitivas, Além disso, pode ser usado para avaliar automaticamente a qualidade do projeto em fases iniciais utilizando diagramas UML.

Métricas de Aggarwal et al. (2007): Definem duas métricas *Number of Catch Blocks per Class (NCBC) and Exception Handling Factor (EHF)* para medir a robustez de software orientado a objetos. Essas métricas utilizam fórmulas matemáticas simples, seguem a abordagem semelhante do MOOD de Abreu e Carapuca (1994) e validadas em estudos empíricos.

Métricas de Badri et al. (2009): Apresentaram a métrica *Quality Indicator (QI)* que pode substituir várias métricas e coletar a capacidade de teste e manutenção de projetos orientados a objetos. A métrica é proposta com base nas fluxo de controle e cálculos de probabilidades das interaç oes entre as classes. Essa métrica foi validada empiricamente em sete projetos java e os resultado foram comparados com as métricas orientada a objetos conhecidas.

- **Métricas de Sistemas Embarcados**

No contexto de Sistemas Embarcados, pode-se encontrar na literatura algumas iniciativas de definição de métricas para modelos, assim como adaptações de métricas tradicionais e métricas de outros contextos para os domínios de Sistemas Embarcados. A seguir, apresentam-se alguns estudos que envolvem a definição de métricas para modelos de sistemas embarcados, independentemente da linguagem de modelagem utilizada.

Segundo Friedenthal et al. (2008), as métricas desempenham um importante papel para avaliar a qualidade de sistemas baseados em desenvolvido sobre a abordagem MBSE. Entre as aplicações práticas de medição de projeto no contexto de MBSE, incluem-se:

- Avaliação da integridade, da exatidão, a da consistência do modelo;
- Utilização de convenções para a modelagem;

- Estimativa de *design* e esforço de desenvolvimento; e,
- Acompanhamento da evolução do projeto e progresso do desenvolvimento.

No domínio de projetos de Sistemas Embarcados modelados em *Simulink*, cita-se o estudo elaborado por Antonio et al. (2014). Nesse estudo, define-se um conjunto de métricas internas aplicadas a modelos para analisar o atributo de qualidade externo de compreensibilidade. Essas métricas exploram os atributos relacionados com os elementos de modelos Simulink, tais como blocos funcionais, transições, *Fan-in, Fan-out* e parâmetros de configuração do bloco. Os autores definiram as métricas *NBM (Number of Blocks), NTM (Number of Transitions), Fan-in, Fan-out* de cada bloco e a métrica *MC (Model Complexity)*, que foi elaborada com base na pesquisa original de Henry e Kafura (1981).

Wei et al. (2012) realizam a especificação de métricas para análise de modelos *Simulink/Stateflow*, incluindo-se um modelo de qualidade definido com base na norma ISO/IEC 9126. São definidas métricas para os seguintes atributos de qualidade: analisabilidade, mutabilidade, estabilidade, testabilidade, compreensibilidade e adaptabilidade. Em seguida, Dajsuren et al. (2013) complementaram com métricas para modularidade, argumentando que esse é um dos principais atributos para a característica de manutenibilidade.

No trabalho de Ahrens et al. (2011), define-se um conjunto de métricas objetivas para avaliação de modelos que representam arquiteturas de sistemas em uma abordagem de projeto *top-down* ou *bottom-up*. Além disso, as métricas são associadas com os atributos de qualidade externos. Esse conjunto de métricas será apresentado em detalhes no decorrer do Capítulo 5.

Wüst (2013) apresenta um conjunto básico de métricas para SysML. Nesse conjunto de métricas estão inclusas métricas para contagem de número de blocos e requisitos em um determinado pacote, bloco ou requisito; número de portas e conectores no bloco; nível hierárquico do bloco; e, algumas adaptações das métricas aplicadas em UML. Sobretudo, Wüst (2013) ressalta que esse conjunto de métricas para SysML é um ponto de partida para a definição de métricas e regras especificas para a SysML, pois esse conjunto não apresenta uma base de estudos científico de validação para processos MBSE.

2.5 Computação de Métricas

As ferramentas para a computação de métricas de software permitem avaliar os artefatos de um projeto com base em métricas extraídas dos elementos que compõem os

modelos. Além disso, proporcionam a análise dos valores correspondentes. Ou seja, as ferramentas de métricas são programas que implementam um conjunto de métricas para projetos de software.

De acordo com o estudo comparativo entre as ferramentas de métricas realizado por Lincke et al. (2008), destaca-se que existem diferentes formas para a definição, implementação e interpretação das métricas. No estudo experimental realizado pelos autores, as métricas e as medidas obtidas para avaliar o *design* de projetos se diferem consideravelmente entre as ferramentas analisadas.

Os resultados da interpretação das métricas de software estão fortemente ligados com a inserção de modelos de qualidade. Dessa maneira, quando as métricas não são confiáveis e geram valores diferentes ou inválidos, os modelos de qualidade ficam expostos a resultados e análises imprecisas. Outro ponto relevante no resultado final da avaliação é a forma de definição das métricas, que normalmente estão fixas na implementação da ferramenta. Contudo, isso gera dependência entre os resultados e as ferramentas, influenciando na forma de análise dos resultados das métricas e, como consequência, isso se transmite para a fase de atribuição de nível de qualidade dos modelos do projeto.

Para os estudos experimentais de comparação entre ferramentas de métricas realizados por Lincke et al. (2008), foram calculados os valores de métricas utilizando um conjunto de métricas padrão e três sistemas de tamanhos diferentes de software. Além disso, definiu-se um modelo de qualidade de software simples para o tratamento do atributo de qualidade manutenibilidade.

Ao analisar os valores obtidos para cada uma das ferramentas de métricas (comerciais e livres) selecionadas para o estudo, pôde-se notar a dependência entre os resultados com as ferramentas. Os resultados obtidos para o mesmo conjunto de classe e métricas são diferentes entre as ferramentas analisadas (LINCKE et al., 2008).

No contexto deste trabalho, realizou-se estudos sobre ferramentas de métricas com o intuito de identificar características das mesmas, tais como os tipos de ferramentas, as estratégias, arquitetura e as principais métricas implementadas. Identificou-se também os principais requisitos para o processo de avaliação automática do nível de qualidade de modelos por meio de métricas. Os detalhes são apresentados no Capítulo 4.

2.6 Considerações Finais

O processo de desenvolvimento de software envolve um conjunto de atividades necessárias para transformar os requisitos dos usuários em um produto de software. Durante esse processo, há atividades que visam especificamente a aferir a qualidade dos artefatos desenvolvidos e, se necessário, indicar pontos de melhoria. A qualidade pode ser refletida por meio de uma série de atributos de qualidade, que geralmente são agregados em um modelo de qualidade. Embora, em geral, esses atributos não possam ser diretamente mensurados, propriedades internas, definidas na forma de métricas, são combinadas e associadas aos atributos de qualidade de interesse.

Neste capítulo, apresentou-se uma conceituação sobre processo de desenvolvimento de software, modelos de qualidade e métricas internas, com especial atenção à etapa de projeto (do inglês, *design*) de software. Os próximos capítulos desta dissertação descrevem o trabalho de mestrado realizado e os resultados alcançados, os quais relacionam-se diretamente com os três temas explorados neste capítulo.

Capítulo 3

Framework *Quality of Models based on Metrics* – QM^2

3.1 Considerações Iniciais

Neste capítulo apresenta-se a definição do *framework* conceitual, o qual tem como objetivo estabelecer os procedimentos necessários para a atribuição de níveis de qualidade a projetos de software, por meio da avaliação dos modelos que compõem o projeto. O *framework*, denominado QM^2 (***Q**uality of **M**odels based on **M**etrics*), é definido em processos. Os processos são representados pela notação BPMN e estão divididos em Processo de Definição, Processo de Extração, Processo de Avaliação e Processo de Síntese. Esses processos associam-se com o intuito de medir e atribuir níveis de qualidade a modelos de projetos de software, de acordo com os atributos do modelo de qualidade adotado.

O capítulo está estruturado da seguinte forma: na Seção 3.2 são apresentados os principais conceitos da linguagem BPMN utilizados para representar o QM^2. Em seguida, na Seção 3.3 traz-se uma visão geral do *framework*, destacando as etapas que compõem o processo conceitual proposto. Por fim, na Seção 3.4 são descritos os detalhes do funcionamento dos processos do QM^2.

3.2 A linguagem BPMN

Os processos do QM^2 são representados utilizando a linguagem de modelagem de processos BPMN – *Business Process Model and Notation* – disponibilizada pelo OMG. A linguagem BPMN é um padrão para modelagem de negócios, que tem por finalidade

possibilitar a compreensão dos seus procedimentos internos e estabelecer a comunicação padronizada entre esses procedimentos (Object Management Group, 2011).

A BPMN fornece uma notação gráfica por meio do diagrama BPD – *Business Process Diagram*. Esse diagrama é baseado no fluxo de trabalho semelhante ao diagrama de atividades da UML, facilitando o entendimento das colaborações de desempenho e as transações entre as entidades. Além de possibilitar a adaptação das organizações às novas regras de negócio (WHITE, 2005).

A linguagem apresenta um amplo conjunto de elementos para a representação de processos. Serão descritos apenas os principais elementos utilizados na modelagem dos processos do *framework*, pois a BPMN não é o foco deste trabalho. A especificação completa da BPMN encontra-se em um documento disponibilizado pelo OMG (Object Management Group, 2011).

Um processo de negócio na BPMN é a associação de atividades ou conjuntos de atividades (tarefas ou subprocessos) para a descrição de um conjunto de trabalho, ou seja, um processo é uma sequência lógica de passos a serem seguidos e que são representados no diagrama BPD. Esses elementos estão divididos em quatro categorias: objetos de fluxo; objetos de conexão; partição (*swinlanes*); e artefatos. Os objetos de fluxo são divididos em três tipos: Atividades, Desvios (*Gateways*) e Eventos (Object Management Group, 2011).

Uma *atividade* representa a execução de um trabalho realizado pela organização. A atividade pode ser composta ou atômica, e pode ser executada uma ou mais vezes por meio de iterações definidas. Uma atividade composta é denominada *subprocesso* e contém uma série de outras atividades. O subprocesso pode ser representado no diagrama como uma atividade única e semelhante a uma tarefa, com um sinal "⊞", o qual representa a abstração das atividades contidas no subprocesso. Uma atividade atômica é denominada de *tarefa*. As tarefas representam uma ação realizada em um processo e não podem ser decompostas em outras atividades.

A primeira versão da notação BPMN foi lançada em 2004[1], e desde então tem-se apresentado como uma opção de ferramenta para a modelagem de processos (WHITE, 2005). Devido ao alto nível de detalhes e ampla utilização da notação BPMN, os processos do QM^2 propostos neste trabalho consideram essa notação para a representação.

Nas seções que descrevem cada uma das partes do QM^2, usa-se o termo *processo* para se referir aos *subprocessos* e *atividades* macro representados nos diagramas. Quando uma

[1] `http://www.bpmn.org/` – Acessado em Setembro de 2014.

atividade tratar de uma *tarefa*, o termo *tarefa* será empregado.

3.3 Visão Geral do QM²

Nesta seção, apresenta-se uma visão geral do QM² e seus respectivos processos para a atribuição do nível de qualidade a projetos de software. O *framework* tem por finalidade avaliar o nível de qualidade dos modelos em nível de projeto. Um exemplo típico de projeto é o que consiste em um conjunto de diagramas UML como, por exemplo, diagramas de classes, comunicação e sequência.

O QM² tem o objetivo de auxiliar os engenheiros de software no gerenciamento do processo de avaliação, coleta de medidas, atribuição do nível de qualidade e interpretação dos dados referentes ao projeto do software.

Além disso, o *framework* pretende garantir a precisão dos resultados, a reprodução das avaliações e facilitar a verificação em futuras replicações. De acordo com Comella-Dorda et al. (2002) a obtenção de resultados consistentes pode ser somente atingida por meio de um processo de avaliação bem definido e com alto nível de qualidade.

O QM² possibilita avaliar o nível de qualidade dos modelos durante a fase de projeto, destacando os pontos positivos e negativos da modelagem. Por meio dessa avaliação pode-se, por exemplo, identificar a capacidade de reutilização dos componentes contidos na modelagem. Essa identificação acontece quando o nível de qualidade dos componentes é considerado alto ou satisfatório, então eles podem ser selecionados para a reutilização. Além disso, o QM² prevê a definição de modelos de qualidade, definição de métricas internas de software e a realização do mapeamento entre as métricas internas e atributos de qualidade.

Na Figura 3.1, apresenta-se uma visão geral dos processos para avaliação do nível de qualidade dos modelos proposto pelo QM². A avaliação é composta de quatro processos: (i) *Processo de Definição*; (ii) *Processo de Extração*; (iii) *Processo de Avaliação*; e, (iv) *Processo de Síntese*. Os processos apresentados na Figura 3.1 são descritos a seguir.

Processo de Definição

O *Processo de Definição* (representado pelo processo *Definir* na Figura 3.1) consiste na elaboração do planejamento da avaliação e inclui várias atividades, tais como: definição do modelo de qualidade, análise da linguagem de domínio, definição das métricas internas e o mapeamento entre as métricas internas e os atributos externos de qualidade.

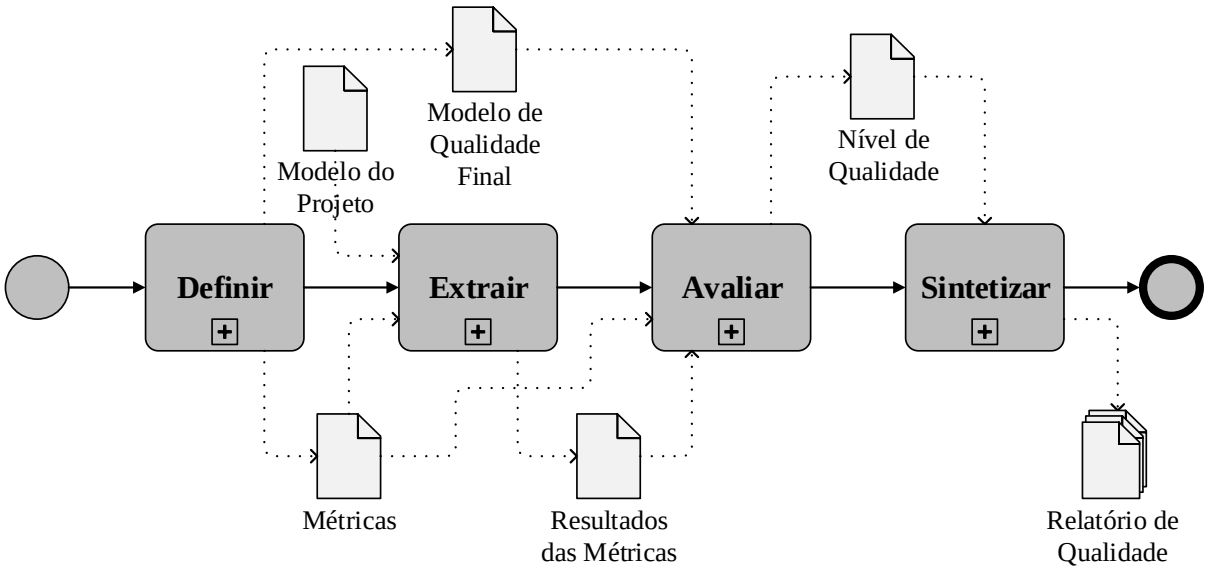


Figura 3.1: Visão Geral do QM².

O modelo de qualidade estabelece os relacionamentos entre os atributos externos com as métricas internas. A tarefa de mapeamento é realizada por intermédio do conjunto de métricas previamente definido e pela análise da linguagem de domínio, que é realizada para compreender a estrutura dos elementos dos modelos do projeto. Os detalhes do *Processo de Definição* são apresentados na Seção 3.4.1.

Processo de Extração

O *Processo de Extração* de dados (representado pelo processo *Extrair* na Figura 3.1) integra as atividades de coletar as medidas diretamente dos modelos, utilizando as métricas previamente definidas e armazenar os resultados obtidos. As entradas para o processo de extração são: o modelo do projeto de software (conjunto de diagramas), que é representado por uma linguagem de modelagem como a UML, SysML ou Simulink; o modelo de qualidade e as métricas. Os detalhes do *Processo de Extração* são apresentados na Seção 3.4.2.

Processo de Avaliação

O *Processo de Avaliação* dos dados (representado pela atividade *Avaliar* na Figura 3.1) consiste em realizar a associação dos resultados das métricas obtidos no *Processo de Extração*, com os atributos do modelo de qualidade previamente definidos. Esse processamento é realizado utilizando fórmulas matemáticas que fazem parte do modelo de qualidade, ou seja, são essas fórmulas que realizam a ponderação entre as métricas internas e os atributos de qualidade. Os detalhes do *Processo de Avaliação* são apresentados na Seção 3.4.3.

Processo de Síntese

O *Processo de Síntese* dos resultados (representado pela atividade *Sintetizar* na Figura 3.1) concentra-se na elaboração dos relatórios de qualidade. Esses relatórios são gerados por meio dos valores obtidos durante o mapeamento entre as métricas internas com os atributos do modelo de qualidade. Os detalhes do *Processo de Síntese* são apresentados na Seção 3.4.4.

3.4 Detalhes do QM²

O *framework* QM² é composto por quatro processos conforme apresentado na Seção 3.3 (Figura 3.1). Esses processos associam-se com o objetivo de avaliar a qualidade dos modelos de um projeto de software. Cada processo é composto de atividades internas que são chamadas de processos ou atividades.

Para a avaliação do nível de qualidade dos modelos utilizando métricas e atributos de qualidade, é interessante a elaboração de modelos completos contendo todos os artefatos necessários. Nesse sentido, ao avaliar um projeto com a modelagem incompleta ou que tenha sido mal elaborado, os valores das métricas e a associação com os atributos de qualidade serão indicadores de que há problemas de modelagem. Por exemplo, os projetos OO devem conter os principais diagramas da UML. Por outro lado, nos modelos do domínio de sistemas embarcados, que são modelados em SysML, é essencial a presença dos principais diagramas da linguagem.

Conforme mencionado por Genero et al. (2003) as limitações nas pesquisas de avaliação de projetos, concentram-se na dificuldade de encontrar projetos completos e com dados concretos para serem avaliados com maior precisão. Os projetos de modelagem completos são importantes, pois a tendência esperada é que quanto mais projetos forem avaliados pelo *framework*, mais precisas serão as métricas definidas, impactando na exatidão dos modelos de qualidade.

Para a realização da avaliação o QM² requer alguns artefatos de entrada, são eles: (i) a especificação da modelagem do projeto; (ii) a especificação do modelo de qualidade; e, (iii) a definição das métricas. Os processos que envolvem esses artefatos são detalhados nas próximas seções.

3.4.1 Processo de Definição

As atividades que estabelecem o *Processo de Definição* (representado pelo processo *Definir* na Figura 3.1 e detalhado na Figura 3.2) estão relacionadas com a elaboração do plano de avaliação. O plano é elaborado a partir da definição do Modelo de Qualidade, o qual inclui a seleção dos atributos de qualidade essenciais para um domínio específico. Após a seleção do conjunto específico de atributos de qualidade, pode-se estabelecer o Modelo de Qualidade Inicial, que contém os atributos de qualidade que se pretende utilizar na avaliação e que podem ser relacionados com as métricas no processo *Mapear Métricas-AQ*. Esse trabalho é representado pelo processo *Definir MQ* da Figura 3.2, sendo que o *MQ* é a sigla para Modelo de Qualidade, e a sigla *AQ* é usada para designar Atributo de Qualidade.

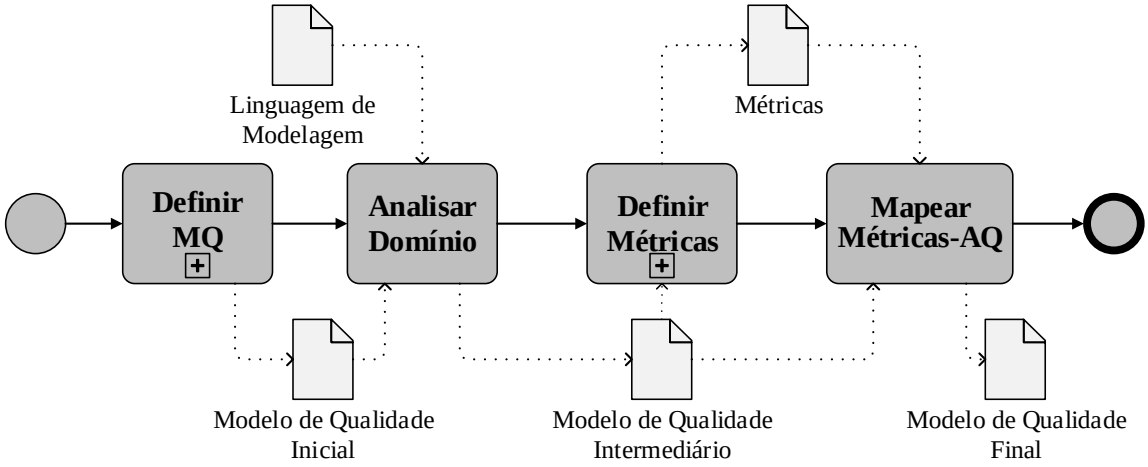


Figura 3.2: Processo de Definição.

Em seguida, é realizada a atividade de análise da linguagem de domínio, a qual corresponde à compreensão das estruturas da linguagem de modelagem utilizada na representação do projeto. Essa atividade é responsável por receber a lista de atributos do Modelo de Qualidade Inicial e realizar seu refinamento, isto é, verificar a possibilidade de realizar a definição das métricas para os atributos, levando em consideração os elementos estruturais e comportamentais da linguagem de modelagem. Caso não seja possível realizar a definição das métricas sobre os atributos do Modelos de Qualidade Inicial, o conjunto de atributos é ajustado, gerando um Modelo de Qualidade Intermediário, contendo os atributos passíveis de definição de métricas e avaliação. Esse trabalho é representado pela atividade *Analisar Domínio* contida na Figura 3.2.

Seguindo o fluxo do *Processo de Definição* do QM², tem-se o processo de definição das métricas que são associadas com os atributos de qualidade do modelo. Com base no

Modelo de Qualidade Intermediário recebido pelo processo *Definir Métricas*, gera-se um artefato com a especificação das métricas. De um modo geral, são definidas as operações e os elementos da linguagem que a avaliação contemplará. O processo que representa a definição de métricas é o processo *Definir Métricas* da Figura 3.2.

A última atividade desse processo é o mapeamento entre as métricas definidas no processo *Definir Métricas* com os atributos do Modelo de Qualidade Intermediário. Nesse processo define-se o Modelo de Qualidade Final, que contém o mapeamento das métricas com os atributos de qualidade final e a ponderação das métricas sobre cada um dos atributos. Esse mapeamento é representado pela atividade *Mapear Métricas–AQ* contido na Figura 3.2.

A seguir, apresenta-se uma descrição detalhada de cada uma das atividades (processos e atividades) do *Processo de Definição*.

Definição do Modelo de Qualidade – "*Definir MQ*" na Figura 3.2

Os modelos de qualidade são definidos para auxiliar a condução das atividades de melhoria de processos e avaliação de produtos das empresas. Esses modelos são utilizados para apoiar o aumento no nível de qualidade do gerenciamento, aquisição e manutenção de software (DROMEY, 1995).

De acordo com Kitchenham e Pfleeger (1996), os modelos de qualidade são usados em várias etapas do processo de desenvolvimento de software. Na etapa de engenharia de requisitos, são definidos os fatores de qualidade e os requisitos para o planejamento do software. Na etapa de desenvolvimento, os modelos de qualidade servem como base para a modelagem e implementação. Além de serem usados nas etapas de projeto e desenvolvimento, também podem ser usados na fase de verificação e validação do software.

Conforme Dromey (1995), os modelos de qualidade promovem recomendações diretas sobre a modelagem do projeto e elaboram abordagens construtivas para alcançar o alto nível de qualidade. Além disso, os defeitos encontrados referentes à qualidade são classificados de acordo com o modelo de qualidade.

Os modelos de qualidade descrevem as características de qualidade por meio de conjuntos de atributos, que são definidos com base no domínio especifico. Os fatores de qualidade podem ser classificados como atributos internos e atributos externos.

Os atributos internos são referentes ao nível de projeto e desenvolvimento, e são obtidos sem a execução do software (ISO/IEC, 2001). É possível determinar vários atributos

internos, tais como: acoplamento e coesão entre os elementos; tamanho do projeto (número de linhas de código ou número de classes do modelo); complexidade (número de pontos de decisão do código ou do modelo); ou, a dependência entre os módulos do software.

Esses são os fatores perceptíveis nos artefatos intermediários do projeto de software. Para realizar essa tarefa são definidos indicadores e métricas internas de avaliação. Os valores das métricas internas são obtidos a partir dos artefatos intermediários e têm como objetivo realizar previsões sobre o nível de qualidade do produto final por meio das características internas do projeto. Essa avaliação é realizada sobre os artefatos estáticos do projeto, tais como, diagramas em diversos níveis de abstração.

Os atributos externos de qualidade, por sua vez, somente podem ser obtidos com a execução do software (ISO/IEC, 2001). Por exemplo, pode-se verificar a dificuldade ou facilidade de navegação entre as telas, a quantidade de tempo necessário para recuperação de informações no banco de dados ou o número de falhas ocorridas durante a execução do software. De acordo com Meyer (1988), os atributos externos têm importância para o produto final, mas são os atributos internos (métricas internas) que asseguram a qualidade dos atributos externos, por meio de técnicas de avaliação aplicadas aos atributos internos.

Portanto, os atributos internos e externos estão estritamente relacionados. Por exemplo, os atributos externos *confiabilidade*, *manutenibilidade*, *usabilidade* e *integridade*, estão ligados com métricas internas que estão associadas com o código, documentação e outros artefatos que dão apoio ao desenvolvimento.

No *framework* proposto neste trabalho, segue-se uma abordagem similar à proposta por Fenton e Bieman (2014), na qual a primeira iniciativa para a definição do Modelo de Qualidade (Figura 3.3) dá-se por meio da seleção de um modelo existente.

Essa maneira de definição é utilizada quando o conjunto de atributos externos que influenciam as métricas internas atende por completo as necessidades do domínio alvo de avaliação. Entretanto, nos casos em que o Modelo de Qualidade não atende as exigências do domínio, um Modelo de Qualidade personalizado é definido baseando-se em modelos, técnicas e abordagens já existentes. Para a associação das métricas internas com os atributos externos de qualidade, ou a definição por completo de um novo Modelo de Qualidade, o QM² inclui os processos necessários.

Quando um Modelo de Qualidade é personalizado, conforme as necessidades específicas do domínio, são destacados somente os atributos de qualidade relevantes para a

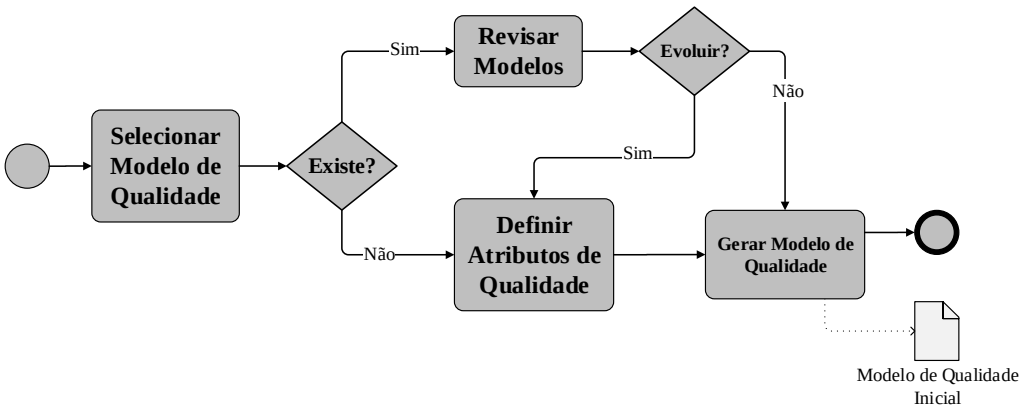


Figura 3.3: Processo de Definição do Modelo de Qualidade.

avaliação. Ressalta-se que essa alteração reflete diretamente na atividade de definição das métricas e no mapeamento, pois a definição das métricas internas é baseada no Modelo de Qualidade.

Conforme é ilustrado na Figura 3.3, o processo de definição de um Modelo de Qualidade é executado com sucesso quando se obtém um Modelo de Qualidade Inicial. Esse modelo é considerado *inicial* em consequência da dependência da atividade de definição das métricas e o mapeamento. Para essa seleção leva-se em consideração o domínio alvo de avaliação.

Nos casos em que o modelo de qualidade não existe, deve-se fazer a seleção e definir os atributos de qualidade do modelo de acordo com os interesses da avaliação (representadas pelas atividades *Selecionar Modelo de Qualidade* e *Definir Atributos de Qualidade* na Figura 3.3). Entretanto, nos casos em que o modelo de qualidade já tenha uma definição, tem-se a opção de revisá-lo (representada pela atividade *Revisar Modelos* na Figura 3.3). A evolução do modelo de qualidade consiste basicamente em remover, alterar ou adicionar atributos a esse modelo.

Análise da Linguagem de Domínio – "*Analisar Domínio*" na Figura 3.2

A tarefa *Analisar Domínio* da Figura 3.2, referente ao *Processo de Definição*, aborda as necessidades de compreensão adequada do domínio sobre o qual está sendo realizada a avaliação. Ou seja, é preciso entender a semântica e a sintaxe dos elementos definidos em uma determinada linguagem de modelagem. Dessa maneira, a atividade auxilia na compreensão da estrutura da linguagem e sobretudo na definição de métricas, baseadas nas informações obtidas dos elementos da linguagem.

O intuito dessa tarefa é basicamente identificar quais são as medidas que podem ser

extraídas de cada elemento da modelagem e estabelecer os objetivos da medição. Isso é essencial para a definição das métricas nas próximas atividades e também para um refinamento do Modelo de Qualidade.

Por exemplo, ao executar a avaliação de diagramas de classes da UML, deve-se ter conhecimento prévio dos seus elementos. De modo geral, conhecer as informações estruturais de uma classe. Isto é, saber que uma classe possui atributos, operações e relacionamentos, então definir as métricas sobre essas informações.

Em outro exemplo, considerando o domínio de Sistemas Embarcados modelados com a linguagem SysML, em particular o diagrama de definição de blocos, o conhecimento prévio das propriedades desse diagrama é fundamental, pois um bloco é estruturado por valores, operações, relacionamentos e portas de entrada e saída. Sendo assim, pode-se estabelecer métricas para esses elementos do diagrama.

O artefato de saída dessa tarefa é um Modelo de Qualidade Intermediário. Nesse artefato são enumerados os atributos de qualidade que são passíveis de serem calculados mediante os resultados da análise da linguagem de domínio. Com base nessa lista de atributos pode-se definir as métricas.

Definição das Métricas – "*Definir Métricas*" na Figura 3.2

O processo de definição de métricas firma-se nos objetivos da medição. Esse processo aborda a identificação dos conjuntos de métricas (do inglês, *metrics suites*) aplicáveis ao domínio desejado. Assim como os modelos de qualidade, as métricas podem ser encontradas na literatura e, se necessário, podem ser revisadas, adequando-se ao domínio de aplicação desejado. Nessa revisão, as métricas são manipuladas para atingir os objetivos de medição previamente estabelecidos.

Fenton e Bieman (2014) destacam que as métricas são importantes no processo de desenvolvimento de software, para auxiliar nas tomadas de decisões e na identificação das possíveis melhorias nos artefatos do projeto, consequentemente no resultado final do software.

De acordo com Wang (1999), o processo de avaliação de software, nas primeiras etapas de desenvolvimento, é essencial quando inclui a coleta de métricas em vários elementos do produto. Portanto, o processo de avaliação deve contemplar a seleção ou definição de métricas, levando em consideração um modelo de qualidade e selecionar apenas as métricas de interesse para a avaliação. Além disso, deve-se considerar o conjunto limitado de técni-

cas para a coleta de métricas e o relacionamento entre essas métricas e as características de qualidade (KAN, 2014).

Uma das principais atividades do QM² é a definição das métricas. Trata-se da seleção de métricas da literatura ou especificação de métricas para aplicações específicas. Essa definição pode ser realizada para diferentes domínios, desde métricas para modelos UML de sistemas de informação convencionais até os modelos de sistemas embarcados. Isso inclui os perfis que são criados para a modelagem de diferentes domínios, tal como o perfil *SysML* estendido da UML para a representação de Sistemas Embarcados e sistemas de tempo real.

Dentre as métricas para modelos UML, citam-se as definidas por Aggarwal et al. (2007), as quais têm como objetivo medir diagramas de classes, utilizando métricas de tamanho e complexidade estrutural, para avaliar o atributo externo de manutenibilidade desses diagramas. Destaca-se também as métricas CK de Chidamber e Kemerer (1994), as métricas de complexidade definidas por Henderson-Sellers (1996), as métricas elaboradas por Dallal e Briand (2010), a métrica *QI* definida por Badri et al. (2009) e as métricas de Aggarwal et al. (2007). Esses são alguns exemplos que se destacam devido às suas fundamentações teóricas e empíricas. Além disso, essas métricas apresentam uma facilidade de aplicação nos diagramas e análise dos indicadores.

No domínio de sistemas embarcados, cita-se o estudo elaborado por Antonio et al. (2014), que define um conjunto de métricas para a análise de compreensibilidade dos modelos *Simulink*[2]. Similarmente, Wei et al. (2012) especificam métricas para analisar modelos *Simulink/Stateflow*, incluindo um modelo de qualidade baseado na norma ISO/IEC 9126. Ahrens et al. (2011), por sua vez, definem um conjunto de métricas objetivas para avaliação de arquiteturas de sistemas modelados em UML (alto nível de abstração), ASCET[3] e Simulink (baixo nível).

Na Figura 3.4 apresentam-se os detalhes das atividades do processo de *Definição de Métricas* do QM². Esse processo inicia com a atividade *Selecionar Métricas*, as quais podem ser extraídas da literatura ou estabelecidas para um domínio específico, ressaltando a importância de validar as métricas teórica e empiricamente. Em seguida, as métricas selecionadas são especificadas na atividade de *Especificar Métricas*, sendo que essa especificação alinha-se com a maneira que serão extraídas dos diagramas. Porém, se as métricas já estiverem definidas no QM², tem-se a opção de revisá-las (atividade *Revisar Métricas*

[2] `http://www.mathworks.com` – Acessado em Setembro de 2014.
[3] `http://www.etas.com` – Acessado em Setembro de 2014.

da Figura 3.4). Nessa revisão, pode-se evoluir a métrica realizando adição, alteração e remoção de informações.

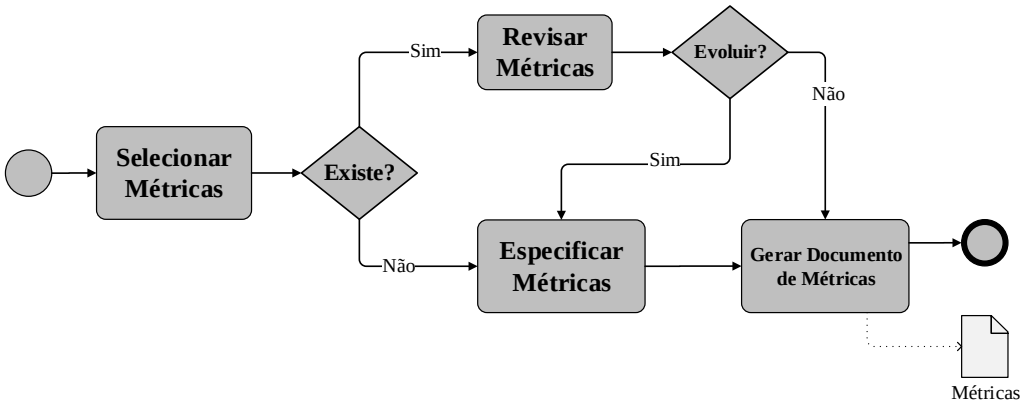


Figura 3.4: Processo de Definição de Métricas.

Nota-se que o processo de definição de métricas é concluído com sucesso quando se obtém um conjunto de métricas para o domínio desejado. Esse conjunto é utilizado na próxima atividade para realizar o mapeamento das métricas com os atributos do modelo de qualidade. Sendo assim, é possível representar diferentes conjuntos de métricas no QM².

Mapeamento (Métricas Internas e Atributos Externos) – "*Mapear Métricas–AQ*" na Figura 3.2

De acordo com a Figura 3.2, o último passo do *Processo de Definição* é o mapeamento entre os atributos do Modelo de Qualidade com as métricas internas. Esse mapeamento é representado pela tarefa *Mapear Métricas–AQ* da Figura 3.2.

A associação é realizada de maneira a atender às especificações do domínio alvo de avaliação, levando em consideração o Modelo de Qualidade Intermediário, que foi definido na tarefa *Analisar Domínio* (Figura 3.2). Além disso, o mapeamento necessita do conjunto de métricas definido no processo *Definir Métricas* (Figura 3.2). A tarefa *Mapear Métricas–AQ* tem por finalidade elaborar um Modelo de Qualidade Final, o qual consiste no mapeamento das métricas com os atributos de qualidade e os possíveis pesos.

Durante a associação das métricas com os atributos de qualidade, podem ocorrer casos em que um determinado atributo é associado a várias métricas. Quando isso acontecer, deverá ser estabelecida uma relevância de cada métrica para a composição final do valor do atributo de qualidade. Para isso, uma possível solução é atribuir pesos para cada associação entre as métricas e os atributos de qualidade.

Os valores para os pesos podem ser extraídos da experiência dos avaliadores, assim como, encontrados na literatura. Cita-se, como exemplo, o trabalho de Bansiya e Davis (2002), que define um modelo de qualidade denominado *Quality Model for Object-Oriented Design (QMOOD)* e realiza o mapeamento ponderado entre as métricas internas, extraídas de diagramas de classes, com os atributos do modelo de qualidade. No contexto de Sistemas Embarcados, cita-se o trabalho realizado por Ahrens et al. (2011), no qual define-se métricas de compreensibilidade arquitetural dos modelos de projetos, as quais são associadas aos atributos de qualidade por meio de fórmulas matemáticas ponderadas.

Na Figura 3.5 apresenta-se um exemplo hipotético de mapeamento entre métricas internas e atributos de qualidade externos. Nesse exemplo, o *Atributo 1* pode ser composto de N métricas (em particular, *Métrica 1* e *Métrica 2*). São essas métricas que definem o nível de qualidade do atributo, cada uma com um impacto diferente nos resultados, dependendo dos pesos atribuídos a cada uma delas. Nota-se que uma mesma métrica pode ser relacionada com mais de um atributo e com pesos diferente nas associações. Por exemplo, a *Métrica 2* tem peso "1" para a associação com o "Atributo 1" e peso "3" na associação com o "Atributo 2". Isso acontece devido à importância que cada métrica tem para a composição do atributo de qualidade.

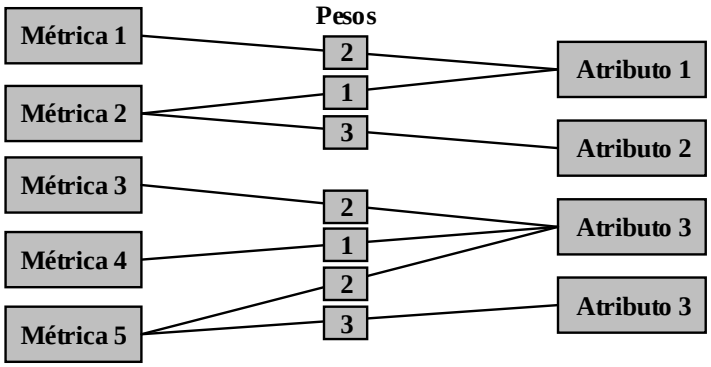


Figura 3.5: Exemplo de Mapeamento.

A seleção das métricas internas está fortemente relacionada com os atributos de qualidade, pois é por meio da associação e ponderação das métricas que se obtém os valores para o nível de qualidade. As propriedades internas normalmente são ponderadas de acordo com os atributos de qualidade e a definição do modelo de qualidade, levando em consideração o domínio de aplicação do projeto. Esse mapeamento normalmente não é trivial, pois trata-se da avaliação de projetos em nível de *design* e cada projeto tem suas especificidades.

3.4.2 Processo de Extração

O *Processo de Extração* apresentado na Figura 3.6, é composto pela atividade de *Extração de Métricas* (representada pela atividade *Extrair*), seguida pela atividade de *Armazenamento* dos resultados das métricas (representada pela atividade *Armazenar*). O processo depende de dois artefatos de entrada: o Modelo do Projeto alvo da extração e das métricas. Esse processo é concluído com sucesso quando se obtém o armazenamento dos resultados das métricas.

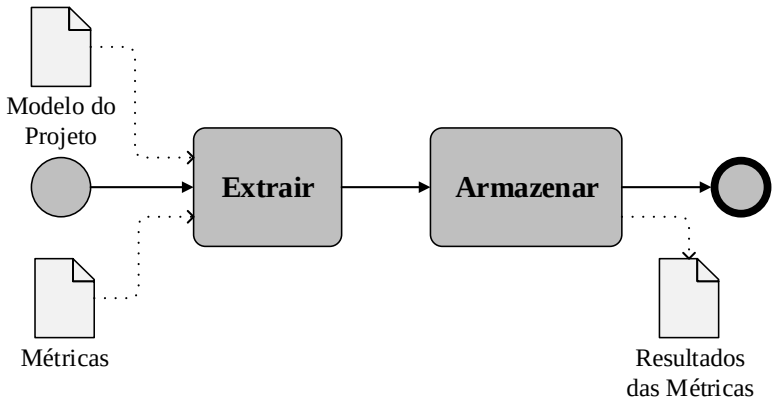


Figura 3.6: Processo de Extração de Métricas.

Extração das Métricas

A extração das métricas (atividade *Extrair* da Figura 3.6) é fundamental para o processo de avaliação e, consequentemente, para atribuição do nível de qualidade a projetos de software. Essa atividade tem como objetivo realizar a coleta das métricas, ou seja, é nesse passo do processo de avaliação que são extraídas as informações dos elementos dos modelos. Para tal, essa atividade exige a entrada do conjunto de diagramas representado em uma linguagem de modelagem e a especificação das métricas a serem coletadas. Ao final da execução da atividade de extração, obtém-se os valores das métricas, que são transferidos para a atividade responsável por realizar o armazenamento.

Armazenamento dos Resultados das Métricas

Após a extração das métricas, realiza-se o armazenamento dos resultados (representada pela atividade *Armazenar* da Figura 3.6). Essa atividade consiste em gerar os dados no formato desejado. Existem vários formatos possíveis para o armazenamento dos resultados das métricas, tais como, os arquivos XML, planilhas e banco de dados. É importante

que os resultados sejam armazenados em formato conveniente para que possam ser acessados nas próximas atividades do processo de avaliação.

3.4.3 Processo de Avaliação

O processo de avaliação é composto por uma única tarefa. Por razões de consistência na nomenclatura, optou-se por utilizar o termo *Avaliar* para nomear tanto o processo (Figura 3.1) quanto a tarefa (Figura 3.7).

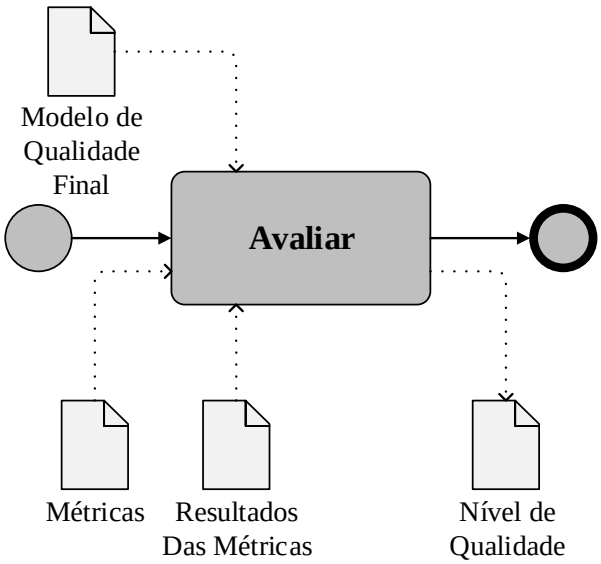


Figura 3.7: Processo de Avaliação.

A atividade para a atribuição do nível de qualidade ao conjunto de modelos é composta por várias características. Os atributos de qualidade podem ser interpretados por diferentes perspectivas, de acordo com os interesses e o domínio do projeto. Desse modo, engenheiros de software elaboram modelos de qualidade para a estimativa dos atributos de qualidade externos por meio das propriedades internas. Apesar de todos os *stakeholders* envolvidos no desenvolvimento dos sistemas concordarem que o produto final tem que ter um nível de qualidade alto, muitas vezes há divergência na forma em que a avaliação e os atributos de qualidade são associados.

Contudo, o processo de avaliação consiste em atribuir um nível de qualidade para cada atributo do modelo de qualidade. Essa atribuição é realizada com base nas atividades executadas no processo de definição e nos resultados obtidos no processo de extração dos dados.

A atividade de atribuição do nível de qualidade depende dos artefatos de entrada, sendo eles: Modelo de Qualidade Final; Métricas; e, Resultados das Métricas obtidos no

processo de extração. O resultado esperado é o conjunto de valores que representam o nível de qualidade do projeto em avaliação. No processamento do Nível de Qualidade os resultados das métricas são agrupados e manipulados, com o intuito de atender às especificações do Modelo de Qualidade Final.

3.4.4 Processo de Síntese

Similarmente ao processo de avaliação, o processo de síntese também é composto por uma única tarefa. Assim, optou-se por utilizar o termo *Sintetizar* para nomear tanto o processo (Figura 3.1) quanto a tarefa (Figura 3.8).

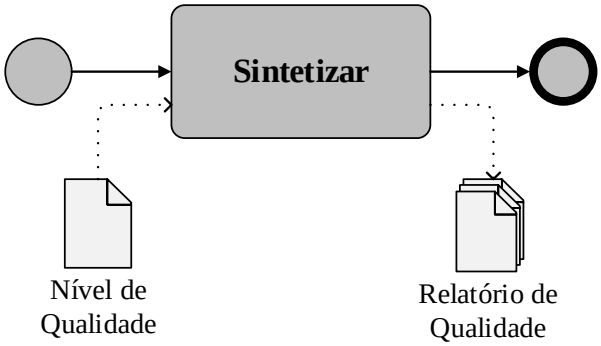


Figura 3.8: Processo de Síntese.

O processo de síntese consiste na atividade de elaboração dos relatórios com o nível de qualidade de cada atributo presente no modelo de qualidade. Esses relatórios são essenciais para a interpretação dos dados, pois dependendo da forma que os resultados são apresentados, pode-se levar a conclusões equivocadas.

Na Figura 3.8 ilustram-se a atividade *Sintetizar* e as entradas e saídas da atividade para elaborar os relatórios que fazem parte do processo de síntese dos resultados. Os relatórios são basicamente a apresentação dos dados obtidos no processamento das métricas que foram avaliadas e estão no conjunto de artefatos que representam o nível de qualidade produzido pelo processo de *Avaliação*.

3.5 Considerações Finais

Neste capítulo, apresentou-se o *framework* conceitual QM^2, proposto para guiar a avaliação do nível de qualidade de modelos (em nível de projeto) por meio de métricas internas e a associação com atributos de qualidade. O QM^2 apresenta um conjunto de definições necessárias para a avaliação de projetos em um determinado domínio de aplicação.

Os processos do QM^2 têm o intuito de orientar as definições dos documentos, artefatos e atividades a serem desenvolvidas.

Inicialmente, o *Processo de Definição* conduz as atividades para estabelecer o modelo de qualidade final. As atividades desse processo visam estabelecer uma avaliação completa, contemplando o conjunto de atributos de qualidade, suas respectivas métricas internas e o mapeamento ponderado. O *Processo de Extração* está relacionado com as coletas das métricas dos modelos e o armazenamento dos resultados. O *Processo de Avaliação* é executado sobre os resultados das métricas com base no modelo de qualidade final definido. Por fim, o *Processo de Síntese* tem como responsabilidade apresentar os relatórios finais da avaliação, levando em consideração os resultados do nível de qualidade obtido.

No próximo capítulo são apresentados os estudos investigativos para estabelecer os mecanismos automatizados, com o intuito de desenvolver um ambiente de apoio completo ao processo de avaliação definido pelo QM^2.

Capítulo 4

Automatização do *Framework QM*^{2}

4.1 Considerações Iniciais

O processo de avaliação de projetos de software, em geral, é de alto custo e requer muito tempo para ser concluído. Assim, as ferramentas de apoio à avaliação são de alta relevância. Além disso, à medida que aumenta a complexidade do projeto de software, as atividades de extração e manipulação de medidas, a partir de modelos do projeto, tornam-se impraticáveis de serem executadas manualmente.

Levando-se essas questões em consideração, este capítulo descreve estudos sobre ferramentas reutilizadas e construídas para auxiliar no processo de avaliação dos modelos de software em nível de projeto, fornecendo apoio ao *framework* QM^2 definido no Capítulo 3. Nesse sentido, na Seção 4.2 discute-se a questão de se desenvolver ou reutilizar ferramentas de apoio à Engenharia de Software, com foco em ferramentas para coleta e gerenciamento de métricas. Em seguida, na Seção 4.3 são apresentados os tipos de ferramentas de métricas necessárias para o propósito deste trabalho, tais como, mecanismos para a coleta de métricas, ferramentas para controle e gerenciamento de projetos de avaliação e ferramentas para geração de relatórios de qualidade do nível de qualidade. Em seguida, nas Seções 4.4 e 4.5 descrevem-se os mecanismos reutilizados e desenvolvidos neste trabalho. Alguns detalhes para a configuração e execução dos mecanismos também são apresentados. Por fim, na Seção 4.6 destaca-se uma análise do ambiente automatizado desenvolvido.

4.2 Definição de Mecanismos Automatizados: Desenvolver ou Reutilizar?

Inicialmente, uma das visões para se obter um processo de avaliação de produto de software automatizado é reutilizar ferramentas propostas por terceiros, porém encontrar ferramentas que atendam completamente a domínios específicos de avaliação é uma tarefa difícil. Por outro lado, tem-se a opção de desenvolver ferramentas. Diante desse impasse, discutem-se as duas visões para automatizar o processo de avaliação proposto no QM^2.

A indústria de desenvolvimento de software adota métricas simples devido à facilidade para seu entendimento e coleta (FENTON; BIEMAN, 2014). O cenário ideal seria possuir ferramentas que executem a coleta de métricas e realizem a associação dos resultados coletados aos modelos de qualidade para a interpretação, tornando o entendimento mais fácil, ou seja, as métricas devem ser objetivas e fáceis de serem coletadas. Portanto, seu uso efetivo requer ferramentas que automatizem a análise e com isso permita a replicação das avaliações.

Para cada domínio de aplicação pode-se utilizar variadas linguagens para a representação dos projetos de software, cada uma com suas especificidades (BRIAND; WÜST, 2002). Dessa maneira, as ferramentas de métricas devem ser extensíveis e genéricas, possibilitando a avaliação de projetos elaborados em diferentes linguagens de modelagem, e fornecer mecanismos de apoio à definição das métricas, definição dos modelos de qualidade, definição e ajuste dos intervalos de qualidade, tendo em vista que os modelos de qualidade podem ser redefinidos e elaborados utilizando um conjunto diferente de métricas e atributos de qualidade (LINCKE et al., 2008). Além disso, as ferramentas de métricas devem possibilitar a seleção e definição de novos arranjos desejáveis de métricas, visando contemplar interesses particulares da avaliação do projeto e também permitir a utilização das métricas já definidas na literatura.

A definição dos intervalos deve ser passível de ajustes, pois o estabelecimento de múltiplos intervalos é necessário para fornecer diferentes interpretações sobre os valores das métricas (BADRI et al., 2009). Por exemplo, o *Eclipse Metrics*[1] permite configurar os valores mínimo e máximo do intervalo para cada métrica. Define-se assim um intervalo de aceitação, sendo que a ferramenta emite um alerta quando os resultados das métricas coletadas estão fora do intervalo definido como satisfatório. Os intervalos devem ser configuráveis, pois as métricas, em geral, não possuem valores de referência absolutos

[1] `http://eclipse-metrics.sourceforge.net/` – Acessado em Fevereiro de 2015.

e os valores considerados ideais podem variar de acordo com os fatores de qualidade estabelecidos, domínio de aplicação e linguagem de modelagem (MEIRELLES et al., 2010).

Outro requisito essencial para ferramentas de apoio ao processo de Engenharia de Software refere-se à comparação e ao controle da evolução das avaliações realizadas para o mesmo projeto. Mantém-se assim, um histórico de avaliação, auxiliando na análise, na interpretação e na tomada de decisões para a elaboração dos modelos de qualidade.

São explícitas as vantagens em desenvolver uma ferramenta de métricas quando os objetivos da avaliação são específicos ou necessita-se de uma análise avançada do projeto e essas funcionalidades não se encontram disponíveis em ferramentas existentes. Em contrapartida, o desenvolvimento de uma ferramenta gera custos, consumo de recursos e tempo. Sendo assim, é de suma relevância realizar uma análise antes de selecionar as ferramentas de avaliação de qualidade dos projetos, levando em consideração os propósitos específicos.

Uma alternativa seria reutilizar uma ou mais ferramentas para executar o processo de avaliação, sendo necessário realizar a integração dessas ferramentas para atender as necessidades da avaliação. É imprescindível determinar: a ferramenta responsável pela coleta das métricas; a ferramenta de gerenciamento e versionamento dos projetos de avaliação; e, por fim, a ferramenta responsável pelas atividades de mapeamento entre as métricas e os atributos do modelo de qualidade.

Para reutilizar uma ferramenta, parcialmente ou totalmente, é necessário realizar estudos prévios sobre as funcionalidades disponíveis e conhecer seus principais pontos de extensão. Com base nesse conhecimento, realiza-se uma série de configurações para que ocorra o correto funcionamento. Em casos particulares, necessita-se desenvolver mecanismos para a integração e transferência de dados.

A seguir, apresentam-se os tipos de ferramentas identificadas no contexto deste trabalho. São descritas as principais funcionalidades das ferramentas selecionadas, destacando as ferramentas indicadas para reutilização. Discute-se também sobre os aspectos de implementação para integrá-las.

4.3 Possibilidades de Reutilização e Discussão

As ferramentas de métricas descritas nesta seção são classificadas por suas funcionalidades e características, entre elas estão: (i) ferramentas de coleta de métricas, normal-

mente são módulos que implementam os *parsers* dos elementos para poder identificar as medidas, baseando-se em um conjunto de métricas; (ii) ferramentas que realizam o gerenciamento e versionamento dos projetos de avaliação; e (iii) ferramentas que realizam o mapeamento entre as métricas internas com os atributos externos do modelo de qualidade.

Em seguida são descritas as ferramentas candidatas para serem reutilizadas na automatização do processo descrito no Capítulo 3, e suas principais características.

4.3.1 Ferramentas de Coleta Automática de Métricas para Modelos

Nesta seção são descritos os estudos investigativos realizados para a identificação e seleção de ferramentas que executam a coleta automática de métricas de modelos. A busca por ferramentas, inicialmente, baseou-se em um Mapeamento Sistemático (MS) realizado pelo autor desta dissertação (SOUZA, 2013), cujo um dos objetivos foi buscar ferramentas para coleta de métricas para Sistemas Embarcados diretamente de modelos. Posteriormente, realizou-se uma pesquisa não sistemática na *Web* para identificar ferramentas de coletas de métricas de propósito geral.

No MS realizado como estudo preliminar desse trabalho, foram levantados estudos que descrevem características de ferramentas para computação de métricas em modelos e estudos que descrevem as ferramentas em si.

De acordo com Zhang e Wang (2004), as medições e métricas de software são aspectos fundamentais relacionados às questões quantitativas. As ferramentas de medição de software são focadas em aplicações de domínio específico, no qual geralmente não cobrem todas as medidas utilizadas na Engenharia de Software.

A ferramenta apresentada por Lind et al. (2011) realiza estimativas para tamanho de código. Essas estimativas são executadas com base em perfis UML, capturando as informações disponíveis nos modelos. Além disso, o estudo menciona a possibilidade de estender a ferramenta para outros componentes UML, para obter informações complementares.

Fenstermaker et al. (2000) apresentam um sistema para coletar métricas na fase de concepção e implementação. Os autores relatam que é fundamental que a medição de processo de *design* seja um pré-requisito para atingir a otimização da produtividade. O sistema consiste em um esquema de métricas padrão e reúne características de artefatos de projetos, processos de *design* e comunicação.

Zhang e Wang (2004) apresentam uma ferramenta chamada *Software Engineering Measurement Expert System Tool (SEMEST)*. A SEMEST aborda medidas de software baseada em outras ferramentas especializadas para fornecer um conjunto abrangente de medidas e métricas de software. Segundo os autores, a ferramenta apresenta um conjunto de extensões e uma base de conhecimento com métricas para diferentes domínios de aplicação. A SEMEST pode ser usada como uma ferramenta especializada para apoiar a implementação de medidas de engenharia de software, análise métrica e *benchmarking* em software industrial.

As três ferramentas mencionadas, encontradas por meio do MS, são geralmente IDEs, *plugins* ou módulos de ferramentas e apresentam recursos limitados. Consistem em extensões, módulos desenvolvidos e adaptações para processar modelos específicos aplicados em estudos individuais. Portanto, para reutilizar essas ferramentas, no contexto deste trabalho, possivelmente demandar-se-ia um trabalho de manutenção e adaptações dessas ferramentas.

Como no MS foi encontrado um número reduzido de estudos relacionados às ferramentas para coleta de métricas em modelos durante a fase de projeto, optou-se por realizar uma pesquisa na *Web*. Nessa pesquisa, foram encontradas algumas ferramentas que realizam a coleta de métricas em modelos. Mas, essas ferramentas extraem somente métricas de modelos do paradigma OO (Orientado a Objetos). A despeito disso, algumas dessas ferramentas encontradas podem ser estendidas e adaptadas para avaliar a qualidade dos modelos desenvolvidos em UML/SysML, sendo que um subconjunto dessas ferramentas executa extrações de métricas para ambas as linguagens de modelagem. Algumas ferramentas com as suas principais características são listadas a seguir:

- ***JMetrics*[2]*:*** Auxilia na coleta e na avaliação de métricas em projetos de software OO e extrai conjuntos de métricas de projetos, pacotes, métodos e atributos. Além disso, gera planilhas com os resultados das métricas. É disponibilizada como software livre, multiplataforma e desenvolvida em Java.

- ***EclipseMetric*[3]*:*** É um *plugin* do Eclipse que analisa classes Java e gera resultados de métricas sobre a qualidade do projeto para cada pacote. Calcula métricas de modelo somente para projetos desenvolvidos na plataforma *Eclipse*. Está licenciada como GPL, foi desenvolvida em Java e é multiplataforma. Praticamente com as

[2] `http://jmetric.sourceforge.net/` – Acessado em Fevereiro de 2015.
[3] `http://eclipse-metrics.sourceforge.net/` – Acessado em Fevereiro de 2015.

mesmas características da *EclipseMetric*, cita-se a ferramenta ***JDepend***[4]***.***

- ***Together***[5]***:*** É um ambiente de modelagem UML que permite o desenvolvimento de software OO desde a especificação de requisitos até o projeto de casos de teste. Apresenta um módulo de garantia de qualidade que é o responsável pelo cálculo das métricas. Um dos destaques é que possui um abrangente conjunto de métricas fixas, que incluem as métricas de Chidamber e Kemerer (1994), Lorenz e Kidd (1994), entre outras. O diagrama precisa ser desenvolvido na própria ferramenta para o cálculo das métricas, ou seja, ela não é genérica e nem livre.

- ***Objecteering***[6]***:*** Esta ferramenta de modelagem pode ser usada para coletar métricas em modelos de classe UML. Porém, os modelos devem ser construídos na própria ferramenta, ou seja, não é uma ferramenta genérica. Essa ferramenta não é livre, pois alguns recursos devem ser adquiridos separadamente. Opera nos sistemas operacionais Windows e Linux ainda com algumas restrições de distribuição.

- ***SDMetrics***[7]***:*** É uma ferramenta que apresenta recursos para computação de métricas para modelos UML/SysML, na qual pode-se testar e analisar a eficácia de modelos. As características da ferramenta incluem a capacidade de criar análises durante o projeto para assegurar a qualidade dos artefatos construídos, o que auxilia na detecção de problemas durante o processo. Também contém um mecanismo dentro da ferramenta que realiza a verificação de regras de *design*, que permite identificar automaticamente projetos incompletos, incorretos, incompatíveis ou duplicados.

Dentre as ferramentas identificadas nas pesquisas, tanto no MS quanto na busca na *Web*, a ferramenta *SDMetrics* apresentou um conjunto completo de funcionalidades para atender os objetivos deste trabalho.

A ferramenta contém a API *OpenCore*, que possui um mecanismo para verificar regras definidas pela equipe de projeto (do inglês, *designing*), um motor de métricas para cálculos e um analisador de modelos *XMI*. Sendo assim, proporciona a reutilização para coletar métricas de modelos UML e SysML, assim como, outras notações que possam ser representadas com base nos padrões *XML (eXtensible Markup Language)* e *XMI (XML Metadata Interchange)*. Desse modo, decidiu-se por reutilizar a API *OpenCore*, com base em estudos e testes realizados que são apresentados na Seção 4.4.1.

[4] `http://sourceforge.net/projects/jdepends/` – Acessado em Fevereiro de 2015.
[5] `http://www.borland.com/products/together/` – Acessado em Fevereiro de 2015.
[6] `http://www.objecteering.com/` – Acessado em Fevereiro de 2015.
[7] `http://www.sdmetrics.com/` – Acessado em Fevereiro de 2015.

4.3.2 Ferramentas de Gestão e Versionamento de Projetos de Métricas para Modelos

O comprometimento dos processos, estratégias e ferramentas com o gerenciamento e acompanhamento da evolução dos projetos, é um fator determinante durante a avaliação do nível de qualidade de projetos de software em nível de *design*. Essa é uma característica importante a ser considerada durante a seleção das ferramentas usadas para a avaliação de projetos.

A evolução dos resultados deve ser armazenada em uma *baseline* para permitir comparações entre os resultados dos projetos, pois, por meio dessa atividade iterativa podem-se inferir medidas, auxiliar no planejamento do desenvolvimento dos projetos, realizar refinamentos nas definições das métricas e ajustar os limiares das métricas e atributos de qualidade de acordo com o propósito específico do projeto (PRESSMAN, 2010).

O gerenciamento dos resultados das métricas de um projeto faz-se necessário, porque é com base na evolução que é possível ter conhecimento dos pontos fracos do projeto, isso possibilita a aplicação de estratégias ou técnicas que proporcionem melhorias, visto que os resultados das métricas são associados aos atributos de qualidade que integram e indicam o nível de qualidade dos modelos.

Esse gerenciamento permite o versionamento dos projetos de métricas extraídas. Trata-se de um ponto relevante, pois possibilita a realização de comparações entre projetos que abordam o mesmo escopo e domínio de aplicação. Além disso, permite a comparação com medidas de tendências, com modelos de referência e com catálogos de valores, no qual definem-se os intervalos de valores e tolerância que podem ser definidos previamente.

Se ao analisar os resultados da avaliação do projeto de modelagem e for constatado a necessidade de alteração de uma parte ou um componente da estrutura dos modelos, com o intuito de melhorar a qualidade ou adaptá-lo, realiza-se essas modificações e após esse procedimento o projeto é reavaliado e um novo conjunto de resultados é obtido. Portanto, com base nos resultados da primeira avaliação, sabe-se se houve evolução e quais foram às melhorias.

Com o objetivo de gerenciar a evolução do nível de qualidade dos projetos de cálculo de métricas, buscou-se por ferramentas que incluíssem essa funcionalidade. Inicialmente, foram analisadas as ferramentas que já realizam a coleta de métricas, isso pelo baixo impacto no desenvolvimento da funcionalidade de versionamento, pois como já coletam as métricas, basta realizar o versionamento dos projetos de avaliação. Entretanto, em uma

análise das ferramentas de coleta de métricas, verificou-se que elas não possuem recursos para o versionamento. Além disso, analisou-se os formatos em que essas ferramentas exportam os resultados, pois se tem como interesse ferramentas que gerem artefatos para a próxima etapa do processo de avaliação do nível de qualidade.

Dentre as funcionalidades esperadas para uma ferramenta de coleta de métricas, cita-se a capacidade de gerar sessões de avaliação. Essas sessões assemelham-se às atualizações (*commits*) registradas pelas ferramentas de apoio ao desenvolvimento, tais como, *SourceForge*[8], *Github*[9] e *GoogleCode*[10]. Outro exemplo, refere-se ao armazenamento de sessões de teste, como as criadas pela ferramenta EclEmma[11], a qual armazena as coberturas estruturais de código obtidas em cada execução dos casos de teste da aplicação em análise.

Visto que as ferramentas de métricas descritas não possuem capacidade de versionamento dos projetos de avaliação de modelos por meio de métricas, optou-se por desenvolver uma ferramenta Web para o gerenciamento das métricas obtidas na coleta, gerenciamento dos projetos e usuários. Os resultados das métricas são armazenados e podem ser processados posteriormente. Esse desenvolvimento englobou um amplo conjunto de estudos sobre APIs, *frameworks*, módulos de ferramentas e a definição da estratégia para realizar o armazenamento dos resultados. A ferramenta desenvolvida, cujos detalhes são apresentados na Seção 4.5.1, agrega ao processo de avaliação do nível de qualidade de modelos, a possibilidade da avaliação ser extensível e reutilizável. A ferramenta permite a avaliação de modelos de diversos domínios de aplicação e realiza múltiplas versões do mesmo projeto.

4.3.3 Ferramentas de Avaliação e Síntese

Os dois últimos processos do *framework* QM^2 (Capítulo 3, Seção 3.3) são os de avaliação e sintetização. Eles têm por finalidade receber as definições das métricas, as medidas coletadas e o modelo de qualidade, realizar o mapeamento das métricas com os atributos de qualidade, e elaborar os relatórios, apontando o nível de qualidade para cada atributo. Sendo assim, esta seção tem como objetivo descrever os possíveis mecanismos automáticos para a execução dessas atividades.

Desde a concepção dessa pesquisa de mestrado teve-se como base a utilização da ferramenta *Spago4Q* (Spago4Q, 2014) para gerenciar a definição dos modelos de qualidade,

[8] `http://sourceforge.net/` – Acessado em Fevereiro de 2015.
[9] `https://github.com/` – Acessado em Fevereiro de 2015.
[10] `https://code.google.com/` – Acessado em Fevereiro de 2015.
[11] `http://www.eclemma.org/` – Acessado em Fevereiro de 2015.

realizar o mapeamento das métricas com os atributos de qualidade e gerar os relatórios para interpretações. Consequentemente, estudaram-se as alternativas para realizar automaticamente os processos de avaliação e sintetização nessa ferramenta, que é uma aplicação desenvolvida durante o projeto Qualipso (Qualipso, 2009) e utiliza como base para o seu desenvolvimento o *SpagoBI*.

A plataforma *Qualipso*[12] – *Quality Plataform for Open Source* – oferece recursos que possibilitam a avaliação da qualidade de diversos tipos de projetos e, principalmente, propõe a definição e implementação de tecnologias, procedimentos, leis e políticas, com o objetivo de potencializar as práticas de desenvolvimento de software livre, tornando-as confiáveis, reconhecidas e estabelecidas na indústria. Os detalhes sobre a plataforma *Spago4Q* são apresentados nas Seção 4.4.2.

4.4 Mecanismos Reutilizados

4.4.1 *OpenCore*

A *SDMetrics* (WÜST, 2013) é uma ferramenta usada para avaliar a qualidade dos projetos OO modelados em UML, que permite sua extensão para avaliar modelos desenvolvidos utilizando perfis, tal como o perfil SysML.

A *OpenCore*, além de computar métricas para modelos UML e para modelagem de sistemas na linguagem SysML, pode ser reutilizada e estendida para outros contextos. Por exemplo, a *OpenCore*, em conjunto com a *SDMetrics*, foi utilizada em um trabalho que abordou a avaliação de arquitetura de linhas de produto de software (OLIVEIRA JR. et al., 2010), computando métricas aplicadas a modelos elaborados a partir de perfis UML. Dentre estas razões, a *OpenCore* foi objeto de estudo para que fosse possível realizar as adaptações necessárias e integrá-la aos objetivos deste trabalho de pesquisa.

A API *OpenCore* calcula vários tipos de medidas para projetos desenvolvidos em UML/SysML. Dentre elas estão as medidas de tamanho, *design*, acoplamento e complexidade. Essas medidas: (i) estabelecem o padrão de qualidade e pontos problemáticos no projeto; (ii) preveem qualidades relevantes do sistema, tais como propensão a falhas, análise dos aspectos para manutenção e esforços de testes; (iii) indicam a eficácia do sistema para garantir a qualidade, encontrar falhas em etapas iniciais do processo e economizar em custo de desenvolvimento; e, (iv) motivam o refinamento das estimativas de esforços

[12] `http://qualipso.icmc.usp.br/` – Acessado em Fevereiro de 2015.

de implementação e testes.

A *OpenCore* é a implementação em Java das principais funcionalidades da ferramenta *SDMetrics*, ou seja, é o núcleo da ferramenta. Considerado uma API, a *OpenCore* está disponível sob a licença *GNU Affero General Public License*[13].

As principais funcionalidades da *OpenCore* são:

- Parser *XMI* configurável para arquivos de entrada *XMI* 1.x/2.x;
- Motor de métricas para extração de medidas de diagramas; e,
- Mecanismo de regras para verificação de design dos projetos.

A distribuição livre dessa API fornece um conjunto completo de funcionalidades para importação de arquivos *XMI*, medição e verificação de regras. Além disso, a *OpenCore* inclui os códigos fonte Java, os casos de teste JUnit[14] e documentação da API.

Lista de Métricas

Wüst (2013) apresenta um conjunto de métricas para projetos UML e realiza a coleta dessas métricas para os seguintes artefatos: diagrama de pacote, classe, objeto, comunicação e composição, diagramas de interface, atividades e sequência, diagramas de casos de uso, máquina de estado, componentes, nós e métricas gerais do projeto. Esse conjunto de métricas pode ser reutilizado para aplicações específicas, além disso, podem ser utilizadas como base para a definição de novas métricas.

Na Tabela 4.1 é apresentado, como exemplo, um subconjunto de métricas calculadas pela *OpenCore* para diagramas de classe.

Funcionamento da *OpenCore*

Para definir e calcular métricas para *design* de software, a *OpenCore* necessita ter informações dos tipos de elementos e os relacionamentos que estão presentes na linguagem de domínio utilizada para a representação do projeto. Por exemplo, a UML utiliza o padrão do OMG (Object Management Group, 1997), assim como, um metamodelo definido baseado no *Meta Object Facility* (MOF) do OMG.

O *XMI* (*XML Metadata Interchange format*) é outro padrão importante para o mecanismo utilizado para a computação de métricas da *OpenCore*. Esses arquivos descrevem

[13] http://www.gnu.org/licenses/agpl.html – Acessado em Fevereiro de 2015.
[14] JUnit – `http://junit.org/` – Acessado em Outubro de 2014.

Tabela 4.1: Métricas para Diagramas de Classe – adaptadas de (WÜST, 2013).

Métrica	Categoria	Descrição
NumAttr	Tamanho	Número de atributos na classe.
NumOps	Tamanho	Número de operações em uma classe.
NumPubOps	Tamanho	Número de operações públicas da classe.
Setters	Tamanho	Número de operações iniciando com 'set'.
Getters	Tamanho	Número de operações iniciando com 'get'.
NumDesc	Herança	Número de descendentes da classe.
NumAnc	Herança	Número de ancestrais da classe.
Dep_Out	Acoplamento (importação)	Número de elementos que esta classe depende.
Dep_In	Acoplamento (exportação)	Número de elementos que dependem da classe.

o conjunto de regras para serializar os elementos para o padrão *XML* e geração dos DTD e XML *schema*. Porém, esse padrão não é utilizado diretamente para a definição das métricas, pelas seguintes razões (WÜST, 2013): (i) existem diferentes formatos de *XMI* (versões de *XMI* 1.x e 2.x) para diferentes representações de diagramas; (ii) os arquivos *XMI*, DTD e schemas *XML* são pesados para o simples propósito de definir métricas; e (iii) na prática, as ferramentas nem sempre aderem plenamente aos padrões de arquivos *XMI* para a representação de diagramas.

Essas são as razões pela qual a interoperabilidade de ferramentas, por meio de trocas de arquivos *XMI*, nem sempre funciona como deveria. Assim sendo, a *OpenCore* define um metamodelo simplificado e reduzido. Nesse metamodelo define-se o conjunto de elementos e todas as informações relevantes para calcular as métricas dos modelos. Além disso, no metamodelo simplificado abstraem-se as várias versões do *XMI* que são utilizadas para representar os modelos. Portanto, para suportar uma versão específica do *XMI*, precisa-se definir o mapeamento dos elementos do metamodelo, definido pela *OpenCore* para uma determinada versão do *XMI*. Esse mapeamento é realizado por meio de regras de transformações armazenadas em um arquivo específico para tal.

Ressalta-se que é necessário ter conhecimento prévio sobre os elementos do metamodelo para se definir novas métricas em diferentes domínios, pois as métricas definidas fazem referência aos elementos e relacionamentos contidos no metamodelo. Além disso, os conhecimentos sobre as regras de transformação são primordiais quando o arquivo exportado não está integralmente nos padrões *XMI*, não atende ao metamodelo definido e quando pretende-se definir novas métricas, levando em consideração extensões específicas do *XMI*.

De modo geral, essas extensões contêm informações adicionais de ferramentas para

representações personalizadas do modelo e informações de *layout* dos diagramas. No entanto, qualquer modelo baseado na MOF pode ser avaliado, pois é possível estender o metamodelo, alterar as regras de transformação e definir métricas para realizar medições sobre modelos representados em extensões específicas.

Na Figura 4.1, apresenta-se a estrutura geral do fluxo dos arquivos utilizados pela *OpenCore*. Esses arquivos são necessários nos processamentos intermediários da extração de métricas. O artefato *"UML Model"* representa o modelo do projeto construído em uma ferramenta de modelagem, que deve ser capaz de exportar arquivos em formato *XMI*. Em sequência, gera-se o artefato "*XMI Source File*", que é o arquivo contendo a representação da modelagem do projeto no formato aceito pelo *OpenCore*.

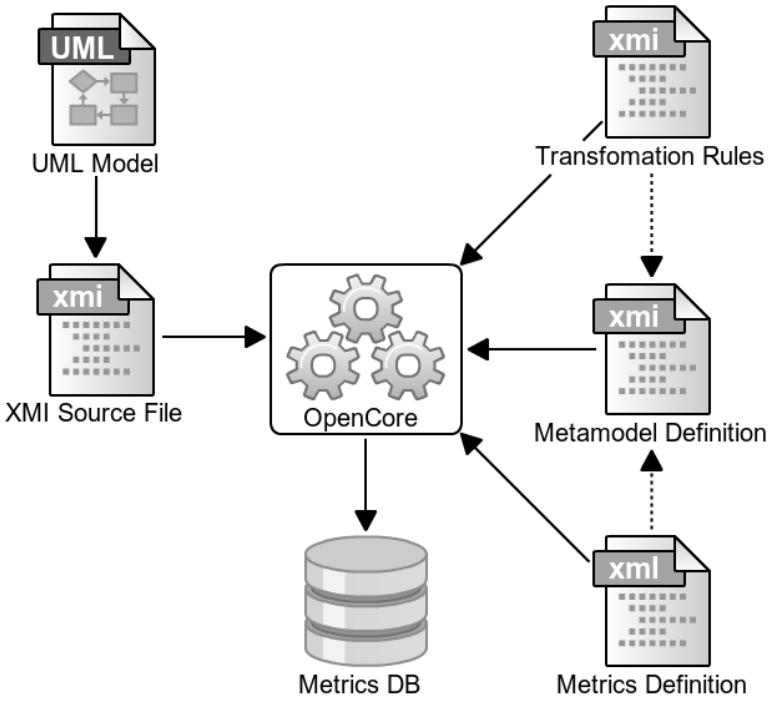


Figura 4.1: Fluxo de Arquivos da *OpenCore* – adaptado de Wüst (2013).

Em conformidade com a Figura 4.1, os arquivos de configuração requeridos pela *OpenCore* para a computação das métricas em modelos são: "*Transformation Rules*", que contém a especificação das regras de transformação; "*Metamodel Definition*", que contêm os elementos e informações relevantes sobre a linguagem de modelagem; e, "*Metrics Definition*", que inclui a definição completa das métricas a serem extraídas dos diagramas do projeto a ser avaliado. No Apêndice A, apresenta-se com mais detalhes como é estruturado cada um dos arquivos de configuração da *OpenCore*. O apêndice também contém explicações de como elaborá-los com base em exemplos de segmentos de códigos.

Na seção a seguir, descreve-se os estudos e testes realizados sobre a reutilização da

API *OpenCore*, assim como observações acerca das restrições e limitações da *OpenCore*. Ressalta-se que os testes executados são de viabilidades de integração da *OpenCore* com o ambiente de avaliação e não apresenta características experimentais.

Reutilização da *OpenCore*

Com a finalidade de verificar a compatibilidade da *OpenCore* com os arquivos gerados pelas ferramentas de modelagem de projetos de software, criou-se um conjunto de modelos UML para ser processado pela API. Esse conjunto de modelos inclui os principais diagramas da UML, tais como: Diagrama de Classe, Diagrama de Caso de Uso, Diagrama de Sequência, Diagrama de Atividade, Diagrama de Comunicação, Diagrama de Estado, Diagrama de Interface e Diagrama de Componente. Esses diagramas foram criados com o objetivo de verificar se a ferramenta exporta um arquivo *XMI* compatível com os arquivos de configuração da *OpenCore*, por exemplo, *Metamodel Definition*, *Transformation Rules* e *Metrics Definition* e se realiza a computação das métricas.

Em uma análise preliminar, foram encontradas ferramentas de modelagem com diversas características, então selecionou-se ferramentas com licenças acessíveis, tais como GPL, EPL e edições livre, que exportam para arquivos *XMI* e que fornecem apoio a especificação da UML 2.x.

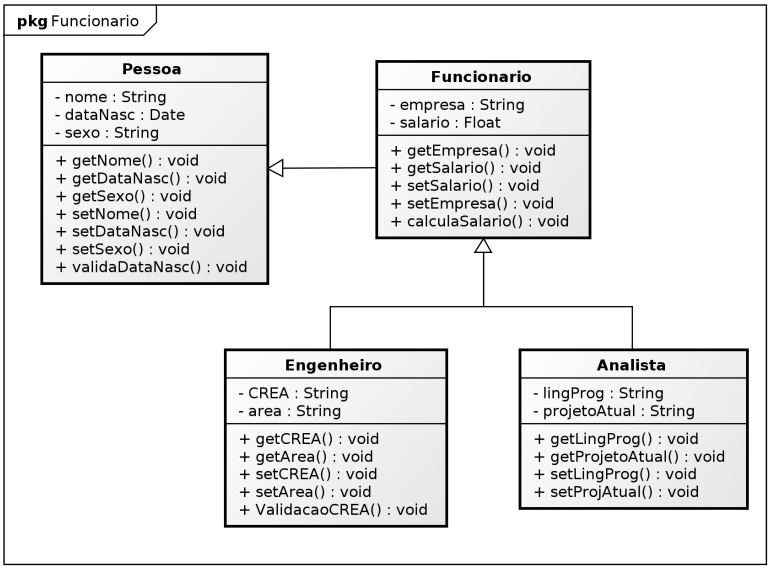
Os diagramas foram elaborados nas seguintes ferramentas:

- **StarUML**[15]**:** licença GPL, suporte a modelagem de UML2 e a MDA;
- **Modelio**[16]**:** licença GPL, com suporte a UML2, SysML, BPMN.
- **Poseidon**[17]**:** licença proprietária com versão de uso gratuito (*Free Community Edition*), suporta a especificação da UML2.1, SysML, BPMN.
- **Astah**[18]**:** licença proprietária com versão de uso gratuito (*Astah Community*), desenvolvida em java, suporta a especificaçãoda UML2.x, SysML, ER e Flowchart.

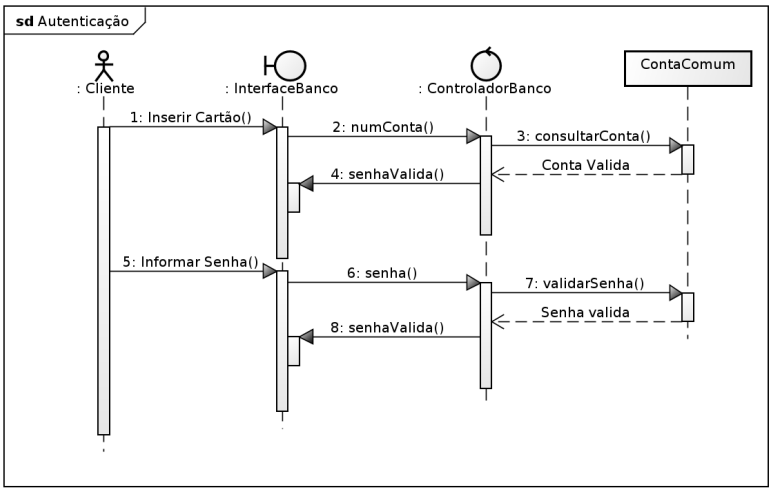
Os diagramas criados e exportados são simples, tais como os apresentados por Guedes (2011), conforme observa-se no diagrama de classe na Figura 4.2(a) e, em seguida, no diagrama de sequência na Figura 4.2(b). No próximo passo do teste, realizou-se a

[15] `http://sourceforge.net/projects/staruml` – Acessado em Fevereiro de 2015.
[16] `http://www.modelio.org` – Acessado em Fevereiro de 2015.
[17] `http://www.gentleware.com` – Acessado em Fevereiro de 2015.
[18] `http://astah.net/` – Acessado em Fevereiro de 2015.

importação dos arquivos *XMI* para a ferramenta *OpenCore* e executou-se uma sequência de computação das métricas. Na Tabela 4.2 apresentam-se os resultados.



(a) Diagrama de Classe.



(b) Diagrama de Sequência.

Figura 4.2: Exemplos Utilizado nos Testes – adaptados de (GUEDES, 2011).

Ao analisar os resultados dos testes, verifica-se que a maioria dos diagramas elaborados na ferramenta *Poseidon* não foi calculado, pois esses diagramas retornaram com erros de leitura ou tiveram resultados inválidos quando computadas as métricas contidas na especificação padrão da *OpenCore*. As ferramentas *StarUML e Modelio* tiveram seus diagramas computados pela *OpenCore*, mas nenhuma delas obteve sucesso total, pois foi constatado um problema comum relacionado com os diagramas de sequência e comunicação.

Tabela 4.2: Ferramentas de Modelagem e Diagramas Calculados pelo OpenCore.

Diagramas UML	StarUML	Modelio	Poseidon	Astah
Diagrama de Classe	✓	✓	✓	✓
Diagrama de Caso de Uso	✓	✓	✓	✓
Diagrama de Atividade	✓	✓		✓
Diagrama de Sequência				
Diagrama de Comunicação				✓
Diagrama de Estado	✓	✓		✓
Diagrama de Interface	✓	✓	✓	✓
Diagrama de Componente	✓	✓		✓
Diagrama de Pacote	✓	✓	✓	✓

Após verificar a conformidade entre os arquivos *XMI* exportados pelas ferramentas de modelagem e os suportados pela *OpenCore*, realizou-se outro estudo visando à inclusão, alteração e remoção de métricas, e a definição de novos arquivos de métricas. Para realizar a definição de novas métricas deve-se ter conhecimento da estrutura dos arquivos de configuração, definição de métricas da *OpenCore* e o modelo em formato *XMI*.

Nessa nova avaliação, o objeto de estudo é o arquivo de definição de métricas. Sendo assim, com base nas métricas existentes na documentação da *OpenCore*, definiu-se outro conjunto de métricas, que foi então computado a partir dos diagramas elaborados na avaliação anterior.

No Código 4.1, apresenta-se três métricas definidas como exemplo, no qual a primeira métrica trabalha com projeção de relação, a segunda realiza uma computação de métricas condicional e, por fim, uma métrica com projeção condicional sobre a propriedade do elemento.

```
1  <!-- Métrica retorna o número de elementos por classes -->
2  <metric name="NumElements" domain="class">
3    <description> O número de elementos da classe.</description>
4    <projection relation="context" />
5  </metric>
6
7  <!-- Métrica Condicional: Se o número de elementos for maior que
8    3 então divide por 2, caso contrário retorna 1 -->
9  <metric name="NumElementsSuperiorN" domain="class">
10   <description> O número de elementos na classe superior a 3.</description>
11   compoundmetric condition="NumElements>3" term="NumElements/2" alt="1"/>
12 </metric>
13
14 <!-- Número de atributos com prefixo Set -->
15 <metric name="SET" domain="class" category="Size">
16   <description> O número de operações iniciado com 'SET'.</description>
17   <projection relset="ownedattributes" target="property"
18     condition="name startswith 'set'"/>
19 </metric>
```

Código 4.1: Exemplo de Definição de Métricas em *XML*.

O intuito de realizar esse conjunto de análises é compreender os artefatos e arquivos de configuração da *OpenCore*, para então reutilizá-la no desenvolvimento do ambiente de avaliação de modelos. Para a integração de APIs ao ambiente, principalmente a *OpenCore*, identificou-se a necessidade de desenvolver um *"driver"* para simplificar e auxiliar nos procedimentos de transferência de arquivos entre a *OpenCore* e o restante do ambiente de avaliação.

A *OpenCore* é uma API com vários recursos para a verificação de regras e computação de métricas para modelos, mas esses recursos não são integralmente utilizados no desenvolvimento da ferramenta. Portanto, para reutilizar a *OpenCore*, foi desenvolvida uma camada intermediária utilizando o padrão de projeto *Facade* (GAMMA et al., 1995), o qual implementa uma interface para o restante da aplicação e restringe o acesso às funcionalidades da *API* do restante da aplicação.

Na Figura 4.3 apresentam-se, de maneira simplificada, as partes referentes à implementação realizada para a reutilização da *OpenCore*. A classe `OpenCoreFacade` implementa as interfaces para as invocações dos métodos da API *OpenCore*. As classes externas da *OpenCore* precisam apenas ter conhecimento das funcionalidades e dos parâmetros da classe `OpenCoreFacade`, não se preocupando com as estruturas internas da implementação da *OpenCore*.

Ao final da análise e testes realizados na *OpenCore*, pôde-se verificar a viabilidade de reutilizá-la para a definição e coleta de métricas para modelos. A integração da API ao ambiente será descrita na Seção 4.5.1, na qual são detalhados os aspectos de decisão de projeto e desenvolvimento.

4.4.2 *SpagoBI for Quality – Spago4Q*

O *Spago4Q– SpagoBI for Quality* – apresenta um conjunto de funcionalidades que permite a definição dos modelos de qualidade e a geração de vários tipos de relatórios. Tais relatórios geralmente são aplicados em processos de *Business Intelligence (BI)*, permitindo análises mais indutivas em suas interpretações. Está sob a licença GNU LGPL e oferece serviços para o desenvolvimento e manutenção dos projetos de software, usando uma infraestrutura de apoio com o propósito de gerenciar a prestação de serviços, levando em consideração os principais modelos de qualidade, tais como CMMI (CMMI Product Team, 2010), ISO 9001:2000 (ISO, 2008) e AQAP (AQAP, 2009).

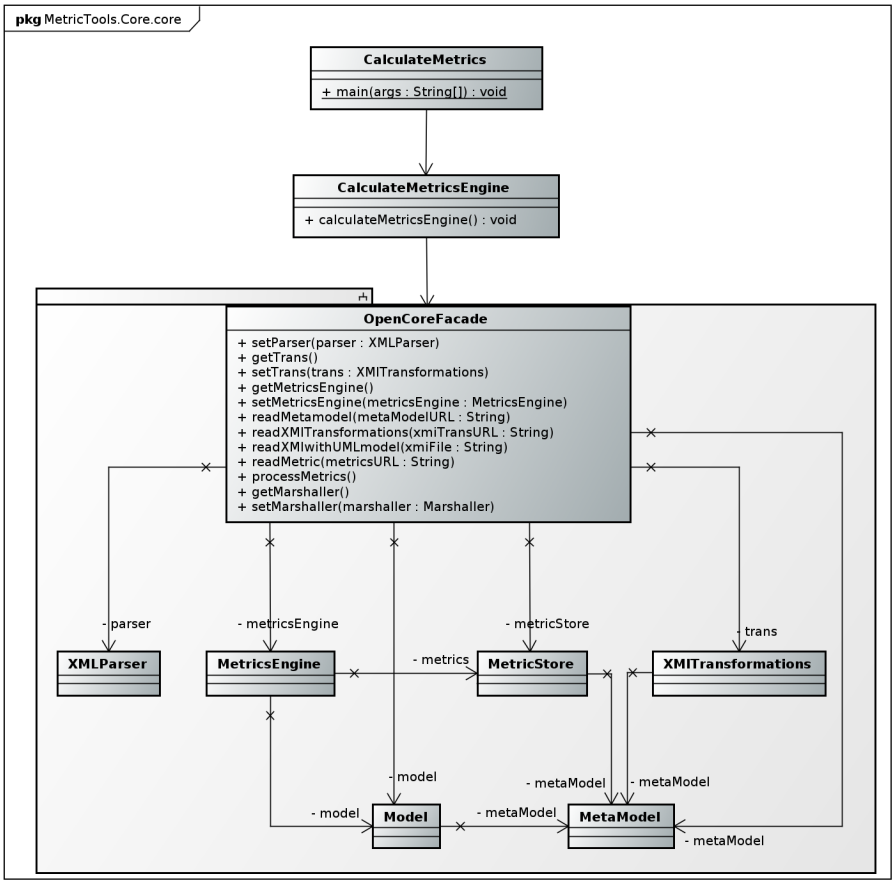


Figura 4.3: Padrão *Facade* Desenvolvido para Integrar a *OpenCore*.

Funcionamento do *Spago4Q*

A plataforma Spago4Q (2014) disponibiliza uma série de pacotes responsáveis pela extração das informações referentes aos projetos. Os extratores coletam dados diretamente dos processos e produto por meio dos principais ambientes, que são utilizados no desenvolvimento e em todo o ciclo de vida do software, assim como, ferramentas de modelagem, ferramentas de testes, ferramentas de análise de qualidade e *framework* de gestão.

A arquitetura do *Spago4Q* é estabelecida como uma especialização da plataforma SpagoBI (2014) – *Suite Open Source Business Intelligence*, projetada para ser facilmente reutilizada nos contextos organizacionais complexos. A plataforma fornece um metamodelo avançado que torna o *Spago4Q* integrável a outros ambientes (COLOMBO et al., 2008).

O *SpagoBI* é uma estrutura mais complexa do que o *Spago4Q*, para permitir a análise de BI. Além disso, disponibiliza um metamodelo completo para representação e descrição de processo de desenvolvimento genérico, *framework* de medição, extratores e *framework* de avaliação (BIANCO et al., 2010; PETRINJA et al., 2009).

Como a plataforma *Spago4Q* é software livre e fornece interfaces para a integração e reutilização das suas funcionalidades, então é possível adaptar extratores adicionais para realizar a coleta dos dados para determinados domínios (ARDAGNA et al., 2010; TRAVERSO-RIBóN et al., 2013).

Na Figura 4.4 apresenta-se a arquitetura geral para integração de extratores ao *Spago4Q*. De acordo com a arquitetura estabelecida, os dados são extraídos dos projetos ou repositórios por meio de extratores, que são responsáveis por distribuir os dados para o *Data Warehouse*. Em seguida, os mecanismos analíticos do *SpagoBI* realizam a geração dos relatórios e *dashboards*. Os *dashboards* (ou Painel de Controle) são recursos para a apresentação visual das informações mais relevantes, ajustadas em uma única tela para fácil acompanhamento das informações. Ressalta-se que neste trabalho, a etapa de extração que é contemplada pelo *Spago4Q* (parte superior esquerda da Figura 4.4) é realizada pela ferramenta *MMTool*, que é descrita na Seção 4.5.1.

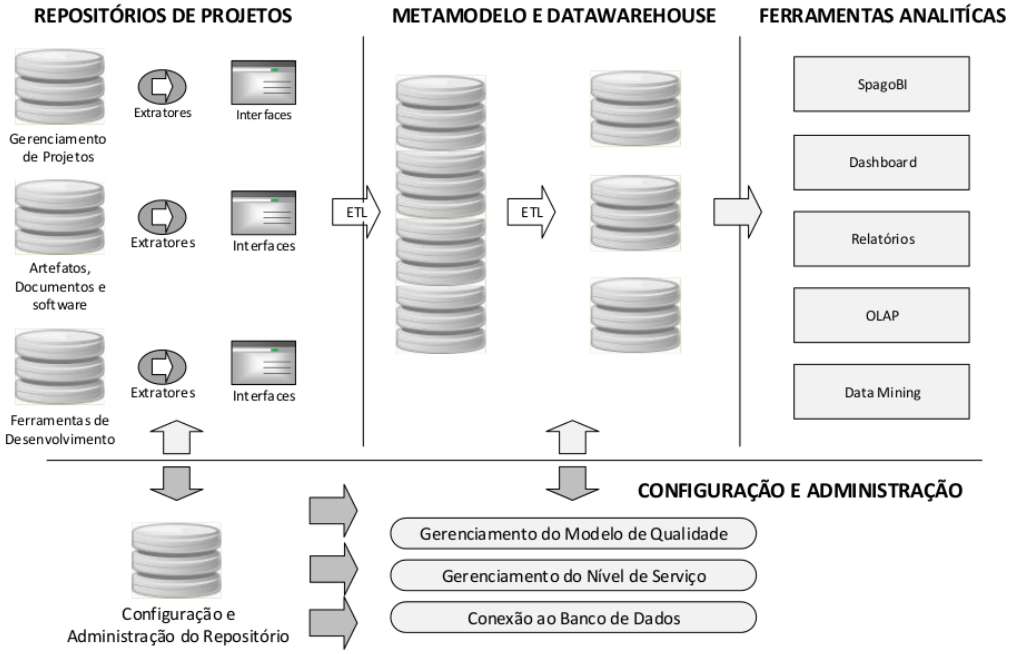


Figura 4.4: Arquitetura do *Spago4Q* – adaptada de Spago4Q (2014).

Reutilização da Plataforma *Spago4Q*

No contexto deste trabalho, a definição de um modelo de qualidade e o mapeamento das métricas com os atributos são processos fundamentais, conforme apresentado no processo definido pelo QM^2 no Capítulo 3. O objetivo é executar avaliações independentes do modelo de qualidade, no qual os modelos de qualidade de um domínio específico são

selecionados e definidos. A seguir, apresenta-se o roteiro para a definição do processo completo de avaliação e sintetização do QM^2.

Instalação, Configuração e Execução do *Spago4Q*

De acordo com a documentação disponível pela plataforma Spago4Q (2014), foi possível analisar o esforço necessário para sua reutilização. Basicamente, a reutilização do *Spago4Q* requer a instalação e a configuração de um conjunto de programas, as definições dos artefatos, a especificação dos mecanismos de conexão com banco de dados e a geração dos relatórios.

As aplicações de software necessárias para a execução da plataforma, são: *SpagoBI*, Java SE JDK[19], Apache Tomcat[20] e MySQL[21]. Tendo em vista que o *Spago4Q* é uma ferramenta executada sobre a plataforma *SpagoBI*, deve-se instalá-la e configurá-la. A plataforma *SpagoBI* é inteiramente desenvolvida em Java e utiliza o servidor de aplicação Web Apache Tomcat. O MySQL é o gerenciador de banco de dados utilizado pela plataforma e para integração via banco de dados.

Realizou-se um estudo piloto para investigar e prever problemas no fluxo de trabalho e na integração das ferramentas que coletam métricas com a plataforma *Spago4Q*. Esse estudo consistiu na definição de um modelo de qualidade, mapeamento das métricas com os atributos e geração de relatórios analíticos.

Após a instalação e configuração do ambiente, realizou-se as configurações dos mecanismos de definição para executar o processo de avaliação. Inicialmente, são configurados os mecanismos para a recuperação de dados dos projetos. Em seguida definem-se os modelos de qualidade, o mapeamento das métricas com os atributos externos e, por fim, são elaborados os documentos e relatórios. Para cada atividade deve-se configurar os seguintes mecanismos:

- Recuperação de dados dos projetos:

  Data Source, Data Set, definição das LOV (do inglês, *List Of Values*) e *Analitycal Driver*.

- Definição das métricas e mapeamento das métricas com os atributos:

  Thresholds, KPI *(Key Performance Indicator)*

[19] `http://www.oracle.com/br/index.html` – Acessado em Fevereiro de 2015.
[20] `http://tomcat.apache.org/` – Acessado em Fevereiro de 2015.
[21] `http://www.mysql.com/downloads/` – Acessado em Fevereiro de 2015.

- Definição do modelo de qualidade:

  Model Definition e Model Intance

- Elaboração de documentos e relatórios:

  Create Documents e Run Report

As etapas para a elaboração do processo de avaliação no *Spago4Q* são baseadas no ciclo PMAI (do inglês, *Plan-Measure-Assess-Improve*) conforme apresentado por Ruffatti (2013):

- **Planejamento:** é a definição do conjunto de métricas, definição dos intervalos das métricas, especificação matemática das métricas e os pesos para as métricas.
- **Medição:** é a coleta dos dados dos projetos, cálculo e normalização dos valores das métricas (valores devem estar entre 0 e 1, inclusive), e o cálculo de desempenho geral para o conjunto de métricas que compõem o valor final do atributo de qualidade.
- **Avaliação:** é a representação dos resultados em forma de relatórios para os analistas e gestores.
- **Melhorias:** estão relacionadas com as tomadas de decisão. Visam buscar soluções para melhorar a qualidade geral, quando os valores forem negativos, baixo ou fora do intervalo aceitável.

Relatórios KPI- *Key Performance Indicator*

A elaboração do relatório *KPI* é praticamente a definição do modelo de qualidade e o mapeamento das métricas com os atributos contidos no modelo. O *Spago4Q* fornece uma interface para essas definições, que são definidas por meio de *KPI's*.

A definição dos *KPI's* envolve a especificação de algoritmos ou consultas para o cálculo das métricas e para a recuperação em banco de dados. Quando esses cálculos são simples, pode-se recuperar por meio de sentenças SQL. Entretanto, em casos complexos cria-se uma classe Java *ad-hoc* para computá-los ou instanciar um *Web Service* para essa função.

O *KPI* realiza a recuperação diretamente do *Data Warehouse* do *Spago4Q*, que contém todos os dados dos projetos, os quais são obtidos por meio da extração de métricas. Na Figura 4.5 apresenta-se um exemplo de documento *KPI* construído no *Spago4Q*. Neste documento constam as informações dos projetos, os atributos do modelo, os valores de

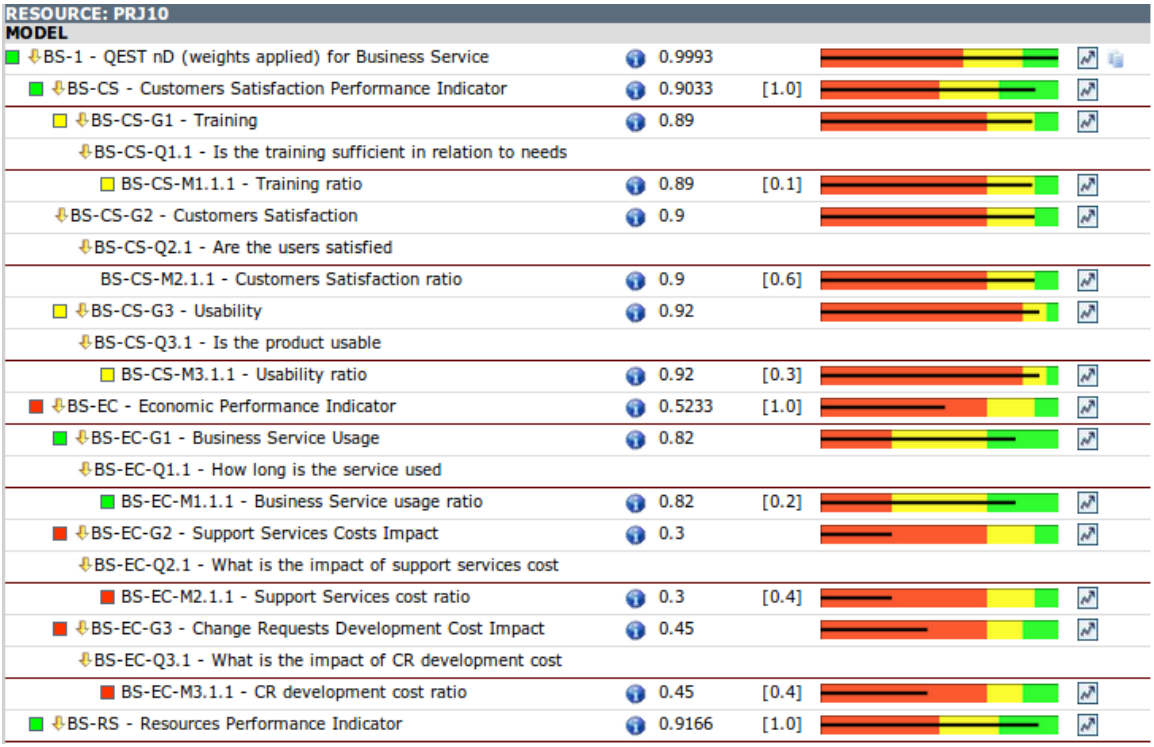


Figura 4.5: Exemplo de Relatório *KPI* – extraído de Spago4Q (2014).

cada métrica, os pesos para cada métrica, a representação gráfica dos *KPI's* e o acesso ao histórico de avaliação realizada para cada *KPI*.

Os chamados *KPI's* globais podem ser inseridos nos documentos *KPI*, sendo calculados pelo valor absoluto do *KPI*, isto é, o somatório do produto de cada métrica que pertence ao atributo ou fator de qualidade com seus respectivos pesos.

Os modelos de qualidade definidos podem exigir a ponderação (atribuição de pesos) para cada métrica correlacionada com os atributos de qualidade, indicando o impacto que cada métrica exerce sobre cada atributo. Essa etapa está contida no processo de definição do QM^2, apresentado na Seção 3.4.1 (Figura 3.2). As métricas que são mapeadas para os atributos de qualidade presentes no modelo, são extraídas diretamente do banco de dados pelo *Spago4Q*.

4.5 Mecanismos Desenvolvidos

Para a integração das ferramentas *OpenCore* e *Spago4Q* e criar o ambiente automatizado de avaliação da qualidade dos projetos de software, identificou-se a necessidade de desenvolver uma ferramenta responsável por: (i) ler os diagramas; (ii) gerenciar dos projetos avaliados; (iii) integrar o motor de métricas (*OpenCore*) reutilizado para coletar

as métricas dos modelos; e, (iv) estruturar e fornecer os dados no formato adequado para o *Spago4Q*. Para isso, desenvolveu-se a ferramenta ***MMTool*** – ***M**odel **M**etrics **Tool***. Os aspectos gerais que envolveram o desenvolvimento e a integração dessas ferramentas são apresentados a seguir.

4.5.1 *MMTool*

A ferramenta *MMTool*, desenvolvida para a extração das métricas e o gerenciamento dos projetos de avaliação, está dividida em dois módulos principais, cada um responsável por um conjunto de funcionalidades:

- O módulo ***MMTool*.Web** implementa os recursos de gerenciamento e autenticação de usuários, gerenciamento e leitura das métricas definidas, relacionando-as com as respectivas categorias, e gerenciamento dos projetos de coleta de métricas. Ademais, realiza o versionamento dos projetos de métricas manipulando os artefatos de entrada de maneira a fornecê-los ao motor de métricas.

- O módulo ***MMTool*.Core** é implementado para realizar a leitura dos artefatos, computar os resultados das métricas e armazená-los. A coleta das métricas é realizada pela API *OpenCore*. Após isso, os resultados são armazenados em XML e persistidos na base de dados.

A seguir na Figura 4.6, apresenta-se a visão lógica da arquitetura da ferramenta representada pelos módulos descritos e suas respectivas dependências.

O módulo ***MMTool*.Web** é a aplicação web responsável pelo gerenciamento das informações dos usuários, das métricas e dos projetos. Essas informações são mantidas em uma base de dados pelo pacote de persistência. O módulo ***MMTool*.Core** é executado ao final da criação do projeto de avaliação.

A interação entre os módulos inicia-se por chamadas à classe `../MetricsEngine`, realizadas pelas classes do pacote `MMTool.Web/../controller/`. Nessa interação são passados os parâmetros da avaliação, tais como, os arquivos de métricas e os modelos (diagramas).

No desenvolvimento da ferramenta *MMTool* utilizou-se o padrão arquitetural *Model-View-Control* (MVC) (Oracle Corporation, 2002) . As três camadas do Padrão Arquitetural MVC visam separar e controlar a lógica da aplicação da apresentação das informações aos

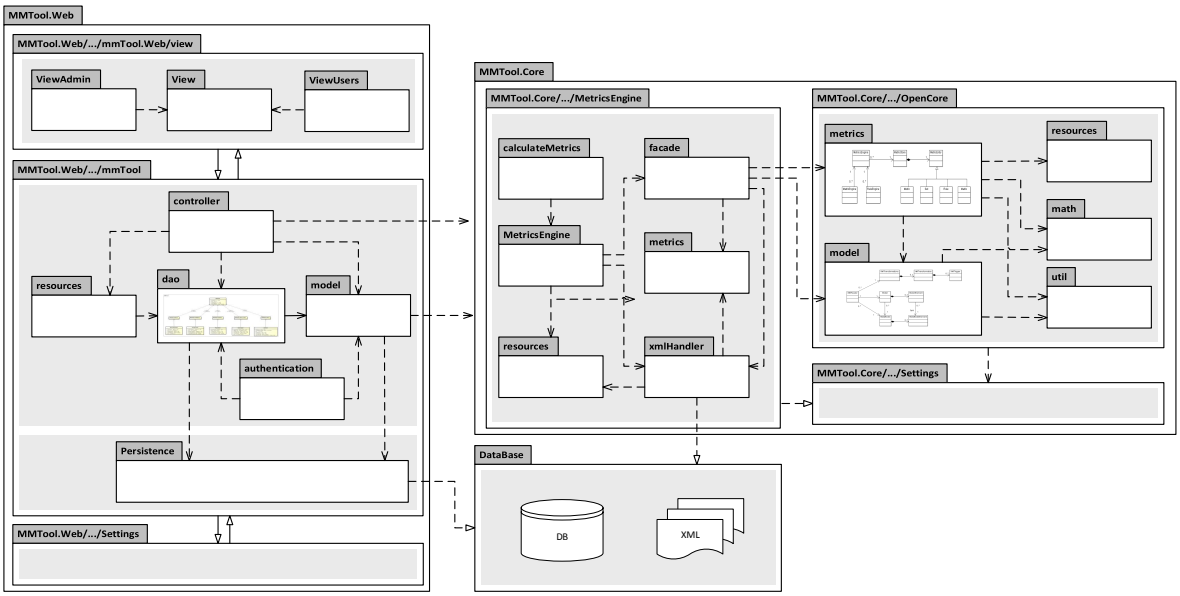


Figura 4.6: Arquitetura da Ferramenta *MMTool*.

usuários. No desenvolvimento, aplicou-se também alguns conceitos de padrão de projetos definido por Gamma et al. (1995), tais como *facade, singleton, abstract factory e template method*. Além disso, empregou-se o padrão de projetos estrutural *Data Access Object (DAO)*, no qual realiza-se a separação das regras de negócio da manipulação das regras de persistência de dados. A descrição interna de cada módulo é apresentada nas próximas seções.

Desenvolvimento do Módulo *MMTool*.Web

No desenvolvimento do módulo *MMTool*.Web foram utilizados alguns *frameworks* e tecnologias para auxiliar e prover facilidade na implementação. O desenvolvimento desse módulo é baseado em J2EE. Optou-se pela utilização das seguintes tecnologias e *framework*:

- Java EE 7 Web[22]: é uma plataforma para desenvolvimento de aplicações Java Web. Fornece apoio para a construção de software que utiliza *Enterprise JavaBeans, JavaServer Pages e Servlets*.

- JavaServer Faces – JSF 2.2[23]: é um *framework* para o desenvolvimento de aplicações Java Web especificado pela plataforma Java EE, portanto é multiplataforma.

[22] `https://www.oracle.com/java/index.html` – Acessado em Fevereiro de 2015.
[23] `https://javaserverfaces.java.net/` – Acessado em Fevereiro de 2015.

Baseia-se na programação de interfaces Web utilizando componentes predefinidos, viabiliza a reutilização e fornece suporte nativo a AJAX (LOBO FILHO, 2006).

- PrimeFaces 4.0[24]: é uma biblioteca de software livre de componentes para interfaces gráficas de usuário. Contém um conjunto de componentes para o desenvolvimento de aplicações Web baseadas em JSF.

- Apache Tomcat 8[25]: é um servidor de *container Web* software livre estável, pois contém as características exigidas pelas aplicações comerciais. Baseia-se em Java e executa aplicações desenvolvidas em Servlets e JSPs.

- Hibernate 4.2.6[26]: é um *framework* para o mapeamento Objeto-Relacional (do inglês, *ORM – Object/Relational Mapping*) para desenvolvimento Java.

- Connector MySQL Java 5.1[27]: é o *plugin* responsável pela conexão com o banco de dados MySQL.

Na Figura 4.7 apresenta-se a arquitetura do módulo ***MMTool*.Web**, o qual contém os pacotes da camada de Visão e as páginas web. O módulo de Visão transmite as informações para o pacote `MMTool.Web/../mmTool`, o qual contém os pacotes da camada de Controle e Modelo, assim como o pacote `dao`, que é responsável pelo acesso à camada de persistência de dados.

A seguir na Tabela 4.3, apresentam-se os dados referentes ao esforço para a implementação dos módulos ***MMTool*.Web**.

A análise é realizada com base em métricas de projetos extraídas diretamente do código fonte, tais como, número de classes, números de linhas e número de linhas comentadas. Nessa análise utilizou-se a ferramenta *Understand*[28]. Nota-se que os arquivos analisados são do tipo *Java* e as métricas foram extraídas das 86 classe do projeto, totalizando 14344 linhas entre comentadas, brancas e de código. Ressalta-se que as métricas referentes aos *framework* e APIs foram excluídas.

[24] `http://www.primefaces.org/downloads` – Acessado em Fevereiro de 2015.
[25] `http://tomcat.apache.org/` – Acessado em Fevereiro de 2015.
[26] `http://hibernate.org/orm/downloads/` – Acessado em Fevereiro de 2015.
[27] `http://www.mysql.com/downloads/` – Acessado em Fevereiro de 2015.
[28] `https://scitools.com/` – Acessado em Fevereiro de 2015.

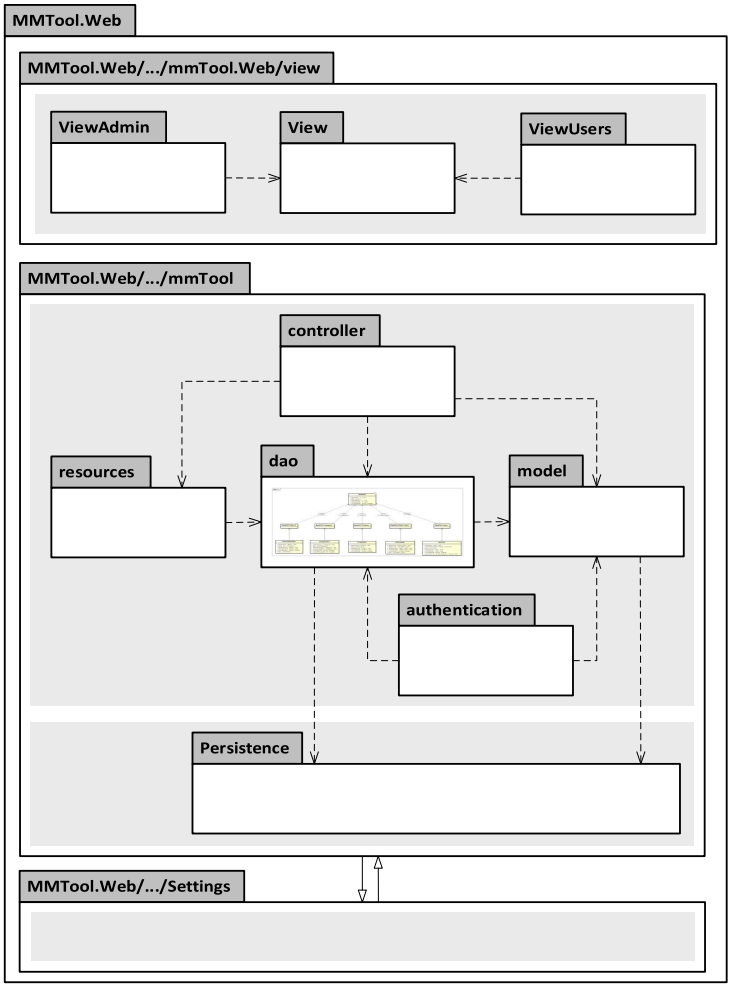


Figura 4.7: Arquitetura do Módulo *MMTool*.Web.

Tabela 4.3: Esforço para o Desenvolvimento do *MMTool*.Web.

***MMTool*.Web**	
Tipo de Arquivo: Java	
Métricas	
Nº de Classes	86
Nº de Arquivos	120
Nº de Funções/Métodos	872
Nº Total de Linhas (NL)	14344
Nº de Linhas Brancas (BLOC)	2095
Nº de Linhas de Código (LOC)	10001
Nº de linhas com Comentários (CLOC)	2248
Razão: CLOC / LOC	0,22

Desenvolvimento do Módulo *MMTool*.Core

Esse módulo é implementado para realizar a extração das métricas dos modelos. Ele foi desenvolvido com o auxílio de API's. Dentre as funcionalidades desse módulo está a

leitura dos arquivos XML, os quais contêm a definição das métricas. Para isso, utilizou-se o *Java Architecture for XML Binding* (JAXB)[29], que realiza a leitura e geração de XML, transforma objetos Java em elementos XML e faz o *parsing* de elementos XML em objetos Java.

A principal reutilização realizada nesse módulo é a integração da API *OpenCore*, que é responsável pela extração das métricas. Porém, para reutilizá-la, foi necessário construir uma camada utilizando o padrão de projeto *Facade* (GAMMA et al., 1995), pois a *OpenCore* fornece um grande conjunto de funcionalidades implementadas pelas suas classes. Dessa maneira, o intuito foi reduzir a complexidade, minimizando a comunicação e dependências entre a *OpenCore* e os demais módulos da ferramenta por meio de um objeto fachada que implementa uma interface única e simplificada.

Como os detalhes da *OpenCore* foram descritos na Seção 4.4.1, nesta seção concentra-se na apresentação dos pontos referente à integração da API à *MMTool*.

Conforme apresentado na Figura 4.8, o módulo divide-se em dois pacotes. O pacote `MMTool.Core/../OpenCore` refere-se à API *OpenCore*, enquanto o pacote `MMTool.Core/../MetricsEngine` contém a implementação das funcionalidades responsáveis pela preparação dos arquivos de entrada, deixando-os no formato compatível com o padrão de entrada da *OpenCore*. Além disso, esse pacote contém as classes responsáveis pela leitura dos elementos XML, dos elementos que especificam as métricas e dos elementos XMI dos diagramas.

O esforço de implementação do módulo ***MMTool*.Core** é caracterizado na Tabela 4.4.

Na Tabela 4.4, apresenta-se o esforço de implementação do motor de métricas. Observa-se que foram implementadas 13 classes em Java com 64 métodos. Essas classes são responsáveis pelas chamadas das funcionalidades da API *OpenCore* e manipulação dos arquivos *XML*. Conforme apresentado no decorrer desse trabalho, a API *OpenCore* implementa grande parte das funcionalidades de extração das métricas, isso pode ser notado pelo número de linhas de código (LOC), pois foram adicionadas apenas 503 linhas para a implementação do padrão *facade*.

[29] `https://jaxb.java.net/` – Acessado em Fevereiro de 2015.

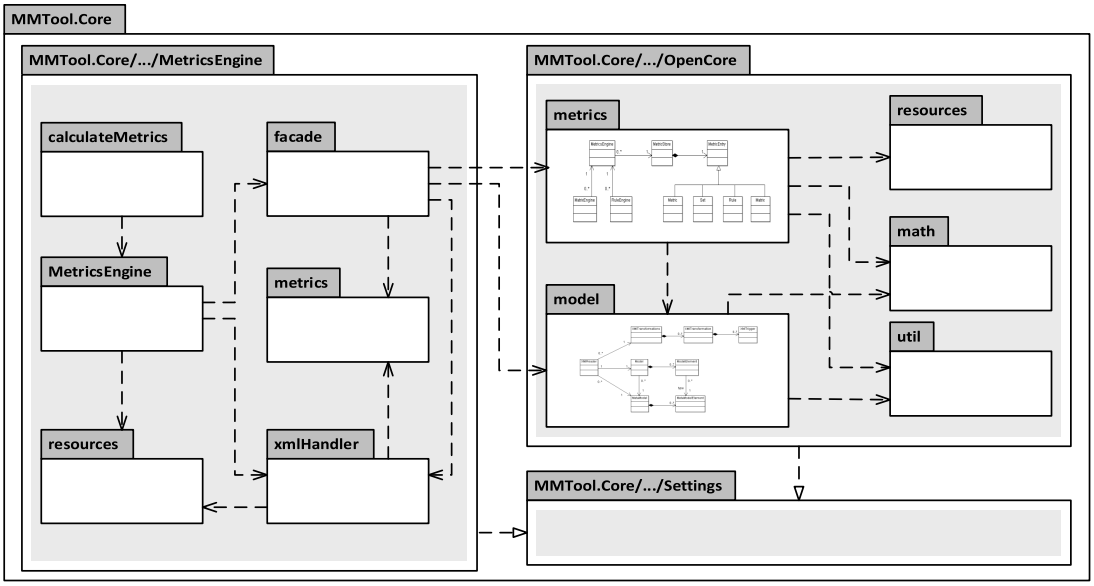


Figura 4.8: Arquitetura do Módulo *MMTool*.Core.

Tabela 4.4: Esforço para o Desenvolvimento do *MMTool*.Core.

***MMTool*.Core**	
Tipo de Arquivo: Java	
Métricas	
Nº de Classes	13
Nº de Arquivos	13
Nº de Funções/Métodos	64
Nº Total de Linhas (NL)	856
Nº de Linhas Brancas (BLOC)	136
Nº de Linhas de Código (LOC)	503
Nº de linhas com Comentários (CLOC)	217
Razão: CLOC / LOC	0,43

4.6 Análise do Ambiente Automatizado Desenvolvido

Na Figura 4.9, apresentam-se os mecanismos utilizados para cada um dos processos de avaliação do nível de qualidade dos modelos do QM^2 definido no Capítulo 3. A seguir são descritas as soluções automatizadas para cada um dos processos:

- As atividades do processo ***Definir*** são realizadas pela associação das funcionalidades fornecidas pela ferramenta *MMTool* e pela plataforma de avaliação *Spago4Q*, nas quais as métricas internas são definidas em arquivos XML e importadas para a ferramenta *MMTool*. Por outro lado, os modelos de qualidade e, consequentemente, os atributos dos modelos são especificados no *Spago4Q*. Além disso, a plataforma *Spago4Q* fornece suporte para o mapeamento das métricas internas com os atributos

de qualidade.

- As atividades do processo ***Extrair*** são realizadas pela ferramenta *MMTool*. A atividade de extração das métricas é realizada por meio da API *OpenCore* que está acoplada à ferramenta. Esse processo é finalizado com o armazenamento dos resultados em arquivos XML e no banco de dados. A *MMTool* também é responsável pelo versionamento dos projetos de avaliação.

- O processo ***Avaliar*** é automatizado pela plataforma *Spago4Q*. Esse processo consiste em realizar a recuperação dos dados, por *Queries* ou algoritmos Java, e processá-los por meio do mapeamento realizado no processo ***Definir***.

- O processo ***Sintetizar*** consiste na geração dos relatórios *KPI's*, que são elaborados na plataforma *Spago4Q*. Esses relatórios são a representação e monitoramento do nível de qualidade dos modelos. Os *KPI's* são responsáveis por agrupar as métricas e associá-las aos atributos de qualidade.

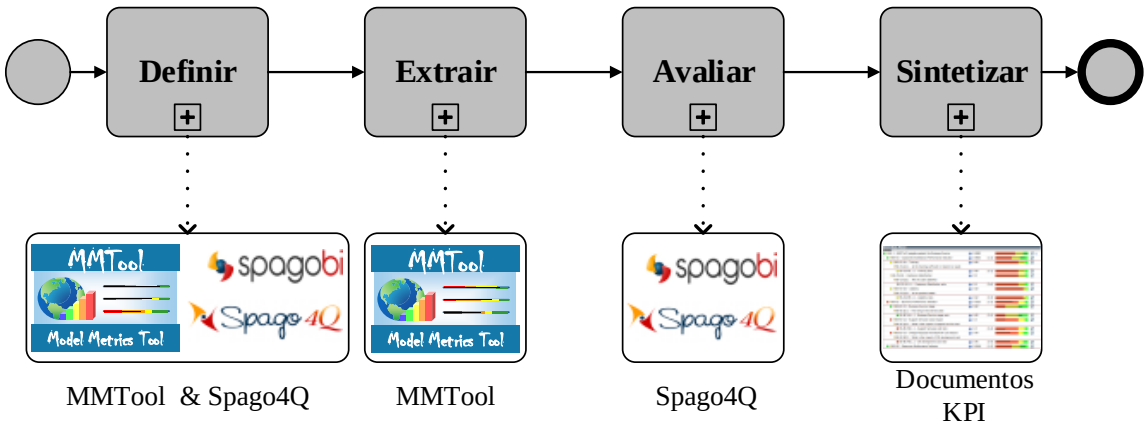


Figura 4.9: Automatização do QM^2.

Ressalta-se que a ferramenta *MMTool* fornece apoio à avaliação do nível de qualidade dos projetos, possibilitando a avaliação de diferentes domínios e linguagens de modelagem. Pode-se definir métricas e modelos de qualidade personalizados ao domínio pretendido. Os modelos de qualidade têm o intuito de proporcionar análises da qualidade completa. A *MMTool*, em conjunto com as ferramentas reutilizadas, fornece funcionalidades para a definição e coleta de métricas em modelos, definição de modelos de qualidade, associação dos atributos de modelos de qualidade com as métricas, monitoramento e interpretação dos resultados da avaliação dos modelos por meio de *dashboards* e relatórios.

4.6.1 Fluxo de Trabalho do Ambiente

Esta seção tem a finalidade de apresentar uma visão geral do fluxo de trabalho fornecido pelo ambiente de avaliação. O fluxo é apresentado na Figura 4.10. Esse fluxo consiste nos seguintes passos:

1. A ferramenta de modelagem é o ambiente no qual são elaborados os diagramas dos projetos. Essa ferramenta deve fornecer recursos para exportar os diagramas no formato XMI;

2. Realizar a exportação dos arquivos *XMI* e, em seguida, compactar os arquivos em formato ZIP. Esse arquivo pode conter vários diagramas (em *XMI*), representando vários tipos de diagramas;

3. Importar o projeto (arquivo ZIP) para a ferramenta *MMTool* (usuário autenticado). Em seguida, uma sequência de atividades é executada automaticamente, sendo elas:

   (a) Descompactação e cópia dos arquivos para o diretório do usuário;

   (b) Configuração automática dos parâmetros da extração;

   (c) Configuração dos diretórios dos arquivos de diagramas e das métricas;

   (d) Envio dos arquivos do projeto à *OpenCore*. Para tanto, realiza-se a extração das métricas[30]; e,

   (e) Executa-se a geração dos arquivos XML, contendo os resultados da computação das métricas e armazena-os em banco de dados.

4. Após obter os resultados das métricas e armazená-las no banco de dados, são executadas as funcionalidades sobre responsabilidades do *Spago4Q*:

   (a) Carregamento dos resultados das métricas para o *Spago4Q* por meio de sentenças SQL.

   (b) Obtenção do retorno das *queries*, cujos valores referem-se às métricas definidas;

   (c) Mapeamento entre os resultados das métricas e os atributos do modelo de qualidade definido no *Spago4Q*; e

   (d) Elaboração dos *dashboards* e dos relatórios *KPI*.

[30]Na figura, estão ocultos os processos internos dos mecanismos da *OpenCore*, os quais já foram descritos na Secão 4.4.1.

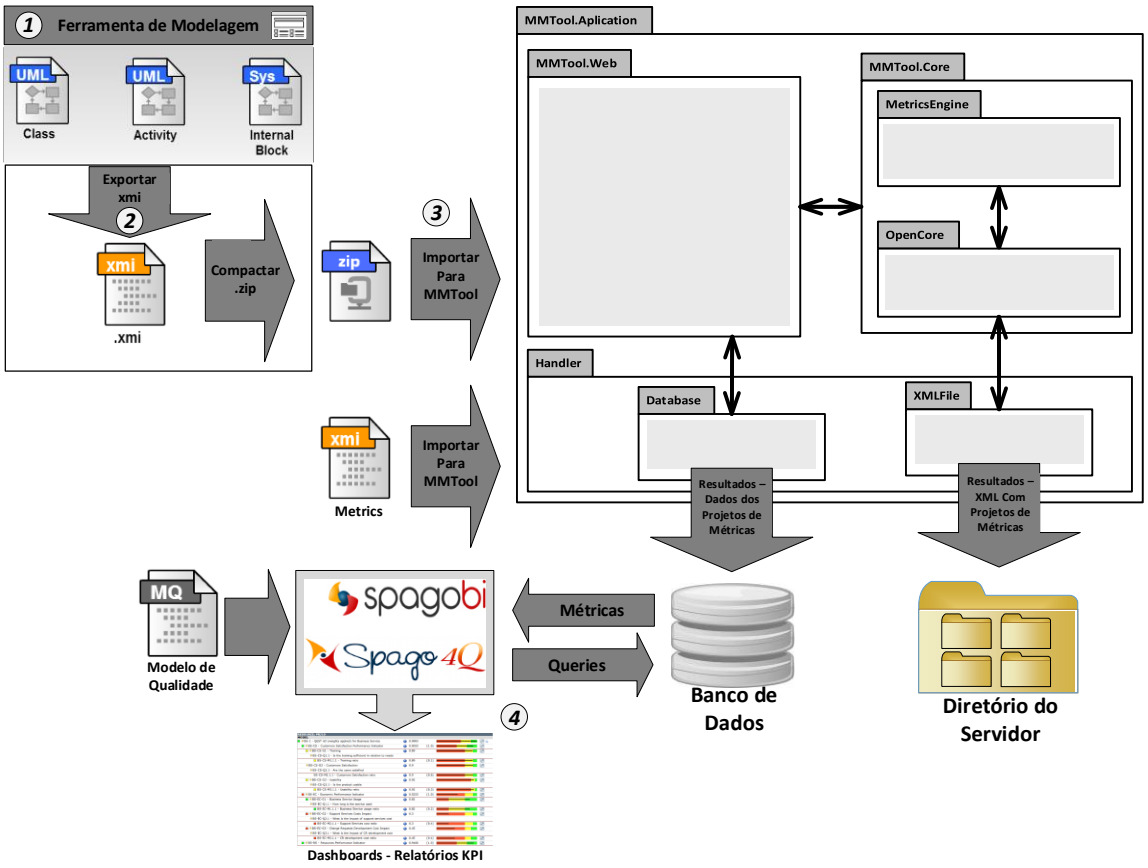


Figura 4.10: Fluxo do Ambiente.

A integração entre as ferramentas consiste em três módulos principais: (i) a ferramenta responsável pelo gerenciamento dos projetos de avaliação, gerenciamento das métricas definidas e do armazenamento dos resultados das métricas; (ii) o módulo que realiza a coleta das métricas, a qual é realizada por meio da API *OpenCore* integrada à ferramenta; e, (iii) a plataforma de avaliação, que realiza a recuperação dos resultados das métricas, correlaciona-os com os atributos dos modelos de qualidade específicos e gera os relatórios.

4.6.2 Restrições e Limitações do Ambiente

- Sabe-se que a API *OpenCore* utilizada no núcleo do motor de métricas apresentou problemas na extração de métricas dos diagramas de sequência e comunicação;
- A ferramenta *MMTool* depende da especificação das métricas, exclusivamente em formato XML, conforme uma gramática definida pelo padrão da API *OpenCore*;
- Por convenção, adotou-se a utilização de UML2.x (*XMI* 2.x), baseado nos quais os arquivos de regras de transformação e matamodelo são configurados internamente na ferramenta;

- Identificou-se uma dependência entre o *Spago4Q* e o *MMTool*, principalmente nas atividades de mapeamento e avaliação, devido à transferência dos dados gerados no *MMTool* ao *Spago4Q* para a execução do processo de síntese;
- A ferramenta está condicionada à execução em nível de protótipo, pois há a necessidade da elaboração de testes em todo o ambiente na fase de produção;
- No ambiente, os módulos sob responsabilidade do *Spago4Q* estão limitados à utilização das ferramentas apresentadas na Seção 4.4.2, assim como as suas respectivas versões e funcionamento;

Apesar das restrições e limitações no âmbito tecnológico, o ambiente atende os propósitos originais para automatização do QM^2, sendo eles: (i) coletar métricas de modelos; (ii) apoiar a definição de métricas e modelos de qualidade; (iii) realizar o mapeamento das métricas internas com atributos do modelo de qualidade; e (iv) elaborar e apresentar relatórios com o nível de qualidade dos modelos, de acordo com o modelo de qualidade definido.

4.7 Considerações Finais

Neste capítulo, foram destacadas as definições dos mecanismos para a automatização do QM^2. Além disso, fez-se uma descrição detalhada sobre os principais tipos de ferramentas reutilizadas no desenvolvimento do ambiente de avaliação. Foram apresentados estudos sobre o funcionamento dessas ferramentas, e os respectivos aspectos de desenvolvimento realizados para a integração das ferramentas no contexto desse trabalho.

Observou-se que a API *OpenCore* destaca-se no quesito de extração de métricas em modelos, devido às funcionalidades apresentadas para a recuperação das informações diretamente dos arquivos *XMI* que contêm os diagramas. Em relação à realização da avaliação e da síntese dos resultados, selecionou-se a plataforma de avaliação de qualidade *Spago4Q*. O *Spago4Q* apoia a definição do modelo de qualidade, realiza o mapeamento das métricas internas com os atributos de qualidade e fornece recursos para a elaboração de relatórios.

Conforme apresentado neste capítulo, para a integração desses dois módulos desenvolveu-se uma ferramenta Web, que é responsável por um conjunto de funcionalidades, tais como gerenciamento e versionamento dos projetos de avaliação.

As análises realizadas nesses capítulos visaram verificar o funcionamento da integração das ferramentas, com o intuito de avaliar a execução correta das funcionalidades propostas

no QM^2. No próximo capítulo apresentam-se os estudos de viabilidade conduzidos para avaliar o ambiente desenvolvido.

Capítulo 5

Aplicação Automatizada do QM^2

5.1 Considerações Iniciais

O foco deste capítulo é a descrição de um estudo de viabilidade da aplicação do *framework* QM^2 definido no Capítulo 3. No decorrer do capítulo, descreve-se também a aplicação dos recursos de automatização desenvolvidos e descritos no Capítulo 4. Também são apresentadas as definições necessárias para a avaliação do nível de qualidade de projetos de software com base no QM^2, incluindo a identificação dos modelos de qualidade, a seleção dos atributos de qualidade e a definição das métricas. O estudo foi realizado no contexto de Sistemas Embarcados, considerando modelos de qualidade e métricas para esse domínio de aplicação. A justificativa para a seleção do contexto de Sistemas Embarcados encontra-se na Seção 5.2.2.

Na Seção 5.2 e nas respectivas subseções, descrevem-se em detalhes a aplicação do QM^2. A Seção 5.3 aborda um estudo complementar realizado como avaliação piloto do QM^2 e do ambiente automatizado. Por fim, na Seção 5.4 apresenta-se uma discussão sobre o *framework* conceitual QM^2, aspectos do ambiente e os resultados obtidos nos estudos executados.

5.2 Aplicação do Estudo de Viabilidade

Nesta seção são descritos os processos do QM^2 executados durante sua aplicação em um estudo de viabilidade com suas respectivas automatizações, quando pertinentes. O objetivo do estudo e a metodologia adotada são apresentados a seguir.

5.2.1 Objetivo e Metodologia do Estudo de Viabilidade

O objetivo do estudo de viabilidade realizado é definido como:

> *Verificar a viabilidade da aplicação dos processos definidos pelo framework conceitual QM², em conjunto com os mecanismos que apoiam a automatização desses processos.*

Para atingir o objetivo proposto, inicialmente realizou-se investigações para identificar os artefatos solicitados pelo QM². Buscou-se por modelos de qualidade e métricas em estudos na literatura. Posteriormente, definiu-se o modelo de qualidade juntamente com o conjunto de métricas.

Em seguida, identificou-se os modelos de projetos de software que poderiam ser submetidos à avaliação pelo *framework* QM². Além disso, planejou-se a implementação das métricas internas e o mapeamento com os atributos externos do modelo de qualidade.

Por fim, todos os artefatos do projeto e arquivos de definição das métricas foram submetidos ao ambiente de avaliação. A avaliação do projeto em estudo foi realizada, obtendo-se um resultado referente ao nível de qualidade em relação a um atributo externo de qualidade. Ressalta-se que o ponto de partida foi a identificação do domínio de aplicação para o estudo, o qual é descrito na seção a seguir.

5.2.2 Definição do Domínio de Aplicação do Estudo de Viabilidade

A definição do domínio de aplicação é uma atividade conceitual que não gera artefatos para a aplicação do QM². Nessa atividade, os avaliadores estabelecem o escopo e as diretrizes para realizar a avaliação. Entre os domínios de aplicação cita-se como exemplo: Sistemas de Informação, Sistemas Adaptativos e Sistemas Embarcados.

Como um dos objetivos do QM² é ser genérico, sendo independente de domínio de aplicação, modelos de qualidade e métricas internas, pode-se selecionar diferentes domínios de aplicação. Cada domínio pode ter seus diagramas representados por diversas linguagens de modelagem, tais como a UML (empregada, por exemplo, para modelar sistemas de informação) e a SysML (utilizada, por exemplo, para modelar Sistemas Embarcados).

A definição do domínio de aplicação do estudo apresentado neste capítulo, tem como base os projetos de pesquisa conduzidos no âmbito do INCT-SEC (2013), no qual este

trabalho está inserido. No contexto do INCT-SEC são realizadas modelagens para projetos de veículos aéreos e terrestres não-tripulados como o Tiriba[1] e o CaRINA[2]. Além disso, as pesquisas também incluem o desenvolvimento de sistemas robóticos para monitoramento de áreas terrestres (PAULA, 2015) e manipulação de objetos (Kanguera Project, 2010).

Em particular, os modelos utilizados para a execução do estudo de viabilidade foram extraídos do trabalho de Paula (2015), cujo objetivo principal é a definição de uma arquitetura de referência para Sistemas Embarcados robóticos e o estudo baseou-se em diagramas instanciados dessa arquitetura. Os diagramas são modelados em UML e são construídos para o desenvolvimento de aplicações robóticas. Sobre a arquitetura de referência proposta pelo autor, são realizados estudos de casos com o desenvolvimento de sistemas para os robôs da Lego Mindstorms (2014).

Portanto, definiu-se Sistemas Embarcados como sendo o domínio de aplicação para a execução do estudo de viabilidade da aplicação do QM^2. Nas próximas seções são apresentados os processos de avaliação do nível de qualidade dos modelos, por meio de métricas mensuráveis. Além disso, associam-se os valores das métricas com atributos do modelo de qualidade e infere-se sobre o nível de compreensibilidade estrutural do projeto.

5.2.3 Processo de Definição – Seleção do Modelo de Qualidade

O primeiro passo do processo *Definir* do QM^2 refere-se à definição do *Modelo de Qualidade Inicial* (processo *Definir MQ* da Figura 3.2), no qual é selecionado o modelo de qualidade e os atributos de qualidade, tendo a opção de revisá-los se necessário. Ressalta-se que a definição dos modelos de qualidade está estritamente relacionada com o domínio de aplicação a ser avaliado.

Observa-se uma variedade de modelos visa apoiar o desenvolvimento de software no âmbito geral, especificamente para garantir a qualidade dos produtos de software. Porém, o conjunto de modelos de qualidade apresentado, não abrange totalmente os domínios de aplicação (DALLAL; BRIAND, 2010). Mesmo assim, esses modelos de qualidade tornam-se o ponto de partida para pesquisas que definem modelos de qualidade específicos (SHARMA et al., 2012). Para isso, realizam-se adaptações de acordo com as necessidades dos domínios e linguagens a serem aplicados na avaliação.

Como selecionou-se o domínio de Sistemas Embarcados para conduzir o estudo de viabilidade do QM^2 e do ambiente automatizado de avaliação proposto, essa atividade

[1] `http://www.inct-sec.org/br/aplicacoes/vant-tiriba` – Acessado em Fevereiro de 2015.
[2] `http://www.inct-sec.org/br/aplicacoes/carina` – Acessado em Fevereiro de 2015.

enfatiza a definição do modelo de qualidade para esse domínio, conforme descrito a seguir.

Seleção do Modelo de Qualidade para Sistemas Embarcados

Em relação aos Sistemas Embarcados, os modelos de qualidade podem ser considerados iniciativas importantes para garantir a qualidade do produto, pois eles têm características particulares, tais como, a utilização de hardware dedicado e restrição de tempo real. Esses fatores influenciam diretamente nos atributos essenciais para a avaliação da qualidade dos projetos desses sistemas.

Nesse sentido, realizou-se uma sequência de estudos para a identificação de um modelo de qualidade pertinente, com detalhes de especificação das etapas e artefatos, para que fosse possível a realização das definições estabelecidas pelo QM^2. Nestas circunstâncias, utilizou-se como base o trabalho apresentado por Oliveira et al. (2013), no qual realizou uma revisão sistemática da literatura sobre modelos e atributos de qualidade para Sistemas Embarcados. Nessa revisão sistemática foram identificados estudos que abordam a definição, a avaliação e aplicações de modelos de qualidade para Sistemas Embarcados. Ressalta-se que os estudos são para projetos de uma maneira geral, pois consideram todas as fases de desenvolvimento de Sistemas Embarcados.

Com a finalidade de selecionar um modelo de qualidade adequado, foi realizada uma análise dos três estudos identificados por Oliveira et al. (2013), todos relacionados a modelos de qualidade. São eles:

- *A Quality Model of Lightweight Component for Embedded System*

  Jeong e Kim (2011) adaptaram o modelo de qualidade de DeLone&McLean (DELONE et al., 2003), que é um modelo de qualidade para sistemas de informação, para o domínio de Sistemas Embarcados. Além disso, os autores apresentam um conjunto de atributos e critérios de qualidade para componentes de Sistemas Embarcados.

- *Practical S/W Component Quality Evaluation Model*

  Choi et al. (2008) realizam um cruzamento dos atributos de qualidade dos modelos – *Samsung software Component Quality evaluation Model (SCQM)* – com os modelos de qualidade ISO/IEC–9126 (2001) e a versão atualizada ISO/IEC–25010 (2005). Dessa análise extraiu-se oito características/atributos de qualidade e por meio de uma matriz de impacto definem a correlação entre atributos de qualidade e os fatores.

- *Objective Evaluation of Software Architectures in Driver Assistance Systems*

Ahrens et al. (2011) apresentam um estudo de métricas objetivas para avaliar a arquitetura de Sistemas Embarcados na indústria automobilística. Além disso, definem um modelo de qualidade, no qual os atributos estão organizados de forma hierárquica. Esse modelo de qualidade tem como base outros modelos de qualidade e os autores destacam os atributos de qualidade passíveis de medição, tais como manutenibilidade e modularidade.

Para o estudo de viabilidade descrito neste capítulo, optou-se por trabalhar com o modelo de qualidade elaborado por Ahrens et al. (2011) devido o nível de detalhes apresentados, pois define um modelo e correlaciona os atributos de qualidade com as métricas internas, ponderando-as de acordo com a importância para a composição do valor final do atributo.

Esse modelo de qualidade consiste em um conjunto de sete atributos de qualidade, cada um contendo vários subatributos, exceto o atributo Conformidade. Esses subatributos definidos são considerados atributos atômicos, podendo ser diretamente associados a métricas internas. Na Figura 5.1 apresentam-se os (sub)atributos do modelo de qualidade estabelecido por Ahrens et al. (2011).

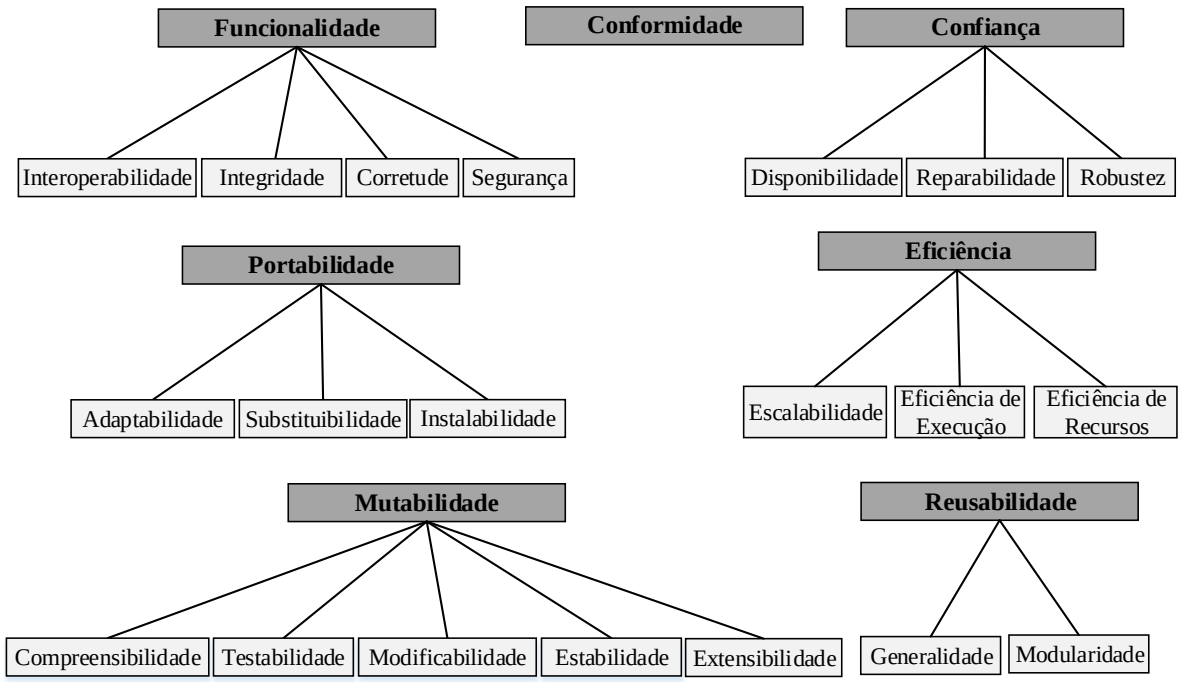


Figura 5.1: Modelo de Qualidade Definido por Ahrens et al. (2011).

Nesse modelo combinam-se e estendem-se vários subatributos. Ademais, o modelo especializa cada subatributo para as necessidades individuais da avaliação da arquitetura de software embarcado em automóveis. A descrição de cada (sub)atributo pode ser

encontrada no trabalho de Ahrens et al. (2011).

Na Tabela 5.1 apresenta-se uma correlação dos (sub)atributos do modelo de qualidade proposto por Ahrens et al. (2011) com os atributos dos modelos de âmbito geral. Ressalta-se que outros modelos específicos são encontrados na literatura, tais como, o modelo de qualidade para diagramas de classes de projetos OO elaborado por Bansiya e Davis (2002) e o modelo de qualidade para sistemas de informação proposto por Delone et al. (2003).

Tabela 5.1: Correlação dos Atributos dos Modelos de Qualidade.

Atributos	**Ahrens et al. (2011)**	**McCall (1977)**	**Boehm (1978)**	**ISO–9126 (2001)**	**ISO–25010 (2005)**
Funcionalidade	✓			✓	✓
Interoperabilidade	✓	✓		✓	✓
Integridade	✓	✓	✓		✓
Corretude	✓	✓			✓
Segurança	✓			✓	✓
Conformidade	✓				
Confiança	✓	✓	✓	✓	✓
Disponibilidade	✓				✓
Reparabilidade	✓		✓	✓	✓
Robustez	✓		✓		
Portabilidade	✓	✓	✓	✓	✓
Adaptabilidade	✓			✓	✓
Substituibilidade	✓			✓	✓
Instalabilidade	✓			✓	✓
Eficiência	✓	✓	✓	✓	✓
Escalabilidade	✓				
Eficiência de Execução	✓	✓			
Eficiência de Recursos	✓	✓		✓	
Mutabilidade	✓				
Compreensibilidade	✓		✓	✓	✓
Testabilidade	✓	✓	✓	✓	✓
Extensibilidade	✓	✓		✓	
Modificabilidade	✓		✓	✓	✓
Estabilidade	✓			✓	
Reusabilidade	✓	✓			✓
Generalidade	✓	✓			
Modularidade	✓	✓			✓

De acordo com a Tabela 5.1, nota-se que há uma intersecção parcial dos modelos de qualidade abordados. Salienta-se que foram considerados os atributos presentes nos níveis mais relevantes desses modelos. Por essa razão, alguns atributos não são listados.

As definições dos atributos de qualidade de Ahrens et al. (2011) são apresentadas na Seção 5.2.6, juntamente com seus respectivos mapeamentos com as métricas internas.

Seleção dos Atributos de Qualidade

Ao realizar a seleção do modelo de qualidade são identificados os atributos relevantes e passíveis de avaliação por meio de métricas, pois em alguns domínios de aplicação é inviável definir métricas para determinados atributos, sendo impossível de avaliá-los. Nesses casos, os modelos de qualidade devem ser adaptados, realizando alteração no conjunto de atributos. Por exemplo, ao executar avaliações de modelos na fase de projeto (*design*), alguns atributos de qualidade não são possíveis de serem avaliados por meio de medição diretamente em diagramas, pois relacionam-se a propriedades dinâmicas (execução) do software. Usabilidade e eficiência são dois exemplos que dependem da execução do software para serem mensurados. Por essa razão, a atividade de seleção e refinamento dos atributos de qualidade é relevante para o processo de avaliação.

No contexto de Sistemas Embarcados, Ahrens et al. (2011) estabeleceram um modelo de qualidade e, em seguida, a seleção dos atributos para executar uma avaliação em um domínio e contexto específico. Além disso, foram definidos atributos de compreensibilidade estrutural para os modelos de projetos e as métricas foram associadas aos atributos de qualidade por meio de fórmulas matemáticas ponderadas.

Nas discussões sobre os atributos do modelo de qualidade de Ahrens et al. (2011), identificaram-se alguns atributos inviáveis de serem mensuráveis por meio de métricas objetivas, são eles: substituibilidade, extensibilidade e generalidade.

A substituibilidade trata de substituição de módulos, componentes e subsistemas. De acordo com Ahrens et al. (2011), a substituibilidade de uma arquitetura de software não pode ser avaliada unicamente em termos absolutos (apenas sobre uma arquitetura), mas sim em relação a outra arquitetura, pois a capacidade de substituir uma arquitetura base está relacionada principalmente com a quantidade e tipos das interfaces e métodos.

Ahrens et al. (2011) destacam que os atributos extensibilidade e generalidade não são possíveis de serem valiados em determinados domínios e contexto específico ou fase do projeto por diferentes razões. Ao utilizar apenas informações estruturais, não se pode avaliar arquiteturas de software em relação a esses atributos, pois para avaliar extensibilidade deve-se conhecer as funcionalidades ou as extensões dos mecanismos concretos, para então determinar quão “fácil” ou “boa” é a extensão. A generalidade, por sua vez, é uma avali-

ação final de uma arquitetura de software e somente é possível avaliá-la com informações do contexto, considerando todos os componentes envolvidos e em funcionamento.

Por essas razões, Ahrens et al. (2011) definem métricas objetivas para os atributos apresentados na Figura 5.2, sendo eles: Mutabilidade, Eficiência e Reusabilidade.

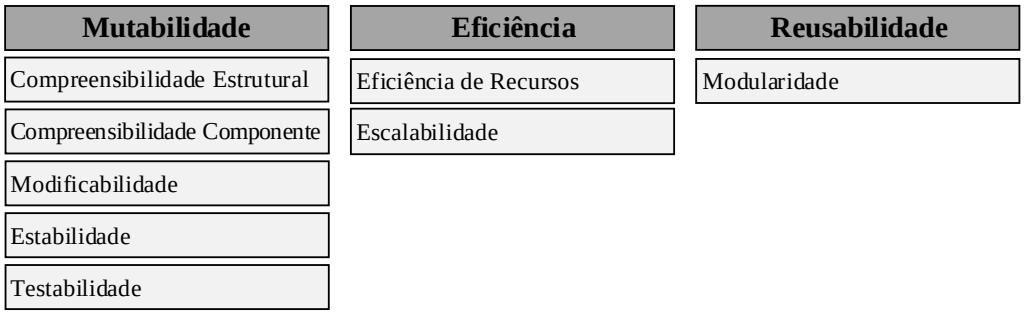


Figura 5.2: Atributos Selecionados para Definir Métricas (Ahrens et al. (2011)).

Na próxima seção apresenta-se dois exemplos de subatributos com a finalidade de compreensão dos tipos de elementos utilizados e as características para a definição das métricas internas. Além disso, são apresentadas as definições das métricas para o atributo compreensibilidade estrutural, que é o atributo selecionado para a execução do estudo de viabilidade.

5.2.4 Processo de Definição – Análise do Domínio

A segunda atividade do processo *Definir* do QM^2 é "*Analisar Domínio*" (representado na Figura 3.2). Essa atividade consiste em realizar a identificação do conjunto de elementos passíveis de serem mensuráveis por meio de métricas, considerando a linguagem de modelagem utilizada na representação dos projetos.

Nos casos em que os projetos são representados em UML, devem-se identificar os elementos que possam ser contados por métricas. Dessa maneira, essa análise auxiliará na compreensão das estruturas da linguagem, sobretudo na definição das métricas que baseiam-se nas informações obtidas dessa análise.

Ao final da análise obtém-se um *Modelo de Qualidade Intermediário*, o qual contém os atributos de qualidade mensuráveis e enumerados com as possíveis métricas. Com base nessa lista de atributos pode-se definir o conjunto de métricas para o domínio de aplicação em avaliação. A seguir apresentam-se exemplos de análises de linguagens empregadas em domínios específicos.

Resultados das Análises da Linguagem

Nesta seção, apresentam-se dois exemplos de análise de linguagem realizadas para a UML. Essa análise tem a finalidade de identificar os elementos para o cálculo dos atributos de qualidade identificados, e extrair informações para a implementação das métricas internas.

Na Tabela 5.2 apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de Compreensibilidade Estrutural (A1)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Tabela 5.2: Análise do Atributo de Compreensibilidade Estrutural.

Atributo de Compreensibilidade Estrutural – A1			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Componentes e Subcomponentes	Componentes	Componentes/Pacotes/Classes	Componente
	Pacotes	Componentes/Pacotes/Classes	Pacotes
	Classes	–	Classe

* Da granularidade alta para a refinada.

Os elementos quantificáveis para a avaliação desse atributo são definidos como componentes e subcomponentes, tais como pacotes, componentes e classes. Ou seja, as métricas são, em termos práticos, as contagens desses elementos. Consequentemente, deve-se extrair as métricas diretamente dos diagramas que contêm esses elementos.

Na Tabela 5.3 apresenta-se a análise para computar o *Subatributo de Modularidade (A7)*, que é diretamente associado ao atributo "*Reusabilidade*". Os resultados da análise mostram que para executar a extração e calcular as métricas precisa-se identificar os componentes, subcomponentes e as suas interfaces, tanto as de entrada quanto as de saída.

Tabela 5.3: Análise do Atributo de Modularidade.

Atributo de Modularidade – A7			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Componentes e Subcomponentes	Componentes	Componentes/Pacotes/Classes	Componente
	Pacotes	Componentes/Pacotes/Classes	Pacotes
	Classes	–	Classes
Interfaces de (Entrada/Saída)	Interfaces Providas	–	Sequência
	Interfaces Requeridas	–	Comunicação

* Da granularidade alta para a refinada.

Para calcular o atributo A7 as informações referentes a componentes e subcomponentes são extraídas de diagramas estruturais, seguindo a hierarquia entre os elementos. Identificou-se que as interfaces - (i) *requeridas* e (ii) *providas* - são contadas de acordo com (i) o número de operações que um componente chama e (ii) o número de chamadas as operações contidas em um determinado componente. As informações para as interfaces são extraídas de diagramas de sequência.

O procedimento de análise foi aplicado aos demais atributos de qualidade do modelo de Ahrens et al. (2011) e os resultados estão descritos no Apêndice B.

5.2.5 Processo de Definição – Definição das Métricas

A definição das métricas ("*Definir Métricas*" representado na Figura 3.2) é uma atividade que integra o processo de definição do QM^2. Nessa atividade, realiza-se a especificação do conjunto de métricas para cada atributo de qualidade. A definição ocorre após a análise da linguagem de modelagem, que identificou os elementos possíveis de serem utilizados na definição das métricas para a avaliação de modelos.

Esse processo tem por finalidade a elaboração de um conjunto de métricas. As métricas definidas e implementadas são utilizadas nas próximas atividades do processo do QM^2, principalmente nas atividades de extração e de mapeamento das métricas com os atributos de qualidade.

Definição das Métricas do Atributo de Compreensibilidade Estrutural

Nesta seção apresentam-se as definições de métricas que estão associadas ao subatributo de qualidade A1 (*Compreensibilidade Estrutural)* do modelo de qualidade de Ahrens et al. (2011) (Figura 5.2). Esse atributo é calculado de acordo com a Equação 5.1, o qual infere a facilidade de compreender os modelos e compõe a propriedade da capacidade de alteração dos elementos do projeto.

$$A1_i = \frac{CCF_i + \frac{\sum_{j=1}^{n} A1_j}{n}}{2} \tag{5.1}$$

Onde:

$1 \leq j \leq n$, i = Componente Corrente;

n = Quantidade de Subcomponentes do Componente i;

CCF_i = CCF do componente i;

$A1_j$ = Compreensibilidade Estrutural do Subcomponente j.

Na Tabela 5.4 apresentam-se os intervalos e os valores para o *Fator de Complexidades de Componente* (CCF, do inglês, *Component Complexity Factor*). O CCF leva em consideração as estruturas internas da arquitetura de software, pois a forma de organizá-la influência na compreensibilidade por parte dos envolvidos nos projetos. Frequentemente, utiliza-se a abstração de componentes ou módulos em seus níveis superiores.

Tabela 5.4: Fator de Complexidade de Componentes (AHRENS et al., 2011).

# Subcomponentes	CCF
0	0
1–4	2
5–9	0
10–11	3
12–14	5
>14	10

Interpretação dos Resultados do Atributo – A1

O atributo *Compreensibilidade Estrutural* é calculado de maneira recursiva, utilizando as métricas definidas sobre os elementos e subelementos; além disso, considera todos os níveis hierárquicos dos diagramas do projeto, tais como número de pacotes, componentes e classes. Nota-se que o valor de compreensibilidade estrutural do componente corrente é calculado e transferido para compor o cálculo do nível superior. Utiliza-se um conjunto de valores para os subcomponentes ($\sum_{j=1}^{n} A1_j$) para estabelecer o valor final para o componente em análise.

De acordo com Ahrens et al. (2011) o intervalo está entre 0 a 10, e quanto menor for o valor obtido, maior é a facilidade de compreensão estrutural de toda a arquitetura. Observa-se que a precisão dos valores obtidos depende do nível de granularidade que o projetista emprega nos modelos elaborados para o projeto. Além disso, destaca-se que esses intervalos são estabelecidos pelos autores no contexto de Sistemas Embarcados automotivos.

Definição das Métricas do Atributo de Modularidade

As métricas que estão associadas ao atributo de qualidade **A7 – Atributo de Modularidade** (Figura 5.2), baseiam-se no número de elementos estruturais e no tamanho

das mensagens trocadas entre os elementos do projeto. O atributo é calculado de acordo com a Equação 5.2 do modelo de qualidade proposto por Ahrens et al. (2011).

$$A7_i = \frac{\frac{\sum Mess_{ext,i}}{\sum Mess_{ext,i} + \sum Mess_{int,i}} + \frac{\sum_{j=1}^{n} A7_j}{n}}{2} \tag{5.2}$$

Onde:

$1 \leq j \leq n$, i = Componente Corrente;

n = Quantidade de Subcomponentes do Componente i;

$Mess_{ext,i}$ = Interfaces de saída do componente i para componente externo;

$Mess_{int,i}$ = Interfaces internas ente subcomponente do componente i;

$A7_j$ = Modularidade do Subcomponente j.

Baldwin e Clark. (2000) caracterizam boa modularidade como sendo forte coesão e fraco acoplamento. Isto é, os elementos estruturais com muitas dependências e várias interações (comunicações) devem ser agregados em níveis superiores. Além disso, os elementos superiores devem se comunicar o mínimo possível. Portanto, forte coesão e fraco acoplamento, na melhor das hipóteses, podem ser avaliados como sendo a medida dos relacionamentos entre os elementos internos e externos.

Interpretação dos Resultados do Atributo – A7

O atributo *Modularidade* considera, para qualquer elemento estrutural, a quantidade e o tamanho das mensagens trocadas no mesmo nível hierárquico, estabelecendo o somatório de todas as mensagens, tanto as internas quanto as externas. Essas mensagens são mecanismos de comunicação entre os módulos que são a menor unidade de distribuição do software.

O atributo é calculado primeiramente para um único componente e, em seguida, os valores dos componentes internos são levados em consideração com a ponderação de 50:50. O valor final é calculado recursivamente e o intervalo dos resultados situa-se entre 0 a 1, no qual valores próximos de 0 indicam uma ótima modularidade, pois não há comunicação externa. Por outro lado, quando os valores obtidos se aproximam de 1, evidencia-se que existe um alto número de trocas de mensagens entre os módulos do software (AHRENS et al., 2011).

Implementação do Conjunto de Métricas Internas

Para a implementação das métricas internas, com a finalidade de realizar o cálculo do valor final do atributo *Compreensibilidade Estrutural* – A1, leva-se em consideração a análise da linguagem realizada na Seção 5.2.4. A análise identifica os elementos possíveis de serem mensurados. Sendo assim, definiu-se métricas internas sobre os elementos apresentados na Tabela 5.2, de acordo com os componentes da Equação 5.1.

Na Tabela 5.5 apresentam-se as métricas internas implementadas em formato *XML*. Essas métricas foram implementadas para realizar a coleta dos valores internos dos diagramas, as quais necessitam obter os valores referentes aos pacotes, componentes e classes, juntamente com o valor interno de cada um dos elementos do modelo. Além dessas métricas, outras foram definidas para auxiliar na computação desse conjunto de métricas.

Tabela 5.5: Métricas Internas para o Cálculo do Atributo A1.

Categoria	Domínio	Nome	Descrição
Tamanho	Pacote	***NumPack***	Número de pacotes no pacote
		NumComp	Número de componente no pacote
		NumCls	Número de classes no pacote
		SumPack	Total de elementos do domínio no pacote
		CCF_Pack_i	Fator de Complexidade de Componente*
Tamanho	Componentes	***NumPack***	Número de pacote do componente
		NumComp	Número de componente do componente
		NumCls	Número de classes do componente
		SumPack	Total de elementos do domínio do componente
		CCF_Comp_i	Fator de Complexidade do Componente*

* Apresentado na Tabela 5.4.

Em seguida é executado o cálculo parcial do atributo $A1_i$ para cada elemento. O complemento do cálculo é realizado recursivamente para todos os elementos dos projetos, sendo que esse cálculo é realizado de forma independente das definições das métricas e do motor de métricas da *OpenCore*. O conjunto de métricas implementado para o cálculo do CCF de cada elemento é apresentado na Tabela 5.4.

No Código 5.1 são mostradas as métricas internas implementadas para o domínio de pacotes em formato *XML*, de acordo com a gramática para a definição de métricas da *OpenCore*. As métricas de pacotes são definidas similarmente às métricas de componentes. A alteração é pontual na propriedade de domínio, trocando-se de `package` para `component`.

```
<!-- Métricas de Pacote para o Atributo A1 -->

<metric name="NumPack" domain="package" category="Size">
  <description>
    The number of sub-packages of the package.
  </description>
  <projection relation="context" target="package"/>
</metric>

<metric name="NumComp" domain="package" category="Size">
  <description>
    The number of component of the package.
  </description>
  <projection relation="context" target="component"/>
</metric>

<metric name="NumCls" domain="package" category="Size">
  <description>
    The number of classes in the package.
  </description>
  <projection relation="context" target="class" />
</metric>

<metric name="SumPack" domain="package" category="Size">
  <description>
    Sum of the number of elements (packages, components and classes) in the package.
  </description>
  <compoundmetric term="NumComp+NumPack+NumCls" fallback="0"/>
</metric>

  <!-- Métricas para Cálculo do CCF de pacotes -->

<metric name="CCF_Pack_Total" domain="package" category="Size">
  <description>
    Sum of the package CCFi.
  </description>
  <compoundmetric term="CCF_Pack_1+CCF_Pack_2+CCF_Pack_3+CCF_Pack_4+CCF_Pack_5+
    CCF_Pack_6" fallback="0"/>
</metric>

<metric name="M1" domain="package" category="Size">
  <description>
    Calculation of structural comprehensibility equation.
  </description>
  <compoundmetric term="(CCF_Pack_Total+(M1/SumPack))/2" fallback="0" recurse="true"
     scope="lower"/>
</metric>
```

Código 5.1: Métricas Internas de Pacotes em *XML*.

A implementação da métrica `SumPack`, por exemplo, executa a computação do somatório dos elementos (pacotes, componentes e classes) do pacote corrente. O domínio é indicado pelo atributo `domain`="`package`". O somatório do resultado das métricas `NumComp + NumPack + NumCls` é realizado pela *tag* `compoundmetric`.

5.2.6 Processo de Definição – Mapeamento

O processo de definição do QM^2 é concluído com a execução da atividade de mapeamento das métricas internas com os atributos de qualidade (*Mapear Métricas–AQ* da Figura 3.2). A atividade de mapeamento considera o conjunto de métricas definidas e

o Modelo de Qualidade Intermediário. Essa atividade resulta no *Modelo de Qualidade Final*, o qual contém as métricas implementadas, os atributos de qualidade selecionados e o mapeamento ponderado das métricas internas com os atributos de qualidade.

O mapeamento ponderado das métricas internas com atributos de qualidade é realizado durante a computação das métricas e a associação com os atributos de qualidade. No caso desse estudo de viabilidade, a métrica interna M1 apresentada no código 5.1 é associada com o subatributo Compreensibilidade Estrutural – A1.

No modelo de AHRENS et al., há atributos e subatributos de qualidade. O QM^2 permite a ponderação entre esses elementos. Entretanto, conforme definido no Capítulo 3, as ponderações também podem ocorrer no nível de métricas internas e nos atributos de qualidade aos quais elas se relacionam. Nesse caso, o modelo de qualidade instanciado no *Spago4Q* deve ter granularidade compatível com essa intenção. Por exemplo, métricas de coesão e acoplamento poderiam ter *KPI's* associados a elas, e ambas poderiam contribuir para a obtenção do nível de qualidade com relação a um determinado atributo, cada uma com sua ponderação.

Na Tabela 5.6 apresenta-se, como exemplo, os atributos ponderados no modelo de Ahrens et al. (2011), que são ponderados de acordo com a experiência dos projetistas. Essa ponderação inicial reflete a opinião dos especialistas sobre a importância da qualidade dos atributos, para o domínio específico de software embarcado automotivo.

Tabela 5.6: Pesos dos Atributos definidos por Ahrens et al. (2011).

Características	Ponderação	Atributos	Ponderação
Mutabilidade	2	Compreensibilidade Estrutural	1
		Compreensibilidade de Componentes	1
		Modificabilidade	3
		Estabilidade	1
		Testabilidade	2
Eficiência	1	Escalabilidade	2
		Eficiência de Recurso	1
Reusabilidade	1	Modularidade	1

No estudo de viabilidade, devido à particularidade da definição das métricas e dos atributos de qualidade, os processos "*Definir Métricas*" e o "*Mapear Métricas–AQ*" (Figura 3.2) são realizados simultaneamente, pois na definição das métricas já se executa o agrupamento das mesmas, atendendo às especificações da equação. A ponderação é realizada pelo processo *Mapear Métricas–AQ*, sendo que esses dois processos poderiam ser representados como processos paralelos no QM^2.

Implementação do Mapeamento

A implementação do mapeamento é composta por duas partes. Inicialmente, a implementação ocorre durante a computação das métricas internas CCF_i, a qual executa a associação das métricas de quantificação dos elementos. O complemento da implementação do mapeamento é realizado com o relacionamento das métricas internas relevantes com os atributos do modelo de qualidade definido. A conclusão do mapeamento é descrita a seguir.

Na automatização do QM^2, o mapeamento é realizado no *Spago4Q*, durante a definição do conjunto de atributos do modelo de qualidade por meio dos *KPI's*. Ao definir os *KPI's*, pode-se atribuir pesos para cada um deles, no qual cada um representa uma métrica interna ou diretamente um atributo de qualidade. Os pesos indicam a importância da métrica, ou do atributo, para a composição de cada atributo, ou de uma categoria de atributos de qualidade.

Devido às particularidades das métricas e do modelo de qualidade selecionado para executar o estudo de viabilidade, houve a necessidade da implementação de um algoritmo (escrito em linguagem Java) para o agrupamento dos valores. Isso foi necessário pois o subatributo *Compreensibilidade Estrutural* (A1) é calculado de maneira hierárquica, porém a *OpenCore* não apresenta suporte para essa atividade, uma vez que não interage sobre a hierarquia dos modelos. Salienta-se que esse processamento auxiliar não é requisito para todos os tipos de avaliação.

Na Figura 5.3, apresenta-se a tela de definição dos *KPI's* e seus respectivos pesos. Na definição dos *KPI's*, faz-se a associação com os *Datasets e Thresholds*. Esses dois mecanismos são previamente configurados, sendo que o *Dataset* é responsável por carregar as informações, enquanto os *Thresholds* representam as faixas de valores definidos, que são os níveis de qualidade para um determinado atributo de qualidade.

Ao final da execução do *Processo de Definição*, conclui-se a elaboração do *Modelo de Qualidade Final*. Na automatização do QM^2, esse processo é completamente realizado com a definição do modelo de qualidade no *Spago4Q*. Essa definição é apresentada por completo na Seção 5.2.8.

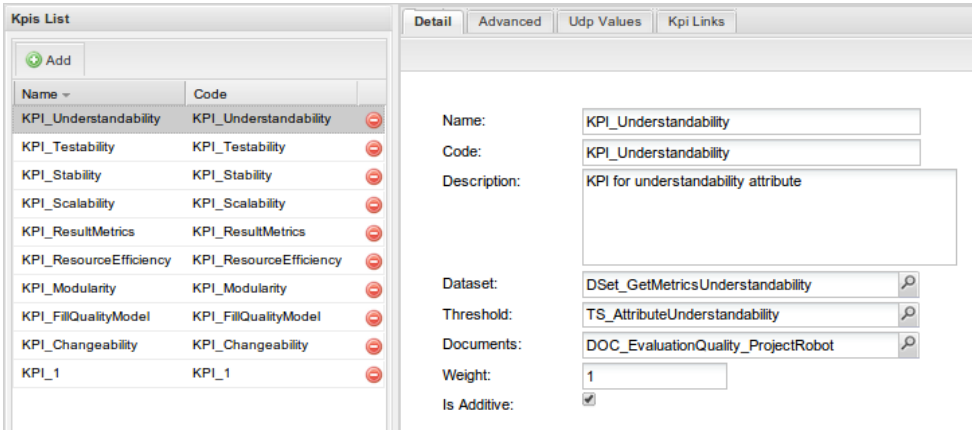


Figura 5.3: Definição dos KPI's.

5.2.7 Processo de Extração de Métricas

O processo de extração de métricas, representado por *Extrair* na Figura 3.1, é realizado pelas atividades *Extrair* e *Armazenar* do QM^2. A automatização do processo de extração é fundamental para todo o processo de avaliação do QM^2, pois em grandes projetos essas atividades são inviáveis de serem executadas manualmente.

A extração das métricas é realizada pelo motor de métricas, implementado na ferramenta *MMTool* com auxílio da API *OpenCore*. A seguir são descritos os resultados obtidos nesse processo, considerando as configurações prévias apresentadas no Capítulo 4.

Extração Automática de Métricas

Conforme estabelecido pelo QM^2, para executar a extração das métricas é necessária a definição de um conjunto de métricas e os modelos (diagramas) do projeto a ser avaliado. As métricas são implementadas em *XML* e importadas para a ferramenta *MMTool*.

Importação das Métricas no Ambiente de Avaliação

Ao importar as métricas em formato *XML*, a *MMTool* realiza o *parsing* dos elementos *XML* e insere as métricas na base de dados. Acoplado a essa funcionalidade está o gerenciamento das versões dos arquivos de definição de métricas. Esse requisito surgiu da necessidade de acompanhar a evolução da definição das métricas. Portanto, todos os arquivos com as definições de métricas submetidos são gravados e mantidos na pasta do usuário localizada no servidor.

Na Figura 5.4(a), apresenta-se a tela de importação das métricas. Nessa operação, seleciona-se a categoria que o conjunto de métricas foi definido.

Na Figura 5.4(b) são listadas as métricas importadas, que se referem às métricas implementadas para o cálculo do ***Atributo de Compreensibilidade Estrutural – A1***. Nota-se que as métricas auxiliares implementadas também são relacionadas nesse conjunto.

(a) Importação das Métricas para a *MMTool*.

(b) Lista de Métricas Importadas para *MMTool*.

Figura 5.4: Importação das Métricas para Ferramenta *MMTool*.

Importação dos Projetos no Ambiente de Avaliação

Na Figura 5.5, apresenta-se um modelo do projeto de sistema robótico instanciado com base na arquitetura de referência, ambos elaborados por (PAULA, 2015). Ressalta-se que os modelos do projeto são elaborados em ferramentas de modelagem com suporte para exportar para *XMI*. Esse arquivo é compactado, tornando-se um projeto de entrada para a ferramenta *MMTool*.

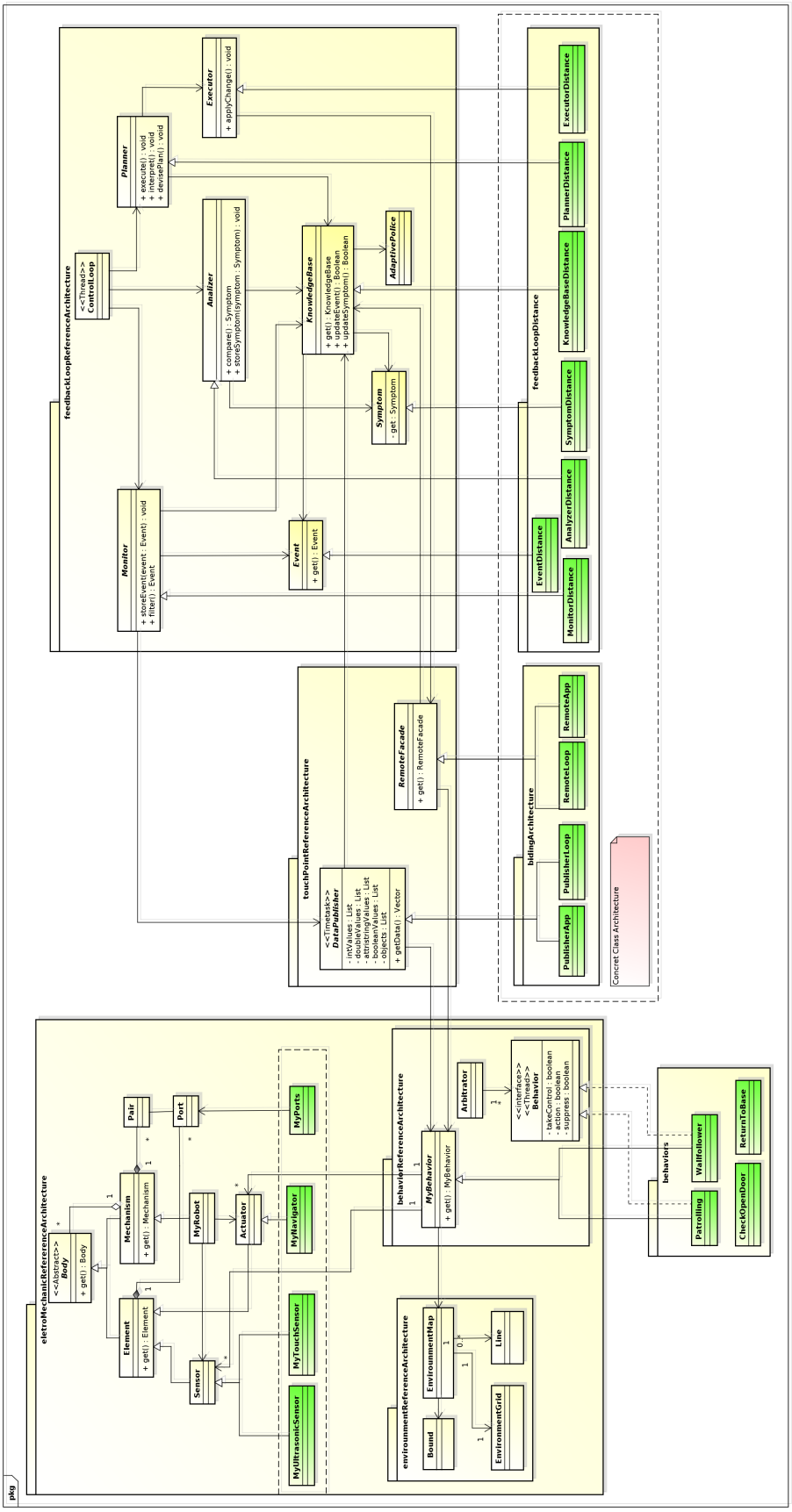


Figura 5.5: Modelo UML do Projeto de Arquitetura do Robô.

Ao criar o projeto, os arquivos *XMI* são descompactados e armazenados na pasta do usuário. Para realizar o estudo de viabilidade, os arquivos passam por um pré-processamento, que é executado com o intuito de identificar falhas na construção dos arquivos *XMI*, validar o arquivo de entrada e, principalmente, para a inclusão do elemento responsável por manter o valor global do nível de qualidade do projeto. Ressalta-se que esse processamento é exigido devido à particularidade do atributo de qualidade selecionado para o estudo de viabilidade.

Na Figura 5.6(a) apresenta-se a tela de criação de projetos. Observa-se que ao registrar o projeto na ferramenta *MMTool* são selecionadas algumas configurações, tal como a opção pelo tipo de métrica (métricas padrão ou métricas definidas pelo próprio usuário). Essa configuração indica qual é o arquivo de métricas utilizado na execução da extração das métricas pelo motor de métricas. A outra configuração a ser realizada é a seleção da categoria do projeto. Essa configuração é passada como parâmetro para o motor de métricas por meio de arquivos de configuração interna da ferramenta. Ao ser invocado o motor de métricas, esse arquivo contendo a implementação das métricas é carregado.

Na Figura 5.6(b) são listados os projetos pertencentes ao usuário e a funcionalidade de acesso ao *SpagoBI/Spago4Q*, que é utilizada para a geração de relatórios e documentos *KPI's*. Nota-se que o mesmo projeto pode ser submetido várias vezes, até mesmo com o mesmo nome, pois a ferramenta *MMTool* mantém o versionamento dos projetos por meio dos *timestamp* associados a cada projeto. Isso permite acompanhar a evolução das avaliações e a realização de comparações entre versões do mesmo projeto[3].

Mediante a importação das métricas e do projeto para a ferramenta *MMTool*, pode-se executar o processo de extração das métricas. As demais configurações, tais como a configuração do metamodelo e regras de transformação[4], são realizadas internamente. O processo de extração de métricas é executado pela funcionalidade "`Calculate`" presente na Figura 5.6(a).

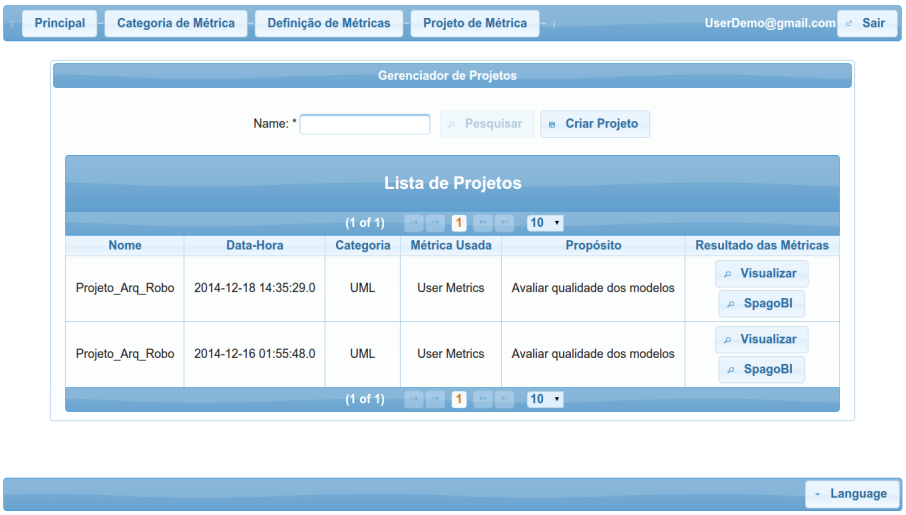
Armazenamento dos Resultados das Métricas

Seguindo o processo de extração de métricas, a última atividade a ser executada é o armazenamento dos resultados, correspondendo assim à atividade *Armazenar* da Figura 3.6 do QM[2]. A estratégia de armazenamento dos resultados é destacada pois tem impacto

[3] A versão corrente da *MMTool* não inclui funcionalidades de visualização do histórico de avaliações de projetos.

[4] A ferramenta *MMTool*, por convenção, está configurada para fornecer apoio à avaliação de projetos UML 2.x e *XMI* 2.x.

(a) Registro e *Upload* do Projeto.



(b) Lista de Projetos de Avaliação do Usuário.

Figura 5.6: Criação de Projetos de Métricas na Ferramenta *MMTool*.

nos processos de avaliação e síntese. O armazenamento das métricas é mantido de duas formas, levando em consideração a maneira em que os resultados podem ser processados para a formulação dos valores dos atributos.

Na primeira abordagem realizou-se o armazenamento em arquivos *XML*. A construção dos arquivos foi realizada com o auxílio da API *Java Architecture for* XML *Binding* (JAXB)[5]. Esse procedimento é realizado logo após a extração das métricas, no qual os resultados são fornecidos pelo motor de métricas. Os resultados podem ser carregados pelo *SpagoBI*/*Spago4Q* em *XML*, porém para isso deve-se implementar essa funcionalidade para realizar a extração das informações dos arquivos *XML*.

[5] `https://jaxb.java.net/` – Acessado em Fevereiro de 2015.

A segunda abordagem utilizada é o armazenamento direto dos resultados em uma base de dados. Essa opção é implementada por classes Java, que são mapeadas e persistidas utilizando-se o *framework Hibernate*. Essa abordagem foi utilizada no desenvolvimento da ferramenta *MMTool*, pois pode auxiliar na implementação de relatórios.

Outro fator de apoio à essa decisão de projeto é a maneira de acesso das plataformas *SpagoBI/Spago4Q*. Tinha-se em vista que os resultados são carregados por meio da configuração de *Datasources* e *Datasets*, sendo esses os procedimentos adotados nesse trabalho para a integração da ferramenta *MMTool* com a plataforma *SpagoBI/Spago4Q*.

5.2.8 Processo de Avaliação dos Projetos

O processo de avaliação dos projetos do QM^2, representado por *Avaliar* na Figura 3.1, é realizado por meio dos mecanismos fornecidos pela plataforma *SpagoBI/Spago4Q*. A avaliação consiste em definir o *modelo de qualidade* na plataforma, carregar os resultados calculados (métricas ou atributos) e fornecer os níveis de qualidade para os documentos *KPI's*.

No primeiro passo desse processo são configurados os *Datasources* e *Datasets*. Em seguida, realiza-se a configuração dos *Thresholds*, que são os intervalos de valores definidos para a inferência do nível de qualidade, nos quais os resultados das métricas são comparados e associados.

Os *Thresholds* podem ser personalizados de acordo com as necessidades do projeto em avaliação, ou com base na experiência dos avaliadores. Para a avaliação do estudo de viabilidade, considerou-se que o resultado para o atributo A1 deve estar entre 0 e 10, conforme destacado em (AHRENS et al., 2011). Entretanto, subdividiu-se esse intervalo em quatro partes iguais, apenas para simular níveis de qualidade. A definição dos *KPI's* é o mapeamento dos resultados com os atributos do modelo de qualidade. Ressalta-se que as configurações dos *KPI's* são apresentadas na Seção 5.2.6.

Na Figura 5.7 apresenta-se a definição do modelo de qualidade completo proposto por Ahrens et al. (2011). Nota-se que os procedimentos para a definição são, em termos práticos, a associação dos *KPI's* aos atributos definidos no modelo. Sob o pretexto de exemplo, são apresentados todos os atributos do modelo de qualidade proposto por Ahrens et al. (2011). Porém, o estudo de viabilidade aqui apresentado inclui somente a avaliação do atributo A1 - Compreensibilidade Estrutural, representado na Figura 5.7 por *Understandability*.

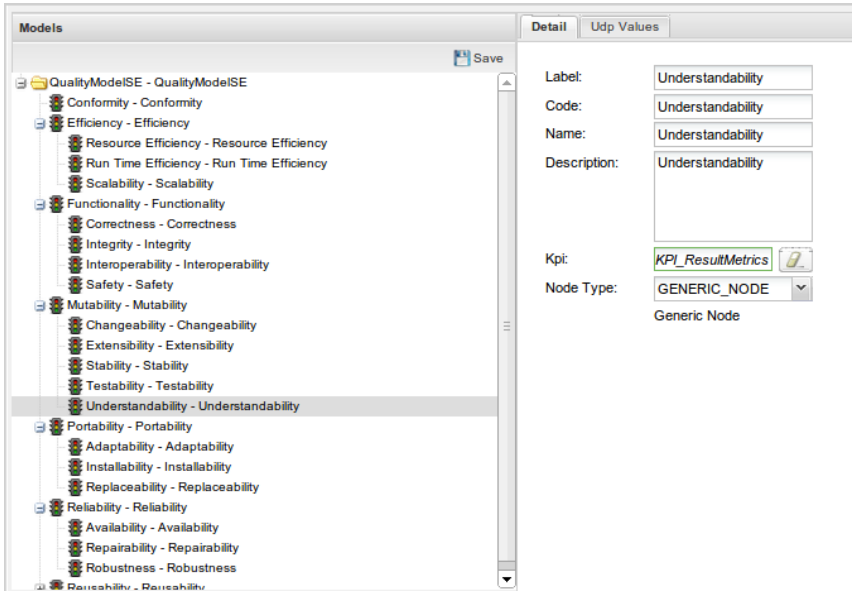


Figura 5.7: Definição do Modelo de Qualidade no *SpagoBI/Spago4Q*.

A avaliação é executada por meio da associação dos resultados aos intervalos definidos pelos *Thresholds* e mapeados para os atributos pelos KPIs. Esse conjunto de associações é realizado para obter o nível de qualidade que o respectivo atributo atinge na avaliação. Os resultados são sintetizados em documentos *KPI's* e apresentados na próxima seção.

5.2.9 Processo de Síntese dos Resultados

O processo de síntese é o último passo do QM[2], sendo representado pela atividade *Sintetizar* da Figura 3.1. A síntese é realizada por meio da elaboração de documentos *KPI's* fornecidos pelo *SpagoBI/Spago4Q*. Os detalhes dessas configurações encontram-se na documentação da plataforma[6].

Na Figura 5.8 apresenta-se o documento *KPI* elaborado com base no modelo de qualidade de Ahrens et al. (2011). Esse tipo de documento permite estabelecer parâmetros, que nesse caso é a seleção do projeto para o qual será executada a síntese.

O documento *KPI* apresenta os resultados obtidos para cada um dos atributos de qualidade. Na Figura 5.8, destaca-se o subatributo *Compreensibilidade Estrutural*, para o qual se obteve o valor de ***0,3459***. Ressalta-se que o valor em si não é importante para o contexto deste trabalho, ou seja, não se pode concluir que o modelo avaliado é de fácil ou difícil compreensão.

As faixas de valores associadas à facilidade ou dificuldade de compreensão estrutu-

[6] `http://www.spagoworld.org/xwiki/bin/view/Spago4Q/` – Acessado Fevereiro de 2015.

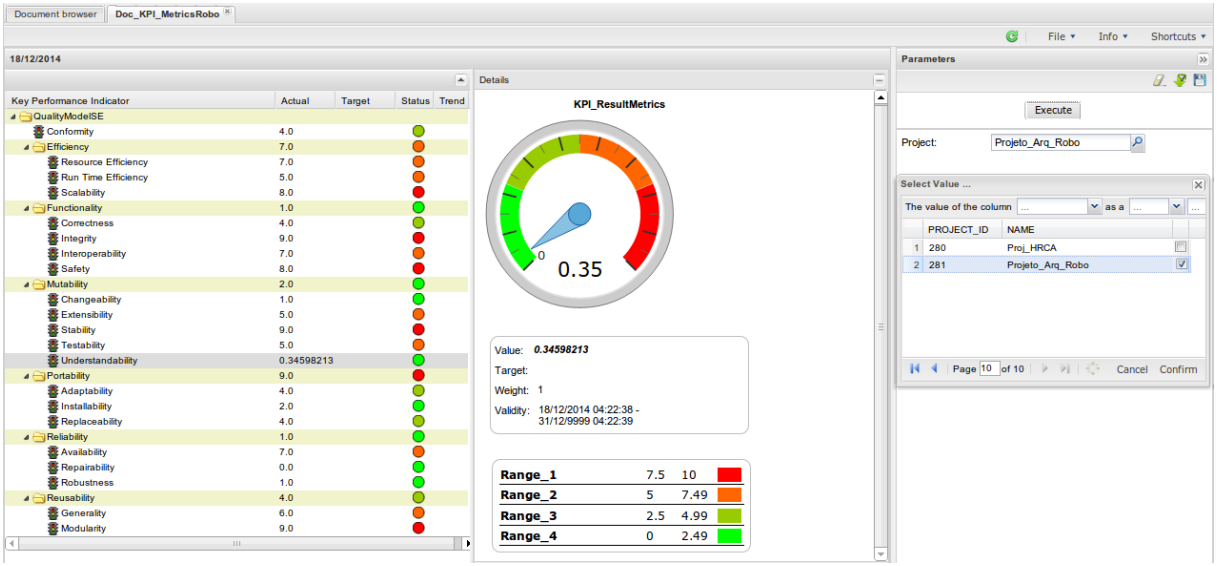


Figura 5.8: Síntese dos Resultados no *SpagoBI/Spago4Q*.

ral no estudo de viabilidade, foram definidas somente para efeito ilustrativo em quatro intervalos variando de 0 (zero) a 10 (dez) no total. Saliente-se, ainda, que os valores gerados para os demais subatributos do modelo ilustrado na Figura 5.8 foram gerados randomicamente.

Comparação Relativa entre Projeto e Arquitetura de Referência

Na Tabela 5.7 apresenta-se uma comparação relativa entre os resultados obtidos na avaliação da arquitetura de referência para Sistemas Embarcados robóticos, proposta por Paula (2015), com o resultado da avaliação da instância do projeto (criada pelo mesmo autor) abordado no estudo de viabilidade. Essa comparação relativa entre versões de projetos é um exemplo de estudos que podem ser realizados com o apoio da automatização do *framework* QM^2.

Tabela 5.7: Comparação relativa da Arquitetura.

Atributo Compreensibilidade Estrutural – A1		
Projeto	Intervalo	Valor
ReferenceArchitecture	0–10	0,255
ProjetoRobo	0–10	0,345
Diferença em (%)	0–100	+35,29

Executou-se o processo completo de avaliação para o projeto da arquitetura de referência, contabilizando o atributo de compreensibilidade estrutural, e obteve-se o resultado de *0,255*. Nota-se que o valor do projeto tem um aumento de *35,29%*, quando comparado

com o diagrama da arquitetura de referência. Esse aumento justifica-se devido ao número de elementos contidos no projeto (classes e pacotes), pois o projeto contém pacotes com as classes concretas referente a instância de um projeto sobre a arquitetura, sendo que essas classes estão destacadas na Figura 5.5 apresentada na Seção 5.2.7.

5.3 Estudo Complementar para Validação da Automatização do QM²

O estudo de viabilidade foi realizado com um projeto real conforme apresentado nas seções anteriores. Esse projeto apresenta todos os elementos necessários para a avaliação do QM² e da automatização realizada. Destaca-se que antes da execução do estudo piloto, realizou-se uma sequência de estudos de verificação do ambiente de avaliação, na qual foram elaborados modelos (diagramas) e realizadas avaliações de maneira *ad-hoc*.

Salienta-se que os projetos elaborados para o estudo complementar são modelados em UML. Esses modelos contêm arranjos de pacotes, componentes e classes que associam-se simulando projetos reais. Durante a elaboração desses modelos, preocupou-se em construir diagramas estruturados de várias maneiras, isso com o intuito de abranger o máximo possível os casos de projetos, desde um simples diagrama de classes até diagramas com vários níveis hierárquicos e cada nível com todos os componentes previstos para o cálculo do subatributo A1, descrito na Seção 5.2.4.

Na Tabela 5.8 apresentam-se os dados relevantes do conjunto de seis projetos elaborados para a realização do estudo complementar. Esse estudo teve o intuito de analisar o conjunto de definições (modelo de qualidade, métricas, avaliação e síntese) realizadas com base no QM² para o estudo de viabilidade. Ademais, visou-se realizar uma sequência de simulações para avaliar o comportamento do ambiente antes da execução do estudo de viabilidade. A seguir são descritos resumidamente os projetos e os resultados obtidos.

- ***Projeto_A:*** Esse projeto contêm 16 classes, todas no primeiro nível do projeto, não contêm pacotes e componentes[7].
- ***Projeto_B:*** Nesse projeto o cálculo das métricas explorou até o segundo nível e os componentes estão em todos os níveis.

[7]Nota-se que o valor final do subatributo A1 é *5* para o Projeto_A, também com o mesmo caráter ilustrativo mencionado para o estudo de viabilidade apresentado nas seções anteriores. Essa observação é válida para os demais projetos apresentados nesse estudo complementar.

- ***Projeto_C:*** O projeto contém um conjunto de pacotes e classes aninhados que atingem o terceiro nível. Nesse projeto simula-se o cálculo das métricas quando os elementos aparecem em vários níveis do projeto.

- ***Projeto_D:*** O projeto foi elaborado para o cálculo das métricas em diagramas contendo pacotes, componentes e classes no mesmo nível. Esse projeto contém no mesmo nível um pacote com cinco classes, um componente com três classes, e por fim, no nível do projeto mais oito classes.

- ***Projeto_E:*** Esse projeto abrange os possíveis projetos compostos por um número elevado de elementos, principalmente no nível mais superior.

- ***Projeto_F:*** Esse projeto foi elaborado focando em simular projetos com um número elevado de pacotes, componentes e classes, exceto no nível mais superior.

Destaca-se o cálculo para o *Projeto_D* apresentado na Tabela 5.8. Esse projeto contém no primeiro nível o componente ***Projeto_D.Componente_A*** com 5 classes, $CCF\_i = 0$ e o somatório $(\sum_{j=1}^{n} A1_j) = 0$. Com a aplicação da Equação 5.1, obtém-se o valor $A1_i = 0$ para esse componente.

Em seguida, calcula-se para o pacote ***Projeto_D.Pacote_B*** que tem 3 classes, $CCF_i = 2$ e o somatório $(\sum_{j=1}^{n} A1_j) = 0$. O resultado para esse elemento é $A1_i = 1$. Por fim, aplica-se a Equação 5.1 para o projeto ***Projeto_D***, sendo que o somatório dos subelementos é $(\sum_{j=1}^{n} A1_j) = 1$, com oito classes, um pacote e um componente e o $CCF_i = 3$. Então, o resultado para a compreensibilidade desse projeto é ***1,55***, em um intervalo de 0 a 10.

Ressalta-se que os projetos elaborados durante a execução desse estudo complementar simulam as propriedades de projetos convencionais em termos de organização estrutural (pacotes, componentes e classes). Além disso, os valores obtidos no cálculo das métricas desses projetos são consistentes com os resultados obtidos de forma manual. O mesmo pôde ser observado para o projeto contemplado no estudo de viabilidade apresentado nas seções anteriores. Por fim, realizou-se baterias de avaliação, com o intuito de validar o mecanismo de versionamento do projeto desenvolvido, o qual pode ser usado de forma essencial para identificar os fragmentos problemáticos dos projetos em medição e acompanhar sua evolução.

Tabela 5.8: Estudos Complementares.

Elementos	Nº Pacotes	Nº Comp.	Nº Classes	Nível	Valor A1
Projeto_A					
Projeto_A	0	0	16	0	**5**
Projeto_B					
Projeto_B.CompBlack.ComponenteBlackLine	0	0	5	2	0
Projeto_B.CompWhite.ComponenteBlue	0	0	3	2	1
Projeto_B.CompWhite.ComponenteGreen	0	0	4	2	1
Projeto_B.CompWhite.ComponenteRed	0	0	4	2	1
Projeto_B.CompWhite	0	3	0	1	1,5
Projeto_B.CompBlack	0	1	0	1	1
Projeto_B	0	2	0	0	**1,625**
Projeto_C					
Projeto_C.PacoteWhite.PacoteRed.PacoteRL	0	0	2	3	1
Projeto_C.PacoteWhite.PacoteRed	1	0	2	2	1,166
Projeto_C.PacoteWhite.PacoteBlue	0	0	3	2	1
Projeto_C.PacoteWhite.PacoteGreen	0	0	4	2	1
Projeto_C.PacoteWhite	3	0	0	1	1,527
Projeto_C.PacoteBlack	0	0	5	1	0
Projeto_C	2	0	0	0	**1,381**
Projeto_D					
Projeto_D.Componente_A	0	0	5	1	0
Projeto_D.Pacote_B	0	0	3	1	1
Projeto_D	1	1	8	0	**1,55**
Projeto_E					
Projeto_E.Pacote_A	0	0	10	1	1,5
Projeto_E.Pacote_B	0	0	12	1	2,5
Projeto_E.Componente_C	0	0	16	1	5
Projeto_E.Componente_D	0	0	16	1	5
Projeto_E	2	2	12	0	**5,437**
Projeto_F					
Projeto_F.Pacote_A	0	0	10	1	1,5
Projeto_F.Pacote_B	0	0	16	1	5
Projeto_F.Componente_C	0	0	12	1	2,5
Projeto_F.Componente_D	0	0	16	1	5
Projeto_F	2	2	0	0	**2,75**

5.4 Discussões sobre o Processo de Avaliação e Estudos Realizados

Nesta seção apresentam-se as discussões sobre o processo de avaliação definido pelo QM^2, apresentado no Capítulo 3. Além disso, destacam-se os pontos essenciais dos mecanismos de automatização apresentados no Capítulo 4. Por fim, discute-se os resultados associados ao estudo de viabilidade conduzido e descrito neste capítulo.

5.4.1 Discussões Sobre o *Framework* Conceitual QM^2

Os resultados obtidos nesse estudo de viabilidade mostram que o *framework* conceitual QM^2 pode ser aplicado em avaliações de projetos de software, pois apresenta uma sequência de processo que prevê e apoia a definição dos documentos e artefatos necessários para a avaliação. Os resultados obtidos no estudo de viabilidade evidenciam que o QM^2 pode ser aplicado em diagramas de pacote, componentes e classes (elementos estruturais), sendo que esses diagramas podem ser híbridos.

Além disso, por meio do processo de definição do QM^2, identifica-se o modelo de qualidade a ser utilizado na avaliação em um domínio de aplicação específico, assim como os refinamentos, seleção e definição dos atributos de qualidade. Esses refinamentos resultam em um modelo de qualidade final.

Outra atividade prevista pelo QM^2 é a definição de métricas internas e a sua associação com os atributos do modelo de qualidade final. No contexto desse estudo de viabilidade, foram consideradas métricas para avaliação do atributo de qualidade compreensibilidade estrutural, o qual está presente no modelo de qualidade definido por Ahrens et al. (2011). Ressalta-se, porém, que a especificação de um novo conjunto de métricas para projetos de Sistemas Embarcados (ou sistemas de software em geral) está fora do escopo deste trabalho.

Nota-se que o QM^2 inclui atividades que definem a condução do processo de avaliação, porém não realiza a verificação da conformidade entre os artefatos, deixando essa responsabilidade ao avaliador. Neste trabalho, entende-se por "verificação de conformidade" o alinhamento entre o modelo de qualidade selecionado/definido e o tipo de artefato que se deseja medir, ou seja, se é possível computar o nível de qualidade almejado para o projeto de software em questão. Além disso, os avaliadores são responsáveis pela definição dos limiares (do inglês, *thresholds*) que definem os níveis de qualidade. As dificuldades estão

em estabelecer os relacionamentos entre as métricas internas com os atributos de qualidade, uma vez que, considera-se um conjunto de métricas e modelos de qualidade para os mais variados domínios de aplicação.

5.4.2 Discussões Sobre o Ambiente de Avaliação Automatizado

A automatização do *framework* QM^2 inclui recursos para: (i) a definição das métricas em formato *XML*; (ii) a extração das métricas dos diagramas em formato *XMI*; (iii) a definição do modelo de qualidade (incluindo o mapeamento das métricas internas com os atributos de qualidade); e (iv) síntese dos resultados. As atividades (i) e (ii) são apoiadas pela API *OpenCore*, enquanto as atividades (iii) e (iv) são apoiadas pela plataforma *Spago4Q*, sendo que a última baseia-se nos documentos *KPI*.

O estudo de viabilidade foi apoiado pela ferramenta *MMTool* no controle e versionamento dos projetos, no gerenciamento das definições de métricas e na autenticação dos usuários. Realizou-se diversas avaliações com o intuito de validar o mecanismo de versionamento do projeto desenvolvido, as quais foram essenciais para identificar os fragmentos problemáticos do projeto e acompanhar a sua evolução.

Ressalta-se que apesar dos recursos automatizados disponibilizados, algumas atividades devem ser realizadas de forma manual pelos usuários desses recursos (por exemplo, projetistas de software), estando dessa forma propensas a erros de execução.

Em particular, as métricas internas devem ser codificadas em formato *XML*, de acordo com a gramática definida pela API *OpenCore*. Além disso, os projetos de software devem ser criados em ferramentas que exportam arquivos em formato *XMI* e que não estão integradas ao ambiente desenvolvido neste trabalho.

Outra atividade que deve ser realizada manualmente é a criação do modelo de qualidade na plataforma *Spago4Q*, incluindo toda a configuração de *Datasets* e relatórios *KPI*, o qual inclui o mapeamento das métricas e atributos externos.

5.5 Trabalhos Relacionados

Embora nos estudos realizados durante este trabalho não se tenha encontrado iniciativas de definição de *frameworks* que contemplassem todo o processo de avaliação de modelos de projetos, nesta seção são descritos alguns trabalhos que se relacionam com etapas do *framework* QM^2.

Na pesquisa realizada por Macia (2009), foram realizados dois estudo experimentais. O primeiro estudo avaliou a eficácia de um conjunto de estratégias de detecção de problemas de *design* específicos ocorridos em modelos UML. O segundo estudo utilizou o modelo de qualidade QMOOD (BANSIYA; DAVIS, 2002) para avaliar o *design* de software em diagramas de classes.

Para a realização desses dois estudos, implementou-se a ferramenta QCDTool, que automatiza a aplicação de métricas, estratégias de detecção e modelos de qualidade em diagramas de classe, além de automatizar a aplicação do modelo QMOOD. Essa ferramenta inclui mecanismos para realizar a interpretação dos modelos, aplicação e definição de métricas em modelos, aplicação de estratégias de detecção e modelos de qualidade, e visualização e exportação dos resultados.

Em relação ao estudo de avaliação de *design* utilizando o modelo de qualidade QMOOD, foram avaliadas quatro versões do *design* de um *framework* de desenvolvimento. Nesse estudo procurou-se identificar as alterações do nível de qualidade dos atributos de compreensibilidade, flexibilidade e reusabilidade. De acordo com Macia (2009), o modelo QMOOD mostrou-se uma possível maneira para identificar variações em propriedades de *design* e, consequentemente, na análise dos atributos de qualidade entre as versões analisadas. O descrito assemelha-se com as etapas de extração de métricas de modelos e avaliação de níveis de qualidade com base em modelos de qualidade deste trabalho.

No contexto de desenvolvimento de software orientado a aspectos, Sant'Anna (2004) apresenta um *framework* de avaliação, que reúne um conjunto de métricas e a definição de um modelo de qualidade. Esse *framework* foi avaliado em dois estudos empíricos de domínios distintos, analisando o nível de qualidade dos atributos de manutenibilidade e reusabilidade. Nos casos abordados, concluiu-se que a programação orientada a aspectos gerou soluções com maior manutenibilidade e reusabilidade do que a orientada a objetos.

Outra iniciativa relacionada à definição de modelo de qualidade para projetos de software OO foi apresentada por Bansiya e Davis (2002). Esse trabalho foi descrito na Seção 2.3.1 do Capítulo 2.

O projeto Qualipso teve por finalidade a definição e implementação de tecnologias, procedimentos e políticas para melhorar as práticas de desenvolvimento de software livre. Um dos produtos desse projeto é um modelo de maturidade para desenvolvimento de software livre – *Open Maturity Model* (PETRINJA et al., 2009) baseado no CMMI (CMMI Product Team, 2010). Esse modelo foi automatizado na plataforma *Spago4Q* e define um conjunto de práticas que devem ser aplicadas para garantir a confiabilidade durante o

desenvolvimento.

Ainda em relação à plataforma *Spago4Q*, ela é normalmente empregada na avaliação de processos de desenvolvimento, focando no seu monitoramento com base em modelos de maturidade, tais como, CMMI, ISO 9001:2000 e AQAP. Nesse contexto, Ardagna et al. (2010) formalizaram um modelo para a avaliação de processos de desenvolvimento, além disso esse modelo de qualidade é integrado na plataforma *Spago4Q*. Segundo os autores, o ambiente permite a avaliação de múltiplos projetos e múltiplos processos. Além disso, o principal benefício da solução é a possibilidade de analisar a performance do processo de desenvolvimento em diferentes pontos de visão por meio dos *KPI's*.

Em outra iniciativa no contexto do projeto Qualipso, Bianco et al. (2010) apresentaram um abordagem para a avaliação de qualidade de produtos de software para *Open Source Software (OSS)*, focando no atributo de confiabilidade. Para isso, definiram um modelo de qualidade conceitual baseado na norma ISO-9126 (ISO/IEC, 2001). Ademais, para a coleta dos dados realizaram a implementação de um conjunto de ferramentas e integração. Dentre elas destaca-se a utilização da plataforma *Spago4Q* para agregar os dados, computar os indicadores de resultados e construir *dashboards* para interpretação. A ferramenta MACXIM foi desenvolvida para realizar a medição de propriedades estáticas do código-fonte, é integrada com o *Spago4Q* e computa 70 métricas de código.

No trabalho de Filho (2013) foram implementados extratores para a plataforma *Spago4Q* e realizado a integração com a ferramenta Kalibro Metrics, que passou a analisar projetos de código Java, C e C++. Além disso, foram desenvolvidas interfaces web para a coleta de métricas usando a ferramenta Analizo por meio do Kalibro Metrics. Por fim, nesse trabalho foram realizadas análises de métricas de projetos extraídas de código-fonte.

5.6 Considerações Finais

Neste capítulo apresentou-se o estudo de viabilidade conduzido para a avaliação do *framework* QM^2 e do ambiente automatizado desenvolvido. O estudo de viabilidade foi aplicado no domínio de Sistemas Embarcados robóticos, no qual executou-se a avaliação de um projeto de arquitetura de referência para robôs, modelado em UML. Nesse estudo, computou-se um nível de qualidade hipotético para o atributo Compreensibilidade Estrutural dos diagramas, utilizando-se medidas internas extraídas dos modelos.

No contexto desse estudo de viabilidade, foram estabelecidos os documentos e artefatos previstos pelo QM^2. Além de avaliar a aplicabilidade do *framework* conceitual, foram

analisados os funcionamentos referente ao ambiente automatizado em um estudo completo, que envolveu todos os processos automatizados.

Além disso, descreveu-se um estudo complementar elaborado em um conjunto de pequenos projetos, servindo como estudo piloto para a execução do estudo de viabilidade. Nesse estudo complementar foram elaborados seis projetos, com no máximo 70 classes e 16 pacotes, e o processo automatizado do QM^2 foi executado por completo. Ressalta-se que devido ao tamanho desses projetos, foi possível realizar a análise e verificação manual dos resultados.

Capítulo 6

Conclusão

A qualidade de produtos de software pode ser medida por meio da utilização de métricas, que auxiliam na organização, no monitoramento, na identificação e na prevenção de falhas durante o desenvolvimento (ROCHA et al., 2001). A associação dos valores obtidos para as métricas com atributos externos de qualidade possibilita a identificação de falhas durante o processo de desenvolvimento, que quando detectadas em fases iniciais do processo, ajudam a reduzir os custos de manutenção (LIU et al., 2000).

Neste trabalho, apresentou-se a proposta de um *framework* conceitual para apoiar o processo de avaliação de artefatos de software, com ênfase na etapa de projeto (*design*). No *framework*, intitulado QM^2, indicam-se quais são as principais atividades e recursos prioritários para a avaliação dos artefatos. O *framework*, cuja automatização também foi descrita neste trabalho, apresenta características genéricas em termos dos modelos de qualidade, das métricas que podem ser coletadas e dos domínios de aplicação nos quais ele pode ser empregado.

A viabilidade de aplicação do *framework* QM^2, em conjunto com o ambiente automatizado proposto, foi avaliada em um estudo no contexto de Sistemas Embarcados robóticos. O sistema robótico alvo foi modelado em linguagem UML com base em uma arquitetura de referência para sistemas dessa natureza. O estudo considerou métricas e modelos de qualidade específicos para Sistemas Embarcados. As métricas foram implementadas para permitir a obtenção do nível de qualidade de acordo com o atributo de compreensibilidade estrutural. De acordo com Ahrens et al. (2011), esse atributo impacta nas características de mutabilidade, reusabilidade e compreensibilidade de Sistemas Embarcados. A execução do estudo de viabilidade e os resultados referentes ao processo evidenciam a principio que o ambiente pode ser aplicado em avaliação de modelos de software em nível de pro-

jeto, pois apresenta uma sequência bem definida e ajustável, se necessário, de atividades a serem cumpridas.

Durante a execução do estudo de viabilidade e de um estudo complementar com modelos de projeto genéricos, observou-se que os mecanismos automatizados criados forneceram apoio computacional para o processo de avaliação do nível de qualidade dos modelos. Entre outras funcionalidades, os mecanismos incluem recursos para a definição de métricas em formato *XML* e para a computação das métricas para diagramas representados em formato *XMI*. A API *OpenCore*, reutilizada para a implementação do motor de métricas da ferramenta *MMTool*, atendeu aos propósitos da extração de métricas. O armazenamento dos resultados e o controle do histórico de versões das medições realizadas são gerenciadas pela ferramenta *MMTool*.

A etapa de automatização que envolve o mapeamento de métricas com atributos externos de qualidade é apoiada pela plataforma de avaliação *Spago4Q*. Nessa plataforma, realizam-se os processos de avaliação e síntese, assim como partes do processo de definição, tais como, a definição do modelo de qualidade e a definição parcial do mapeamento.

As contribuições obtidas com este trabalho, as limitações observadas e as possibilidades de trabalhos futuros são sumarizadas nas próximas seções.

6.1 Contribuições

Podem-se destacar como principais contribuições deste trabalho:

- A definição do *framework* conceitual QM^2, que estabelece os processos necessários para realizar a avaliação do nível de qualidade de artefatos de software em nível de projeto, ou seja, modelos de software (Capítulo 3).
- A criação de mecanismos de apoio automatizado aos processos incluídos no QM^2, o que incluiu a construção de uma ferramenta de gestão de projetos de medição de modelos e a reutilização e configuração de APIs e outras ferramentas de software livre (Capítulo 4).
- A condução de um estudo de viabilidade para avaliar a aplicabilidade tanto do *framework* proposto quanto dos mecanismos criados (Capítulo 5).

Como contribuições secundárias deste trabalho, destacam-se:

- O mapeamento de atributos de qualidade presentes em diversos modelos de qualidade propostos na literatura.

- A implementação de métricas internas ainda não disponíveis pela ferramenta *SDMetrics/OpenCore*.

- O estudo e a configuração completa da plataforma *Spago4Q*, cujo emprego até o presente momento concentrava-se na avaliação de artefatos de software no nível de implementação (ou seja, código-fonte).

6.2 Limitações do Trabalho

Algumas limitações do trabalho realizado são apresentadas a seguir, relacionadas ao *framework* QM^2 proposto e ao ambiente de apoio automatizado:

- **QM^2**: Apenas um estudo de viabilidade foi realizado para verificar a aplicabilidade dos processos contidos no *framework*. Apesar disso, observou-se que existe a possibilidade de personalização desses processos de acordo com as características dos artefatos manipulados (por exemplo, modelos de qualidade e conjuntos de métricas) e dos mecanismos de apoio automatizado aos processos.

- **QM^2**: Não foram realizadas comparações entre o *framework* (e o processo subjacente) proposto com outros processos de avaliação de artefatos de software. Como justificativa a essa limitação, ressalta-se que nas buscas por trabalhos relacionados, não foram identificados outros *frameworks* que contemplassem o processo completo de avaliação de artefatos de software. Por exemplo, mesmo considerando a plataforma *Spago4Q*, que foi criada durante um projeto de pesquisa cujo objetivo foi prover meios de avaliar a confiabilidade de software livre por meio de métricas (Qualipso, 2009), não foram encontrados trabalhos oriundos desse projeto que descrevessem de forma abrangente o processo de avaliação.

- **Ambiente automatizado**: o ambiente não possibilita a automatização completa do processo subjacente do QM^2. Esses pontos foram discutidos na Seção 5.4.2.

- **Ambiente automatizado**: o motor de métricas, baseado na API *OpenCore*, não computa métricas para diagramas de comunicação e sequência, o que impossibilitou, no estudo de viabilidade, a obtenção de níveis de qualidade relacionados a alguns atributos presentes no modelo de (AHRENS et al., 2011).

- **Ambiente automatizado**: apesar da realização do controle de versões de projetos de medição por meio da ferramenta *MMTool*, não estão disponíveis funcionalidades para analisar os dados relacionados à evolução desses projetos.
- **Ambiente automatizado**: a integração da plataforma *Spago4Q* com a ferramenta *MMTool* está limitada. Pois atualmente, o usuário deve acessar a ferramenta *MMTool*, e posteriormente é apresentada a opção para utilizar a plataforma *Spago4Q* e ter acesso ao banco de dados compartilhado entre os mecanismos.
- **QM^2 e ambiente automatizado**: Considerando a natureza deste trabalho como sendo uma proposta de *framework* e de um protótipo de ambiente automatizado, a combinação desses elementos não foi empregada em experimentos práticos. Porém, os resultados obtidos neste trabalho podem ser usados como referência para futuros estudos.

Na próxima seção descrevem-se possibilidades de trabalhos futuros, como forma de resolver algumas limitações observadas e evoluir o trabalho realizado.

6.3 Trabalhos Futuros

As possibilidades apresentadas a seguir agrupam-se em tópicos avaliativos e tecnológicos.

Tópicos Avaliativos:

- O ambiente foi utilizado em um estudo de viabilidade no domínio de Sistemas Embarcados robóticos (arquitetura de referência modelada em UML). Portanto, pode-se considerar outros domínios de aplicação e outras linguagens de modelagem para avaliações em estudos experimentais;
- As métricas consideradas no estudo de viabilidade compreendem propriedades estruturais dos modelos UML avaliados. Trabalhos futuros podem considerar métricas que podem ser aplicadas em propriedades comportamentais da UML, permitindo a avaliação de outros atributos de qualidade presentes nos modelos de Ahrens et al. (2011) e de outros autores.
- Estudos experimentais envolvendo diferentes portes de projetos de software, o que permitirá aumentar a confiança de que o *framework* QM^2 e o ferramental associado que apoiam de forma adequada o processo de avaliação abordado.

Tópicos Tecnológicos:

- Melhorias na interface gráfica da ferramenta *MMTool* e desenvolvimento de um módulo para visualização dos resultados das métricas internas, incluindo o histórico de medições.
- Integração completa da ferramenta *MMTool* com a plataforma *Spago4Q* por meio de *Web Services* ou outras tecnologias adequadas.

Referências

ABDELLATIEF, M.; SULTAN, A. B. M.; GHANI, A. A. A.; JABAR, M. A. A mapping study to investigate component-based software system metrics. *Journal of Systems and Software*, Elsevier Inc., v. 86, n. 3, p. 587 – 603, march 2013. ISSN 0164-1212.

ABRAN, A. *Software Metrics and Software Metrology*. 1^{st}. ed. Hoboken, New Jersey, USA: John Wiley & Sons, Inc. – IEEE Computer Society Press, 2010. ISBN 9780470597200.

ABREU, F. B.; CARAPUCA, R. Object-oriented software engineering: Measuring and controlling the development process. In: *Proceedings of the 4^{th} International Conference on Software Quality (ICSQ'94)*. McLean, VA, USA: ASQ – American Society for Quality, 1994. p. 1–8.

ABUASAD, A.; ALSMADI, I. Evaluating the correlation between software defect and design coupling metrics. In: *Proceedings of the IEEE International Conference on Computer, Information and Telecommunication Systems (IEEE CITS'12)*. Amman, Jordan: IEEE Computer Society Press, 2012. p. 1–5.

AGGARWAL, K. K.; SINGH, Y.; KAUR, A.; MALHOTRA, R. Design metrics for object-oriented software. *Journal of Object Technology*, ETH Zurich, v. 6, n. 1, p. 121 – 138, January-February 2007.

AHRENS, D.; FREY, A.; PFEIFFER, A.; BERTRAM, T. Objective evaluation of software architectures in driver assistance systems. *Computer Science - Research and Development*, Springer-Verlag, v. 28, n. 1, p. 23–43, 2011. ISSN 1865-2034.

ALSHAMMARI, B.; FIDGE, C.; CORNEY, D. Security metrics for object-oriented designs. In: *Proceedings of the 21^{st} Australian Software Engineering Conference (ASWEC'2010)*. Auckland, New Zealand: IEEE Computer Society Press, 2010. p. 55–64. ISSN 1530-0803.

ANTONIO, E. A.; FERRARI, F. C.; CAURIN, G. A. P.; FABBRI, S. C. P. F. A set of metrics for characterizing simulink model comprehension. *The Journal of Computer Science and Technology (JCST)*, La Plata, En línea, Argentina, v. 14, n. 2, p. p. 88–94, octubre 2014.

AQAP. *AQAP - Allied Quality Assurance Publication*. USA: NATO - North Atlantic Treaty Organization, 2009. Disponível Online em: `https://ccj.wat.edu.pl/en/Certification/AQAP.html`. Acessado em Fevereiro de 2015.

ARDAGNA, C. A.; DAMIANI, E.; FRATI, F.; OLTOLINA, S.; REGOLI, M.; RUFFATTI, G. Spago4q and the qest nd model: An open source solution for software performance measurement. In: ÅGERFALK, P.; BOLDYREFF, C.; GONZÁLEZ-BARAHONA, J. M.; MADEY, G. R.; NOLL, J. (Ed.). *Open Source Software: New Horizons*. Notre Dame, IN, USA: Springer Berlin Heidelberg, 2010, (IFIP Advances in Information and Communication Technology, v. 319). p. 1–14. ISBN 978-3-642-13243-8.

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. *IEEE Software*, IEEE Computer Society Press, v. 20, n. 5, p. 36–41, Sept 2003. ISSN 0740-7459.

BADRI, M.; BADRI, L.; TOURE, F. Empirical analysis of object-oriented design metrics: Towards a new metric using control flow paths and probabilities. *Journal of Object Technology*, ETH Zurich, v. 8, n. 6, p. 123–142, October 2009.

BALDWIN, C. Y.; CLARK., K. B. *Design Rules*: The power of modularity. Cambridge, Massachusetts, USA: MIT Press, 2000.

BANSIYA, J.; DAVIS, C. G. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering (TSE)*, IEEE Computer Society Press, Piscataway, NJ, USA, v. 28, n. 1, p. 4–17, January 2002. ISSN 0098-5589.

BARTIé, A. *Garantia da qualidade de software*: As melhores práticas de engenharia de software aplicadas à sua empresa. 1^{st}. ed. São Paulo, SP, Brasil: Makron Books, 1992. ISBN 0321486811.

BASILI, V. R.; ROMBACH, H. D. *Integrating Measurement Into Software Environments*. Maryland, USA: University of Maryland – Departament of Computer Science, 1987.

BECKER, C.; RAUBER, A. Decision criteria in digital preservation: What to measure and how. *Journal Of The American Society For Information Science And Technology*, Wiley Online Library, v. 62, n. 6, p. 1009–1028, February 2011.

BEUS-DUKIC, L.; BØEGH, J. COTS software quality evaluation. In: *Proceedings of the 2^{nd} International Conference on COTS-Based Software Systems (ICCBSS'03)*. Ottawa, Canada: Springer, 2003. (Lecture Notes in Computer Science, v. 2580), p. 72–80. ISBN 3-540-00562-5.

BIANCO, V. del; LAVAZZA, L.; MORASCA, S.; TAIBI, D.; TOSI, D. The qualispo approach to oss product quality evaluation. In: *Proceedings of the 3^{rd} International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. New York, NY, USA: ACM, 2010. (FLOSS '10), p. 23–28. ISBN 978-1-60558-978-7.

BOEHM, B.; BROWN, J.; LIPOW, H.; MACLEOD, G.; MERRIT, M. *Characteristics of Software Quality*: Trw series of software technology. Amsterdam, North Holland,: Elsevier North-Holland Pub.Co, 1978.

BOEHM, B. W. (Ed.). *Software Risk Management*. Piscataway, NJ, USA: IEEE Computer Society Press, 1989. ISBN 0-8186-8906-4.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML : Guia do Usuário*. 2^{nd}. ed. Rio de Janeiro, Rio de Janeiro, Brasil: Elsevier Editora Ltda, 2005. ISBN 10 85-352-1784-3.

BRIAND, L.; DEVANBU, P.; MELO, W. An investigation into coupling measures for c++. In: *Proceedings of the International Conference on Software Engineering (ICSE'97)*. Boston, Massachusetts, USA: IEEE Computer Society Press, 1997. p. 412–421. ISSN 0270-5257.

BRIAND, L. C.; WÜST, J. Empirical studies of quality models in object-oriented systems. Elsevier, v. 56, p. 97 – 166, 2002. ISSN 0065-2458.

BRIAND, L. C.; WÜST, J.; DALY, J. W.; PORTER, D. V. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software*, Elsevier, v. 51, n. 3, p. 245–273, 2000.

BUGLIONE, L.; DAMIANI, E.; FRATI, F.; OLTOLINA, S.; RUFFATTI, G. Improving quality and cost-effectiveness in enterprise software application development: An open, holistic approach for project monitoring and control. *Lecture Notes in Business Information Processing*, v. 80 LNBIP, p. 125–139, 2011.

CARVALHO, F.; MEIRA, S.; XAVIER, E.; EULINO, J. An embedded software component maturity model. In: *Proceedings of the 9^{th} International Conference on Quality Software (QSIC'09)*. Jeju, Coreia do Sul: IEEE Computer Society Press, 2009. p. 426–431. ISSN 1550-6002.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering (TSE)*, IEEE Computer Society Press, Piscataway, NJ, USA, v. 20, n. 6, p. 476–493, Jun 1994. ISSN 0098-5589.

CHOI, Y.; LEE, S.; SONG, H.; PARK, J.; KIM, S. Practical s/w component quality evaluation model. In: *Proceedings of the 10^{th} International Conference on Advanced Communication Technology (ICACT'08)*. Gangwon-Do, Coreia do Sul: IEEE Computer Society Press, 2008. v. 1, p. 259–264. ISSN 1738-9445.

CMMI Product Team. *CMMI for Development, Version 1.3*. Pittsburgh, Pennsylvania, USA, 2010.

COLOMBO, A.; DAMIANI, E.; FRATI, F.; OLTOLINA, S.; REED, K.; RUFFATTI, G. The use of a meta-model to support multi-project process measurement. In: *Proceedings of the 15^{th} Asia-Pacific Software Engineering Conference (APSEC'08)*. Beijing, China: IEEE Computer Society Press, 2008. p. 503–510. ISSN 1530-1362.

COLOMBO, R. M. T.; GUERRA, A. C. *Qualidade de Produto de Software*. São Paulo, SP, Brasil: PBQP – Programa Brasileiro da Qualidade e Produtividade em Software / MCT – Ministério de Ciência, Tecnologia e Inovação, 2014. Disponível Online em: `http://www.mct.gov.br/upd_blob/0203/203505.pdf`. Acessado em Dezembro de 2014.

COMELLA-DORDA, S.; DEAN, J. C.; MORRIS, E. J.; OBERNDORF, P. A. A process for cots software product evaluation. In: *Proceedings of the 1^{st} International Conference on COTS-Based Software Systems (ICCBSS'02)*. London, UK, UK: Springer-Verlag, 2002. p. 86–96. ISBN 3-540-43100-4.

DAJSUREN, Y.; BRAND, M. G. van den; SEREBRENIK, A.; ROUBTSOV, S. Simulink models are also software: Modularity assessment. In: *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA'13)*. New York, NY, USA: ACM, 2013. p. 99–106. ISBN 978-1-4503-2126-6.

DALLAL, J. A.; BRIAND, L. C. An object-oriented high-level design-based class cohesion metric. *Information and Software Technology*, Elsevier Science B.V, v. 52, n. 12, p. 1346 – 1361, December 2010. ISSN 0950-5849.

DELONE, W. H.; EPHRAIM; MCLEAN, R.; SCHOLARS, G. E. S. E. The delone and mclean model of information systems success: a ten-year update. *Journal of Management Information Systems*, v. 19, p. 9–30, 2003.

DOUGLASS, B. P. *The Harmony Process*. I-Logix white paper: I-Logix Inc, 2005.

DROMEY, R. G. A model for software product quality. *IEEE Transactions on Software Engineering (TSE)*, IEEE Computer Society Press, Piscataway, NJ, USA, v. 21, n. 2, p. 146–162, February 1995. ISSN 0098-5589.

DUGERDIL, P.; NICULESCU, M. Visualizing software structure understandability. In: *Proceedings of the 23rd Australian Software Engineering Conference (ASWEC'14)*. Sydney, New South Wales, Australia: IEEE Computer Society Press, 2014. p. 110–119.

FENSTERMAKER, S.; GEORGE, D.; KAHNG, A. B.; MANTIK, S.; THIELGES, B. Metrics: a system architecture for design process optimization. In: *Proceedings of the 37th Annual Design Automation Conference (DAC'00)*. New York, NY, USA: ACM Press, 2000. p. 705–710. ISBN 1-58113-187-9.

FENTON, N.; BIEMAN, J. *Software Metrics*: A rigorous and practical approach. 3rd. ed. Boca Raton, FL, USA: CRC Press, 2014. (Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series). ISBN Print: 978-1-4398-3822-8, eBook: 978-1-4398-3823-5.

FILHO, C. M. de O. *Kalibro: Interpretação de Métricas de Código*. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo - SP - Brasil, 2013.

FIRESMITH, D.; HENDERSON-SELLERS, B. *The Open Process Framework: An Introduction*. : Addison-Wesley, 2002. (The OPEN series). ISBN 9780201675108.

FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321193687.

FRIEDENTHAL, S.; MOORE, A.; STEINER, R. *A Practical Guide to SysML: Systems Modeling Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. 560 p. ISBN 0123743796.

FUGGETTA, A. Software process: A roadmap. In: *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*. New York, NY, USA: ACM Press, 2000. p. 25–34. ISBN 1-58113-253-0.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0-201-63361-2.

GEETIKA, R.; SINGH, P. Empirical investigation into static and dynamic coupling metrics. *SIGSOFT – Software Engineering Notes*, ACM Press, New York, NY, USA, v. 39, n. 1, p. 1–8, feb 2014. ISSN 0163-5948.

GENERO, M.; PIATINI, M.; MANSO, E. Finding "early" indicators of uml class diagrams understandability and modifiability. In: *Proceedings of the International Symposium on Empirical Software Engineering (ISESE'04)*. Washington, DC, USA: IEEE Computer Society Press, 2004. p. 207–216. ISBN 0-7695-2165-7.

GENERO, M.; PIATTINI, M.; MANSO, E.; CANTONE, G. Building uml class diagram maintainability prediction models based on early metrics. In: *Proceedings of the 9^{th} International Symposium on Software Metrics (METRICS'03)*. Washington, DC, USA: IEEE Computer Society Press, 2003. p. 263–275. ISBN 0-7695-1987-3.

GOMAA, H.; SCOTT, D. B. Prototyping as a tool in the specification of user requirements. In: *Proceedings of the 5th International Conference on Software Engineering (ICSE '81)*. Piscataway, NJ, USA: IEEE Press, 1981. p. 333–342. ISBN 0-89791-146-6.

GOYAL, P.; JOSHI, G. Qmood metric sets to assess quality of java program. In: *Proceedings of the International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT'14)*. Ghaziabad, India: IEEE Advandng Technology for Humanity, 2014. p. 520 – 533.

GUEDES, G. T. A. *UML 2: uma abordagem prática*. 2^{nd}. ed. São Paulo, São Paulo, Brasil: Novatec, 2011. 488 p. ISBN 978-85-7522-281-2.

HALSTEAD, M. H. *Elements of Software Science (Operating and Programming Systems Series)*. New York, NY, USA: Elsevier Science Inc., 1977. ISBN 0444002057.

HENDERSON-SELLERS, B. *Object-oriented Metrics: Measures of Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN 0-13-239872-9.

HENRY, S.; KAFURA, D. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering (TSE)*, SE-7, n. 5, p. 510–518, Sept 1981. ISSN 0098-5589.

INCOSE. *Systems Engineering Vision 2020, Version 2.03*: INCOSE – International Council on Systems Engineering. 2007. Disponível Online em: `https://www.incose.org/ProductsPubs/pdf/SEVision2020_20071003_v2_03.pdf`. Acessado em Dezembro de 2014.

INCT-SEC. *Instituto Nacional de Ciência e Tecnologia - Sistemas Embarcados Críticos*. 2013. Disponível Online em: `http://www.inct-sec.org/`. Acessado em Fevereiro de 2015.

ISO. *ISO 9001:2000*: Quality management systems – requirements. USA: ISO – The International Organization for Standardization, 2008. Disponível Online em: `http://www.iso.org/iso/catalogue_detail?csnumber=21823`. Acessado em Outubro de 2014.

ISO/IEC. *ISO/IEC 9126, Information technology – Product Quality – Part1: Quality Model*. International Organization for Standardization – ISO, Disponível Online em: `http://www.iso.org/iso/`, June 2001. Acessado em Outubro de 2014.

ISO/IEC. *ISO/IEC 25000 - Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE*. International Organization for Standardization – ISO, Disponível Online em: `http://www.iso.org/iso/`, 2005. Acessado em Outubro de 2014.

ISO/IEC. *ISO/IEC 25010 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)*: System and Software Quality Models. International Organization for Standardization – ISO, Disponível Online em: `https://www.iso.org/obp/ui/iso:std:35733`, 2011. Acessado em Dezembro de 2014.

JABANGWE, R.; BöRSTLER, J.; SMITE, D.; WOHLIN, C. Empirical evidence on the link between object-oriented measures and external quality attributes: A systematic literature review. *Empirical Software Engineering*, Kluwer Academic Publishers, Hingham, MA, USA, v. 20, n. 3, p. 640–693, jun. 2015. ISSN 1382-3256.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. *The Unified Software Development Process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0-201-57169-2.

JACOBSON, I.; CHRISTERSON, M.; JONSSON, P.; ÖVERGAARD, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Boston, MA, USA: Addison-Wesley, 1992. Paperback.

JEONG, H. Y. The practical quality model for embedded system and software. In: *Proceedings of the 16th International Conference on Network-Based Information Systems (NBiS'13)*. Gwangju, Coreia do Sul: IEEE Computer Society Press, 2013. p. 288–291.

JEONG, H. Y.; KIM, Y. H. A Quality Model of Lightweight Component for Embedded System. *Applied Mechanics and Materials*, v. 121-126, p. 4907–4911, oct 2011. ISSN 1662-7482.

JONES, C. *Applied Software Measurement: Global Analysis of Productivity and Quality*. New York, NY, USA: McGraw-Hill Education, 2008. (McGraw Hill professional). ISBN 9780071643863.

KAN, S. H. *Metrics and Models in Software Quality Engineering*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Company Incorporated, 2014. ISBN 9780133988086.

Kanguera Project. *Project Website*. 2010. Disponível Online em: `http://www.mecatronica.eesc.usp.br/wiki/index.php/Kanguera_Project`. Acessado em Janeiro de 2015.

KITCHENHAM, B.; PFLEEGER, S. L. Software quality: The elusive target. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 13, n. 1, p. 12–21, January 1996. ISSN 0740-7459.

KITCHENHAM, B.; PICKARD, L. Towards a constructive quality model. part 2: Statistical techniques for modelling software quality in the esprit request project. *Software Engineering Journal*, IET, v. 2, n. 4, p. 114–126, 1987.

KLÄS, M.; HEIDRICH, J.; MÜNCH, J.; TRENDOWICZ, A. CQML Scheme: A Classification scheme for comprehensive Quality Model Landscapes. In: *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'09)*. Patras, Greece: IEEE Computer Society Press, 2009. p. 243–250.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. 3rdedition. ed. Boston, MA, USA: Addison-Wesley Professional, 2004. Paperback. ISBN 0321197704.

KUMAR, V.; SHARMA, A.; KUMAR, R.; GROVER, P. Quality aspects for component-based systems: A metrics based approach. *Journal Software–Practice & Experience*, John Wiley & Sons, Inc., New York, NY, USA, v. 42, n. 12, p. 1531–1548, dez. 2012. ISSN 0038-0644.

LANZA, M.; DUCASSE, S.; MARINESCU, R. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin, Germany: Springer Berlin Heidelberg, 2010. ISBN 9783642063749.

LARMAN, C. *Applying UML and Patterns*: An introduction to object-oriented analysis and design and iterative development. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN 0131489062.

LAVAGNO, L.; MARTIN, G.; SELIC, B. *UML for Real: Design of Embedded Real-Time Systems*. Berlin, Heidelberg: Springer-Verlag, 2010. ISBN 9781441953681.

LAVAZZA, L.; MORASCA, S.; TAIBI, D.; TOSI, D. An empirical investigation of perceived reliability of open source java programs. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*. New York, NY, USA: ACM, 2012. p. 1109–1114. ISBN 978-1-4503-0857-1.

Lego Mindstorms. *Lego Mindstorms*. 2014. Disponível Online em: `http://www.lego.com/en-us/mindstorms`. Acessado em Novembro de 2014.

LI, W.; HENRY, S. Maintenance metrics for the object oriented paradigm. In: *Proceedings of the 1st International Symposium on Software Metrics*. Baltimore, Maryland, USA: IEEE Computer Society Press, 1993. p. 52–60. ISBN 0-8186-3740-4.

LIGGESMEYER, P.; TRAPP, M. Trends in Embedded Software Engineering. *IEEE Software*, IEEE Computer Society Press, v. 26, n. 3, p. 19–25, 19–25 2009. ISSN 0740-7459.

LINCKE, R.; LUNDBERG, J.; LöWE, W. Comparing software metrics tools. In: *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'08)*. New York, NY, USA: ACM Press, 2008. p. 131–142. ISBN 978-1-60558-050-0.

LIND, K.; HELDAL, R.; HARUTYUNYAN, T.; HEIMDAHL, T. Compsize: Automated size estimation of embedded software components. In: *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)*. Nara, Japan: IEEE Computer Society Press, 2011. p. 86–95. ISBN 978-0-7695-4565-3.

LIU, K.; ZHOU, S.; YANG, H. Quality metrics of object oriented design for software development and re-development. In: *Proceedings of the 1st Asia-Pacific Conference on Quality Software*. Hong Kong: IEEE Computer Society Press, 2000. p. 127–135.

LOBO FILHO, R. J. H. *Integração entre HTML5 e JSF 2.0 em Aplicações Web Offline*. Florianópolis, SC, Brasil: UFSC – Universidade Federal Santa Catarina, 2006. Monografia.

LONG, J. E. *Systems Engineering (SE) 101, coRJfID: Product & Process Engineering Solutions, Vitech training materials*. Viena, Áustria: Vitech Corporation, 2000.

LORENZ, M.; KIDD, J. *Object-oriented Software Metrics: A Practical Guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-179292-X.

LYKINS, F. M. *Adapting UML for an Object-Oriented Systems Engineering Method (OOSEM)*. Minneapolis: Proceedings of the INCOSE International Symposium, 2000.

MACIA, I. *Avaliação da Qualidade de Software com base em Modelos UML*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro/RJ - Brasil, 2009.

MALDONADO, J. C.; FABBRI, S. C. P. F. Verificação e validação de software. In: ROCHA, A. R. C. da; MALDONADO, J. C.; WEBER, K. C. (Ed.). *Qualidade de Software: Teoria e Prática*. São Paulo, SP, Brasil: Prentice Hall, 2001. p. 66–85. ISBN 978-3-642-13243-8.

MARCHESI, M. OOA metrics for the unified modeling language. In: *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98)*. Washington, DC, USA: IEEE Computer Society Press, 1998. p. 67–. ISBN 0-8186-8421-6.

MARTíNEZ, J.; MERINO, P.; SALMERON, A.; MALPARTIDA, F. et al. UML-Based Model-Driven Development for HSDPA Design. *IEEE Software*, v. 26, n. 3, p. 26–33, 2009.

MCCABE, T. J. A complexity measure. *IEEE Transactions on Software Engineering (TSE)*, IEEE Press, Piscataway, NJ, USA, v. 2, n. 4, p. 308–320, jul. 1976. ISSN 0098-5589.

MCCABE, T. J.; BUTLER, C. W. Design complexity measurement and testing. *Communications of the ACM*, ACM, New York, NY, USA, v. 32, n. 12, p. 1415–1425, dez. 1989. ISSN 0001-0782.

MCCALL, J. A.; RICHARDS, P. K.; WALTERS, G. F. *Factors in Software Quality*. 1977. Vols. I, II, III, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055.

MEIRELLES, P.; SANTOS, C.; MIRANDA, J.; KON, F.; TERCEIRO, A.; CHAVEZ, C. A study of the relationships between source code metrics and attractiveness in free software projects. In: *Proceedings of the 24th Brasilian Symposium on Software Engineering (SBES'2010)*. Salvador, Bahia, Brasil: IEEE Computer Society Press, 2010. p. 11–20.

MEYER, B. *Object-Oriented Software Construction*. 1st. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. ISBN 0136290493.

MILICIC, D. *Software Quality Models and Philosophies*: Software quality attributes and trade-offs, cap. 1, p.3-19. Ronneby, Sweden: School of Engineering, Blekinge Institute of Technology, 2005. Disponível Online em: `http://www.bth.se/com/besq.nsf/pages/017bd879b7c9165dc12570680047aae2?OpenDocument`. Acessado em Dezembro de 2014.

MILLS, E. E. Metrics in the software engineering curriculum. *Annals of Software Engineering*, J. C. Baltzer AG, Science Publishers, Red Bank, NJ, USA, v. 6, n. 1-4, p. 181–200, apr 1999. ISSN 1022-7091.

MURRAY, C. *RUP SE: The Rational Unified Process for Systems Engineering*. 2003. Disponível Online em: `http://www.ibm.com/developerworks/rational/library/content`. Acessado em Dezembro de 2014.

Object Management Group. *Unified Modeling Language (UML)*. 1997. Disponível Online em: `http://www.uml.org/`. Acessado em Setembro de 2014.

Object Management Group. *MDA Guide Version 1.0.1*. 2003. Disponível Online em: `http://www.omg.org/cgi-bin/doc?omg/03-06-01`. Acessado em Dezembro de 2014.

Object Management Group. *Documents Associated With Business Process Model And Notation (BPMN) Version 2.0*: Release date: January 2011. normative documents: formal/2011-01-03. 2011. Disponível Online em: `http://www.omg.org/spec/BPMN/2.0/`. Acessado em Setembro de 2014.

Object Management Group. *SysML*. 2014. Disponível Online em: `http://www.omgsysml.org/`. Acessado em Dezembro de 2014.

OLAGUE, H.; ETZKORN, L.; GHOLSTON, S.; QUATTLEBAUM, S. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on Software Engineering (TSE)*, v. 33, n. 6, p. 402–419, June 2007. ISSN 0098-5589.

OLIVEIRA JR., E.; GIMENES, I.; MALDONADO, J. Systematic management of variability in uml-based software product lines. *Journal of Universal Computer Science*, v. 16, n. 17, p. 2374–2393, 2010.

OLIVEIRA, L. B. R.; GUESSI, M.; FEITOSA, D.; MANTEUFFEL, C.; GALSTER, M.; OQUENDO, F.; NAKAGAWA, E. Y. An investigation on quality models and quality attributes for embedded systems. In: *Proceedings of The 8th International Conference on Software Engineering Advances (ICSEA'13)*. Venice, Italy: IARIA - International Academy, Research and Association, 2013. p. 523–528. ISBN 978-1-61208-304-9.

Oracle Corporation. *Java BluePrints Model-View-Controller MVC*. Disponível Online em: `http://www.oracle.com/technetwork/java/mvc-detailed-136062.html`, 2002. Acessado em Outubro de 2014.

PAULA, M. H. de. *SARA-MR - Uma Arquitetura de Referência baseada em Loops de Controle para Facilitar Manutenções em Software Robótico Autoadaptativo*. Dissertação (Mestrado) — Departamento de Computação - Universidade Federal de São Carlos (UFSCar), São Carlos, SP, Brasil, 2015.

PETRINJA, E.; NAMBAKAM, R.; SILLITTI, A. Introducing the opensource maturity model. In: *Proceedings of the Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS'09)*. Washington, DC, USA: IEEE Computer Society Press, 2009. p. 37–41. ISBN 978-1-4244-3720-7.

PFLEEGER, S. L. *Software Engineering: Theory and Practice*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998. ISBN 0-13-624842-X.

PRESSMAN, R. S. *Software Engineering*: A Practitioner's Approach. 7^{th}. ed. New York, NY, USA: McGraw-Hill, 2010. (Higher Education). ISBN 9780073375977.

PUTNAM, L.; MYERS, W. *Five Core Metrics: The Intelligence Behind Successful Software Management*. New York, NY, USA: Dorset House Publishing, 2003. ISBN 9780932633552.

Qualipso. *Qualipso – Quality Plataform for Open Source*: O projeto qualipso. 2009. Disponível Online em: `http://qualipso.icmc.usp.br/`. Acessado em Outubro de 2014.

RAWASHDEH, A.; MATALKAH, B. A new software quality model for evaluating cots components. *Journal of Computer Science*, v. 2, p. 373–381, 2008.

ROCHA, A. R. C. da; MALDONADO, J. C.; WEBER, K. C. *Qualidade de Software*: Teoria e prática. 1^{st}. ed. São Paulo, SP, Brasil: Prentice-Hall, 2001. 303 p. ISBN 9788587918543.

ROYCE, W. W. Managing the development of large software systems: Concepts and techniques. In: *Proceedings of the WESCON*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1970. p. 328–338. ISBN 0-89791-216-0.

RUFFATTI, G. *Productivity Intelligence: a 3D analysis of Technical, Economical and Social data*. Disponível Online em: http://www.spagoworld.org/blog/2013/08/productivity-intelligence-a-3d-analysis-of-technical-economical-and-social-data/, August 2013. Acessado em Outubro de 2014.

Sant'Anna, C. N. *Manutenibilidade e Reusabilidade de Software Orientado a Aspectos: Um Framework de Avaliação*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro/RJ - Brasil, 2004.

SCHMIDT, D. C. Guest editor's introduction: Model-driven engineering. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 39, n. 2, p. 25–31, February 2006. ISSN 0018-9162.

SHARMA, M.; GILL, N. S.; SIKKA, S. Survey of object-oriented metrics: Focusing on validation and formal specification. *ACM SIGSOFT Software Engineering Notes*, ACM, New York, NY, USA, v. 37, n. 6, p. 1–5, nov. 2012. ISSN 0163-5948.

SOMMERVILLE, I. *Engenharia de Software*. 9^{th}. ed. São Paulo, SP, Brasil: Pearson Education do Brasil, 2011. ISBN 8579361087.

SOUZA, O. M. de. *Ambiente para Avaliação de Modelos de Sistemas Embarcados Baseada em Métricas*. Dissertação (Exame de Qualificação de Mestrado) — Departamento de Computação - Universidade Federal de São Carlos (UFSCar), São Carlos, SP, Brasil, 2013.

Spago4Q. *Spago4Q – SpagoBI for Quality*: Open source platform to measure, analyze and monitor the quality of products, processes and services. 2014. Disponível Online em: `http://www.spagoworld.org/xwiki/bin/view/Spago4Q/`. Acessado em Outubro de 2014.

SpagoBI. *SpagoBI*: Open source platform for business intelligence. 2014. Disponível Online em: `http://www.spagobi.org/`. Acessado em Setembro de 2014.

THIRUGNANAM, M.; SWATHI.J.N. Quality metrics tool for object oriented programming. *International Journal of Computer Theory and Engineering (IJCTE)*, Internarional, Piscataway, NJ, USA, v. 2, n. 5, p. 712–717, October 2010. ISSN 1793-8201.

TRAVERSO-RIBóN, I.; RUÍZ-RUBE, I.; DODERO, J. M.; PALOMO-DUARTE, M. Open data framework for sustainable assessment in software forges. In: *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics (WIMS'13)*. New York, NY, USA: ACM, 2013. p. 20:1–20:8. ISBN 978-1-4503-1850-1.

WAGNER, S. *Software Product Quality Control*. Stuttgart, Germany: Springer Berlin Heidelberg, 2013. ISBN 9783642385704.

WANG, H. *Software Metrics Collection Techniques for Product Assessment*. 1999. Disponível Online em: `http://www.comp.nus.edu.sg/~wanghao/new/ic52a5_frameVersion.html`. Acessado em Setembro de 2014.

WAZLAWICK, R. *Análise e Projeto de Sistemas da Informação*. 2nd. ed. Rio de Janeiro, RJ, Brasil: Elsevier Brasil, 2010. ISBN 10: 85-352-3916-2.

WEI, H.; LOEFFLER, T.; WEGENER, J. Quality model based on ***ISO/IEC*** 9126 for internal quality of MATLAB/Simulink/Stateflow models. In: *IEEE International Conference on Industrial Technology (ICIT'12)*. Athens, Greece: IEEE Computer Society Press, 2012. p. 325–330. ISBN 0-7695-1796-X.

WEILKIENS, T. *Systems Engineering with SysML/UML*: Modeling, Analysis, Design. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN 9780123742742.

WHITE, S. Using BPMN to Model a BPEL Process. *BPTrends - BPM analysis, Opinion and insight*, v. 3, n. 3, p. 1–18, March 2005.

WOODFIELD, S.; SHEN, V.; DUNSMORE, H. A study of several metrics for programming effort. *Journal of Systems and Software*, v. 2, n. 2, p. 97 – 103, 1981. ISSN 0164-1212.

WÜST, J. *OpenCore - Open Source XMI Parser and UML Metrics*. 2013. Disponível Online em: `http://www.sdmetrics.com/OpenCore.html`. Acessado em Setembro de 2014.

ZHANG, S.; WANG, Y. An extension of semest: the online software engineering measurement tool. In: *Proceedings of the Canadian Conference on Electrical and Computer Engineering*. Niagara Falls, Ontario, Canada: IEEE Technology Driving Innovation, 2004. v. 3, p. 1519–1522. ISSN 0840-7789.

Apêndice A

Configuração da API *OpenCore*

A.1 Arquivos de Configuração da *OpenCore*

Neste apêndice são apresentadas as estruturas dos arquivos de configuração da API *OpenCore*, que são pré requisitos para a sua execução. O arquivo *XMI Source File* contém os diagramas do projeto. O *Metamodel Definition* e as *Transformation Rules* associam-se para realizar a leitura dos diagramas e executar os *parsers* quando for necessário. Por fim, o arquivo *Metrics Definition* armazena as métricas definidas em formato *XML*.

A.1.1 *XMI Source File*

O *XMI Source File* armazena a representação da modelagem do projeto em que se pretende executar a análise. O arquivo é gerado por meio da ferramenta utilizada para elaborar a modelagem do projeto, ou por outro aplicativo quando a ferramenta de modelagem não disponibiliza a exportação para o formato *XMI*. O exemplo de Código A.1 mostra o segmento de um diagrama em formato *XMI Source File*, da UML2.x para a representação da definição da classe "Aluno" (O nome da classe pode ser visto no final da linha 7).

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model
  xmlns:uml="http://www.omg.org/spec/UML/20100901"
  xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
  xmi:version="2.1" xmi:id="_0rUq0OtWEeOzmov7vArKOQ" name="projectcata">
  ...
  <packagedElement xmi:type="uml:Class" xmi:id="_0rUq1-tWEeOvArKOQ" name="Aluno">
  <eAnnotations xmi:id="_0rUq2OtWEeOzmov7vArKOQ" source="Objing">
    <contents xmi:type="uml:Property" xmi:id="_0rUq2etWEeArKOQ" name="Visibility">
    <defaultValue xmi:type="uml:LiteralString" xmi:id="_0rUq2ov7vArKOQ" value="0"/>
    </contents>
  </eAnnotations>
  </packagedElement>
  ...
</uml:Model>
```

Código A.1: Estrutura do Modelo UML em *XMI Source File*.

A.1.2 *Metamodel Definition*

Este metamodelo define os tipos e as informações que são armazenadas em cada elemento do modelo UML (*e.g.,* classes, pacotes, componentes, sequência e comunicação) e nos modelos SysML (*e.g.,* diagramas de requisitos, definição de bloco e definição de blocos internos). As informações do metamodelo são definidas em um arquivo *XML*, conforme estrutura do Código A.2.

```
<sdmetricsmetamodel version="2.0" >
<modelelement name="element1">
     <attribute name="attr1" type="data" multiplicity="one"/>
     <attribute name="attr2" type="ref" multiplicity="many"/>
     ..
</modelelement>
<modelelement name="element2" parent="element1">
     <attribute name="attr3" type="extref" multiplicity="one"/>
     ...
</modelelement>
...
</sdmetricsmetamodel>
```

Código A.2: Estrutura do Metamodelo em *XML*.

O arquivo *Metamodel Definition* contém as informações necessárias que fornecem a base para a definição e o cálculo das métricas de um projeto. O metamodelo possui atributos que armazenam dados dos elementos do modelo (*e.g.,* o nome da classe, visibilidade das operações e relacionamentos das classes), atributos que armazenam informações de outros elementos (*e.g.,* tipos de atributos de outras classes e os destinatários de uma mensagem) e atributos especiais utilizados em extensões para definição de perfis.

A.1.3 *Transformation Rules*

As regras de transformação definem uma lista de elementos para transformar o *XMI Source File* e fornecer informações para o *Metamodel Definition*, estabelecendo a forma em que os elementos do *Metamodel Definition* serão recuperados dos *XMI Source File*. As informações necessárias são recuperadas de cada elemento do modelo por meio do mapeamento dos atributos do *Metamodel Definition* com as informações do *XMI Source File*.

Os arquivos *Transformation Rules* são estruturados conforme apresentado no Código A.3, eles são representados no formato *XMI* e construídos pelos atributos expressos na definição do metamodelo. Assim, cada atributo é recuperado por *trigger* a partir do atributo raiz. Os *triggers* são mecanismos que realizam a identificação dos atributos de um elemento do modelo, tais como, ID, nome, contexto e visibilidade. Por exemplo, para realizar a recuperação do atributo ID utiliza-se a *trigger* <*trigger name="id" type="attrval" attr="xmi.id"/*>. Os atributos que representam os elementos são únicos, no qual localiza o *name="id"* do tipo atributo-valor (*type="attrval"*) e atributo igual a *"xmi.id"*.

```
1  <xmitransformations version="2.0" >
2     <xmitransformation ...xmitransformation attributes... />
3        <trigger ...trigger attributes... />
4        <trigger ...trigger attributes... />
5        ...
6     </xmitransformation>
7     <xmitransformation ...xmitransformation attributes... />
8        <trigger ...trigger attributes... />
9        ...
10    </xmitransformation>
11    ...
12 </xmitransformations>
```

Código A.3: Estrutura das Regras de Transformação em *XML*.

A.1.4 *Metrics Definition*

As métricas são definidas em arquivos no formato *XML*, com base no domínio de aplicação da avaliação e dos modelos dos projetos a serem calculados. O arquivo *Metrics Definition* contém as métricas que se pretende coletar por meio da *OpenCore*. Pode ser definido um conjunto de métricas para vários tipos de diagramas, tais como diagrama de classe, diagrama de sequência e diagrama de caso de uso.

A definição das métrica não está restrita a um conjunto fixo de métricas, pois utilizando o padrão SDMetricsML (*SDMetrics Markup Language*) é possível definir e calcular métricas personalizadas baseadas no formato *XML*. Essa definição é feita utilizando *tags*

e atributos predefinidos em formato de elementos *XML* conforme o exemplo apresentado a seguir no Código A.4.

```
1 <metric name="metricname" domain="metricdomain" category="metriccategory"
2   internal="true/false" inheritable="true/false">
3   <description>Description of the metric.</description>
4   <'metric definition' ...>
5 </metric>
```

Código A.4: Estrutura da Definição de Métricas em *XML*.

A *tag metric* do Código A.4 define os seguintes atributos:

- `Linha 1` : O atributo *name= "metricname"* define o nome das métricas, o *domain= "metricdomain"* representa o domínio para qual as métricas estão sendo definidas (*e.g.,* pacote, classe e componente) e o atributo *category= "metriccategory"* descreve as propriedades estruturais das métricas (*e.g.,* tamanho, acoplamento, coesão). A categoria é um atributo de documentação, pois não tem impacto no cálculo das métricas;

- `Linha 2` : Se o atributo for definido como *internal="true"* significa que a métrica é auxiliar, indicando que ela é utilizada na composição de outras métricas; caso contrário, o valor padrão do atributo é "false";

- `Linha 2` : É definido os subtipos do domínio que herdem a definição da métrica (*inheritable="true"*), ou não (*inheritable="false"*);

- `Linha 3` : Descrição da métrica; e,

- `Linha 4` : Define a forma de medir os modelos (*Projection, Compound Metrics, Attribute Value* e entre outros). Por exemplo, "<projection relation="context"/>" é utilizado para métricas que são computadas por meio de projeções e, nesse caso, sobre os elementos com o contexto definido na *tag domain*, ou seja, computa todos os elementos do modelo que possuem atributos fazendo referências para *"context"*.

No início do arquivo de definição de métricas tem-se um conjunto de elementos que indicam os aspectos gerais das métricas, tais como a indicação da versão compatível do *Metamodel Definition* e o verificador de regras para acessar os valores do *XMI*. Esse arquivo contém o conjunto de elementos, o conjunto de valores, as regras de projetos, a lista de palavras e as matrizes de relação.

As métricas devem ter nomes exclusivos para cada domínio. Em razão disso, por exemplo, no Código A.5 não é possível ter duas métricas chamada *"NumPriAttr"* para

classes. Porém, é possível ter métricas com nomes iguais para diferentes domínios, tal como uma métrica com nome *"NumPriAttr"* para classe e uma métrica *"NumPriAttr"* para interfaces.

No Código A.5 apresenta-se dois exemplos de métricas. A métrica *"NumPriAttr"* é responsável por contar o número de atributos privados em uma classe e a métrica *"NumPolyMeth"* para contar o número de métodos polimórficos de uma classe.

```
1  <metric name="NumPriAttr" domain="class" category="size">
2  <description>
3    The number of Private Attribute in a class.
4  </description>
5  <projection relset = "ownedattributes" target = "property" condition = "
       association='' and visibility='private'"/>
6  </metric>
7
8  <metric name="NumPolyMeth" domain="class" category="size">
9  <description>
10   The number of Polymorphic Methods in a class.
11 </description>
12 <projection relset = "ownedoperations" condition="name startswith 'virtual'"/>
13 </metric>
```

Código A.5: Métricas *"NumPriAttr"* e *"NumPolyMeth"* em *XML*.

Após a definição das informações das métricas é definido o processo de cálculo. Na métrica *"NumPriAttr"*, `Linha 5`, utiliza-se a expressão *<projection relset = "ownedattributes" target = "property" condition = "association" and visibility="private"/>*. Nesse exemplo, realiza-se uma operação de projeção sobre o conjunto de atributos da classe. A operação utiliza as propriedades dos elementos (*target = "property"*) com as condições de associação (*condition = "association"*) e visibilidade privada (*visibility="private"*).

Na `linha 12` é definido o processo de cálculo para a métricas *"NumPolyMeth"*, na qual realiza-se a projeção sobre o conjunto de operações/métodos (*projection relset = "ownedoperations"*) das classes com prefixo "virtual" (*condition="name startswith 'virtual' "*).

Apêndice B

Análises das Linguagens e Definição das Métricas

Neste apêndice são apresentados os resultados das análises das linguagens realizadas durante a execução do processo de avaliação com base no QM^2, e no modelo de qualidade definido por Ahrens et al. (2011). Em seguida, descreve-se os subatributos, assim como a identificação dos elementos mensuráveis para o cálculo dos atributos. Os atributos A1 e A7 são descritos no Capítulo 5.

B.1 Compreensibilidade de Componente – A2

B.1.1 Definição das Métricas do Atributo A2

A *Compreensibilidade de Componentes (A2)* depende de dois aspectos: (i) do indicador da quantidade de interfaces que são usadas, e (ii) a complexidade interna do componente. Neste contexto interfaces podem ser consideradas todas as mensagens e parâmetros. Para calcular o valor desse atributo considera-se três fatores: (i) todas as mensagens envidadas e recebidas são somadas, para então descobrir o *Fator da Quantidade de Mensagens* (MQF, do inglês, *Message Quantity Factor*); (ii) soma-se todos os parâmetros envolvidos para descobrir o *Fator da Quantidade de Parâmetros* (PQF, do inglês, *Parameter Quantity Factor*), sendo que, os módulos são afetados negativamente mais pelos parâmetros do que pelas as mensagens; e, (iii) estabelecer o *Fator de Complexidade do Módulo* (MCF, do inglês, *Module Complexity Factor*) por meio da experiência dos arquitetos de software e *stakeholders*.

Esse subatributo é calculado de acordo com a Equação B.1, fornecendo um valor que representa a facilidade de compreensão de componentes.

$$A2_i = \frac{\sum_{i=1}^{n}((MQF_i + PQF_i) * MCF_i)}{n} \tag{B.1}$$

Onde:

n = Quantidade de módulos i;

MQF_i = Fator da Quantidade de Mensagens no módulo i;

PQF_i = Fator da Quantidade de Parâmetros no módulo i;

MCF_i = Fator de Complexidade de Módulo no módulo i;

Na Tabela B.1 apresenta-se os fatores a serem considerados para o cálculo da compreensibilidade de componentes.

Tabela B.1: Fator de Complexidade Interna dos Componentes.

# Mensagens	MQF	# Parâmetros	PQF	Complexidade Interna	MCF
0–3	0	0–10	0	nenhum	0
4–10	1	11–25	1	fácil	1
11–20	2	26–50	2	médio	2
21–40	5	51–100	5	alto	3
>40	10	>100	10		

O valor total da métrica é calculado com base na média das complexidades estruturais de todos os módulos. As métricas atuam no nível do módulo como menores unidades distribuíveis.

B.1.2 Resultados das Análises da Linguagem

Na Tabela B.2, apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de Compreensibilidade de Componente (A2)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Os elementos quantificáveis para a avaliação desse atributo são definidos como módulos, mensagens, parâmetros e complexidade interna, atuando principalmente nas classes. Ou seja, as métricas são, em termos práticos, a contagem dos elementos de interação entre as classes. Consequentemente, deve-se extrair as métricas diretamente dos diagramas que contêm esses elementos.

Tabela B.2: Análise do Subatributo de Compreensibilidade de Componente.

Subatributo de Compreensibilidade de Componente – A2			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Módulos	Classes	–	Sequência Comunicação
Mensagens enviadas	Classes (métodos chamados de uma classe de outra classe)	–	
Mensagens Recebidas	Classes (chamada a métodos de uma classe de outra classe)	–	
Parâmetros das mensagens	Classes (quantidade de métodos chamados e chamada de métodos)	–	
Complexidade Interna	Complexidade de McCabe, Normalizado no intervalo [0,3]	–	

* Da granularidade alta para a refinada.

B.2 Mutabilidade – A3

B.2.1 Definição das Métricas do Atributo A3

O atributo *Mutabilidade (A3)* trata a facilidade de mudança em uma arquitetura de software, essas alterações podem impactar sobre o restante do projeto. Portanto, mutabilidade e estabilidade estão fortemente relacionados. A mutabilidade interna dos módulos ou componentes não pode conter informações que possam ser acessadas diretamente, então é necessário a declaração de interfaces para os módulos. A complexidade dos módulos dependentes é a probabilidade de alteração das interfaces.

Para o cálculo desse atributo utiliza-se três fatores: (i) O *Fator de Complexidade do Módulo* (MCF, do inglês, *Module Complexity Factor*); (ii) O *Fator de uso da Interface* (IUF, do inglês, *Interface Use Factor*), pois uma interface pode ser utilizada em toda a arquitetura; e, (iii) o *Fator do Tipo de Dados* (DTF, do inglês, *Data Type Factor*) que é compreendido com base em uma abordagem estatística, isso reflete na probabilidade de alteração das interfaces do tipo enumeradas ser mais alta que as não enumeradas.

Esse subatributo é calculado de acordo com a Equação B.3, fornecendo um valor que representa a facilidade de alteração.

$$A3_i = \frac{\sum_{i=1}^{n}(DTF_i * IUF_i * MFC_i)}{n} \tag{B.2}$$

$$MCF_i = \frac{\sum_{j=1}^{l_i} MCF_j}{l_i} \tag{B.3}$$

Onde:

n = Quantidade Total de Interfaces;

l_i = Quantidade Total de módulos conectados na interface i;

DTF_i = Fator do Tipo de Dados da interface de saída i;

IUF_i = Fator de Uso da Interface de saída i;

MCF_i = Fator de Complexidade de Módulo no módulo i;

MCF_i = Média dos MCF_j utilizados na interface de saída i com entrada.

Na Tabela B.3, apresenta-se os fatores a serem considerados para o cálculo do atributo mutabilidade.

Tabela B.3: Fator de Mutabilidade

# interfaces	IUF	Tipo de Dados	DTF
1	0	Não enumerado	1
2	1	enumerado	4
3–4	2		–
5–10	5		–
>10	20		–

O uso frequente de uma interface é avaliado negativamente pelo IUF, pois no caso de alteração da especificação de uma interface vários módulos devem ser alterados. Portanto, o produto de IUF, DTF e a média MCF de todos os módulos ligados são calculados para cada elemento da interface e depois calcula-se a média.

B.2.2 Resultados das Análises da Linguagem

Na Tabela B.4 apresentam-se os resultados da análise realizada para a definição das métricas internas para o *subatributo de mutabilidade (A3).*

Os elementos quantificáveis para a avaliação desse atributo são definidos como interfaces dos módulos (classes e componentes), interfaces providas, interfaces recebidas e a complexidade interna, encontrados nos diagramas de sequência, comunicação, componentes e classes. Ou seja, as métricas são o cálculo dos elementos de interação entre as

Tabela B.4: Análise do Subatributo de Mutabilidade.

Subatributo de Mutabilidade (A3)			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Interfaces	Classes, interfaces providas	–	Sequência Comunicação Componentes Classe
Módulos conectados na interface	Classe, interfaces	–	
Interfaces de Saída Usada	Chamada a métodos de uma classes	–	
Tipo de Dados de saída da interface	Classes (Avaliar o tipo de dados da interfaces)	–	
Complexidade Interna	Complexidade de McCabe, Normalizado no intervalo [0,3]	–	

* Da granularidade alta para a refinada.

classes. Consequentemente, deve-se extrair as métricas diretamente dos diagramas que contêm esses elementos.

B.3 *Estabilidade* – A4

B.3.1 Definição das Métricas do subatributo A4

O subatributo *Estabilidade (A4)* refere-se a frequência de alteração necessária em um módulo devido a influências de outros módulos. Basicamente, isso é a alteração de interfaces, e é por isso que módulos com muitas interfaces de entrada são mais vulneráveis a alteração do que os com baixo número de interfaces.

Para o cálculo desse subatributo são considerados todos os valores das interfaces de entrada e saída, incluindo, as trocas de mensagens, parâmetros de calibragem e codificação. Devido a isso, a probabilidade de alteração de uma interface é reusada (DTF) do Subatributo A3.

Esse subatributo é calculado de acordo com a Equação B.4, fornecendo um valor que representa a facilidade de alteração das interfaces de entradas.

$$A4_i = \frac{\sum_{i=1}^{n} \frac{\sum DTF_{in,i}}{\sum DTF_{in,i} + \sum DTF_{out,i}}}{n} \tag{B.4}$$

Onde:

n = Quantidade de módulos i;

$DTF_{in,i}$ = Somatório dos fatores do tipo de dados para todas as interfaces de entrada no módulo i;

$DTF_{out,i}$ = Somatório dos fatores do tipo de dados para todas as interfaces de saída no módulo i;

O cálculo representa a relação da quantidade de interfaces de entrada e saída, e o resultado é um percentual de estabilidade. O intervalo de valores é de 0..1, e quanto mais próximos de zero mais estável é o módulo. O valor ideal pode ser obtido quando não existe interfaces de entradas.

B.3.2 Resultados das Análises da Linguagem

Na Tabela B.5, apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de Estabilidade (A4)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Tabela B.5: Análise do Subatributo Estabilidade.

Subatributo Estabilidade (A4)			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Módulos	Classes	–	Sequência Comunicação Classe
Interfaces de entrada e saída do módulo	Interfaces, interfaces providas e requiridas	–	
Tipo de dados das interfaces (providas e requiridas) utilizadas	Chamada a métodos de uma classe	–	
Tipo de Dados de entrada e saída da interface	interfaces *(enum ou not enum)*	–	

* Da granularidade alta para a refinada.

Os elementos mensuráveis para a avaliação desse subatributo são definidos como interfaces providas e requiridas das classes e componentes. Esses elementos estão presentes nos diagramas de sequência, comunicação, componentes e classes. As métricas internas são calculadas com base nos elementos de interação entre as classes. Portanto, deve-se extrair as métricas diretamente dos diagramas que contêm as interações.

B.4 *Testabilidade* – A5

B.4.1 Definição das Métricas do Subatributo A5

O subatributo *testabilidade (A5)* concentra-se em testar as partes isoladas do software, isto é, testes dos módulos. A capacidade de testar um módulo depende, principalmente, da quantidade do tipo de dados associados as interfaces de entrada e da complexidade interna.

O cálculo é realizado considerando as interfaces de entrada, como mensagens e parâmetros de codificação, e são aplicados para obter o *Fator da Quantidade de entradas* (IQF, do inglês, *Input Quantity Factor*). Na Tabela B.6 apresenta-se os fatores a serem considerados para o cálculo do subatributo.

Tabela B.6: Fator de quantidade de entradas.

# Entrada (Mensagens e Parâmetros)	IQF
0–10	1
11–20	3
21–40	10
>40	20

Esse subatributo é calculado de acordo com a Equação B.5, fornecendo um valor que representa a facilidade de teste dos módulos.

$$A5_i = \frac{\sum_{i=1}^{n}(MCF_i * IQF_i * DTF_i)}{n} \tag{B.5}$$

$$DTF_i = \frac{\sum_{j=1}^{m_i} DTF_j}{m_i} \tag{B.6}$$

Onde:

n = Quantidade Total de módulos;

m_i = Quantidade Total de interfaces de entrada no módulo atual i;

MCF_i = Fator de Complexidade de Módulo no módulo i;

DTF_j = Fator do Tipo de Dados da interface de entrada j;

DTF_i = Média do Fator do Tipo de Dados de todas as interfaces de entrada do módulo i;

IQF_i = Fator da quantidade de módulos i;

A facilidade de testar é proporcional à quantidade de casos de teste independentes, e está relacionada com a quantidade de interfaces de entrada e seus possíveis valores. O tipo de dado enumerado tem relevância, pois os valores de entrada são utilizados para desvios condicionais e todos os possíveis valores representam um único caminho/fluxo no software. Portanto, todos os fluxos devem ser testados e isso aumenta o esforço de teste consideravelmente.

O valor obtido da média DTF – para todas as interfaces de entrada – é calculado para cada módulo e multiplicado pelos IQF e MCF, e por fim é incluído globalmente a média de testabilidade de todos os módulos.

B.4.2 Resultados das Análises da Linguagem

Na Tabela B.7, apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de testabilidade (A5)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Tabela B.7: Análise do Subatributo Testabilidade.

Subatributo Testabilidade (A5)			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Módulos	Classes	–	Sequência Comunicação Classe
Tipo de dados das interfaces (providas e requiridas) utilizadas	interfaces *(enum ou not enum)*	–	
Complexidade Interna	Complexidade de McCabe, Normalizado no intervalo [0,3]	–	

* Da granularidade alta para a refinada.

Os elementos contáveis para a avaliação desse subatributo são definidos como interfaces providas e requiridas das classes e componentes. Esses elementos estão presentes nos diagramas de sequência, comunicação e classes. As métricas internas são calculadas com base nos elementos de interação entre as classes.

B.5 *Escalabilidade* – A6

B.5.1 Definição das Métricas do subatributo A6

O subatributo *Escalabilidade (A6)* baseia-se em medir o quanto é "fácil" extrair ou remover as partes de um software. Isto é principalmente determinado pela dependência dos módulos. Portanto, para remover um módulo que é ligado pelas interfaces com vários outros módulos tem-se um impacto maior do que remover um com poucas ligações.

Para avaliar a dependência é utilizado o *Fator de Uso do Módulo* (MUF, do inglês, *Module Use Factor*). As dependências são geradas pelo uso de parâmetros de saída e mensagens entre os módulos. Na Tabela B.8 apresenta-se os fatores a serem considerados para o cálculo do subatributo.

Tabela B.8: Fator de Uso do Módulo.

# dependências	MUF
0	0
1	1
2	3
3–4	7
5–10	20
>10	50

Esse subatributo é calculado de acordo com a Equação B.7, fornecendo um valor que representa a dependência entre os módulos do software.

$$A6_i = \frac{\sum_{i=1}^{n} MUF_i}{n} \tag{B.7}$$

Onde:

n = Quantidade Total de módulos;

MUF_i = Fator de Uso do Módulo no módulo i;

A escalabilidade é calculada considerando a média do MUF para todos os módulos. Nota-se que MUF aumenta de maneira mais agressiva, isso acontece porque os módulos com uma ou duas dependências são mais facilmente removidos que, por exemplo, um com cinco ou mais.

B.5.2 Resultados das Análises da Linguagem

Na Tabela B.9, apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de escalabilidade (A6)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Tabela B.9: Análise do Subatributo Escalabilidade.

Subatributo Escalabilidade – A6			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Módulos	Classes	–	Sequência
Interfaces de Saída (usadas)	Classes, Interfaces	–	Comunicação

* Da granularidade alta para a refinada.

Os elementos contáveis para a avaliação desse subatributo são definidos como quantidade de módulos e interfaces de saída usadas por outros módulos (classes e suas interfaces). Esses elementos estão presentes nos diagramas de sequência, comunicação e classes. As métricas internas são calculadas com base nos elementos de interação entre as classes.

B.6 *Eficiência de Recursos* – A8

B.6.1 Definição das Métricas do subatributo A8

O subatributo *Eficiência de Recursos (A8)*, em geral, somente pode ser calculado e comparado com medições reais de uma sistema. Esta avaliação é complicada de ser executar em nível arquitetônico. Portanto, utiliza-se uma alternativa que aborda a maturidade do projeto, pois de acordo com ela pode-se obter informação para o cálculo.

Nas fases iniciais dos projetos somente a comunicação externa dos módulos estão disponíveis, por isso, utiliza-se as interações entre os módulos por meio de mensagens para o cálculo. Além disso, considera-se os parâmetros e o uso de variáveis locais. A Equação B.8 fornece um valor que representa o grau para a eficiência dos recursos.

$$A8 = \sum_{j=1}^{n} \left(\sum_{j=1}^{a} S_{Mess_j} + \sum_{j=1}^{b} S_{Para_j} + \sum_{j=1}^{c} S_{Loc_j} \right) \tag{B.8}$$

Onde:

n = Quantidade Total de módulos;

a = Quantidade de Mensagens no módulo j;

b = Quantidade de Parâmetros no módulo j;

c = Quantidade de Variáveis Locais no módulo j;

S_{Mess_j} = Tamanho equivalente da mensagem corrente;

S_{Para_j} = Tamanho equivalente dos Parâmetros corrente;

S_{Loc_j} = Tamanho equivalente da Variável Local;

B.6.2 Resultados das Análises da Linguagem

Na Tabela B.10, apresentam-se os resultados da análise realizada para a definição das métricas internas para o *Subatributo de Eficiência de Recursos (A8)*, que compõem o conjunto do atributo de qualidade *"Mutabilidade"*.

Tabela B.10: Análise do Subatributo Eficiência de Recursos.

Subatributo Eficiência de Recursos – A8			
Original (AHRENS et al.)	**UML**		
Elementos Contáveis	**Elementos***	**Inclui**	**Diagramas**
Módulos	Classes	–	Sequência Comunicação Classe Pacotes Componentes
Interfaces de entrada e saída do módulo	Interfaces, interfaces providas e requiridas	–	
Parâmetros e mensagens de entrada e saída	Parâmetros nas chamadas de métodos	–	
Variáveis Locais do Módulo	Atributos das classes	–	

* Da granularidade alta para a refinada.

Para avaliar esse atributo utiliza-se os elementos contáveis que são definidos como quantidade de módulos, parâmetros e mensagens trocadas entre os módulos. Esses elementos estão contidos nos diagramas de sequência, comunicação, classe, pacote e componentes. Portanto as métricas internas são definidas sobre esses elementos.