5
Abordagem

Este capítulo explicita todos os passos expostos na seção 1.2.2. O estudo proposto foi pensado para o desenvolvimento de uma ferramenta web genérica capaz de ler diagramas de classe, transformá-los em grafos orientados e, sobre eles, aplicar métricas e algoritmos que apresentem informações adicionais importantes no processo de desenvolvimento de software.

A seguir, serão descritas todas as informações geradas pela ferramenta:

- *Grafo orientado:* Grafo cujos nós são classes e as arestas são os relacionamentos UML entre as classes. Para cada tipo de relacionamento será atribuída uma distância, que representa o peso da aresta. Este grafo é extraído do leitor do arquivo de exportação do diagrama de classes.
- *Core Concepts:* Indicador do grau de importância de uma determinada classe. Quanto maior o *core concept*, maior a importância de uma classe dentro do diagrama de classes.
- *Componentes:* Grupos de classes que estão mais próximas dentro do modelo. Cada grupo contém um erro residual, que denota o quão próximas estão as classes dentro dele.
- *Estimativa de complexidade:* Valor estimado da dificuldade de se implementar um projeto ou um componente específico dentro do projeto.
- *Estimativa de coesão:* Valor estimado da coesão dos componentes de um projeto ou de um componente específico.
- *Métricas de tamanho:* Métricas que demonstram o tamanho do diagrama de classes, como número total de classes e número total de relacionamentos.
- *Métricas de acoplamento:* Métricas que demonstram o grau de instabilidade dos grupos de classe. Ou seja, o quanto um componente depende de outros componentes.

- *Atribuições de classes a desenvolvedores:* A ferramenta dá a oportunidade ao gerente de atribuir classes a desenvolvedores dentro do projeto.

Cada informação será mais detalhada em seções específicas dentro deste capítulo.

5.1. Grafo orientado

Esta fase do projeto consiste em ler o arquivo de exportação padrão de diagramas de classes, conhecido como XMI, e gerar um grafo orientado ilustrado pela Figura 10:

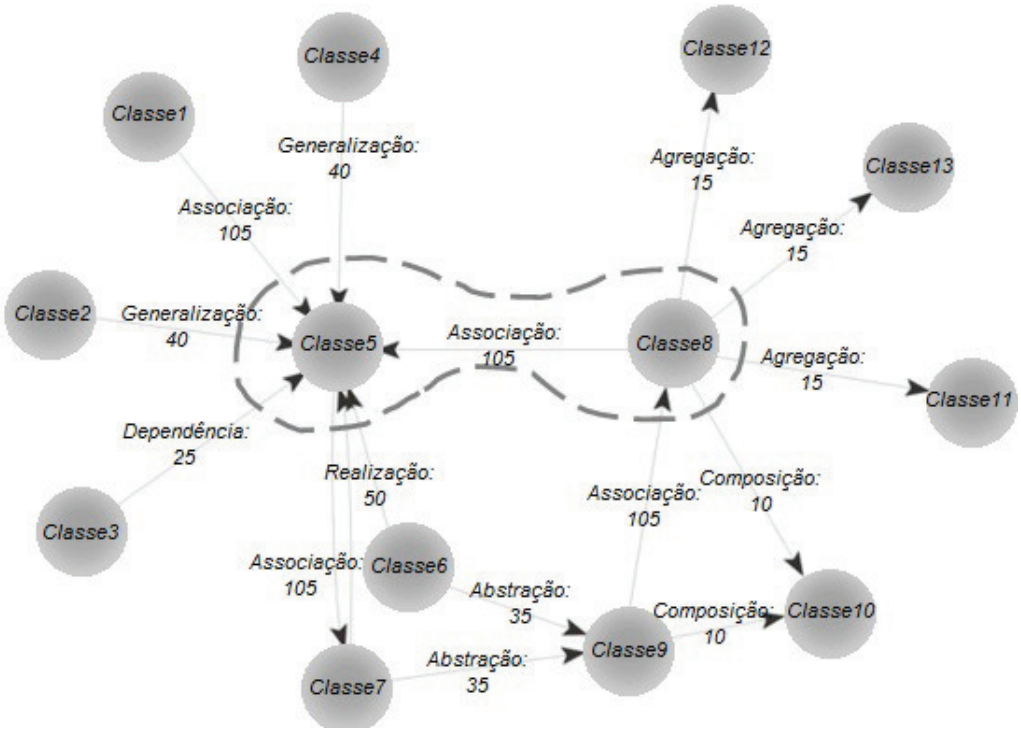

Figura 10 - Grafo orientado gerado na leitura do diagrama representado pelo XMI.

Observe que as arestas são representadas pelos relacionamentos entre classes. Cada aresta contém o tipo de relacionamento e o peso para cada tipo. No grafo acima, por exemplo, uma associação entre classes representa um peso de aresta no valor *105*.

Foi dado suporte para todos os relacionamentos explicados na seção 4.1.1. Os relacionamentos são: abstração, generalização, dependência, realização, associação, composição e agregação.

No processo de geração do grafo, a atividade mais importante é a leitura do arquivo XMI. A próxima sub-seção descreverá como é a técnica de leitura deste arquivo.

5.1.1. Parser Genérico do XMI

Cada ferramenta de edição de diagramas de classe UML tem um dicionário diferente para exportação do modelo. Isto é um problema para quem faz a importação, pois aumenta a dificuldade de se desenvolver um *parser* genérico de leitura do arquivo de exportação.

O lado bom é que existe um padrão no formato do arquivo de exportação. Este padrão é o XMI e o dicionário de cada ferramenta representa o *Schema* dos arquivos XMI. Os *Schemas* descrevem como o metamodelo dos diagramas é exibido no arquivo.

Os arquivos XMI possuem a estrutura de um arquivo XML. Para fazer o leitor deste tipo de arquivo, é necessário, primeiramente, implementar o *parser* de arquivos XML. Depois, estuda-se a sintaxe e a semântica de cada *tag* do XML contida no XMI.

5.1.1.1. Parser XML

As duas principais técnicas de *parser* XML são o *SAX*[12] *Parser* e o *DOM*[13]*Parser*.

De acordo com Allu (2012), o *SAX Parser* não cria nenhuma estrutura interna. As ocorrências de cada entrada do documento são dadas como eventos, cabe a ferramenta tratar os eventos criados. A vantagem do *SAX Parser* é a eficiência de processamento, rapidez e espaço de memória para entradas grandes de dados.

Ainda segundo Allu (2012), o *DOM Parser* cria uma estrutura de árvore em memória a partir de um documento de entrada e guarda as ocorrências de cada evento como um nó. Quando o documento é grande, esta técnica é pouco útil em

[12] Simple API for XML
[13] Document Object Model

virtude do grande espaço de memória ocupada pela árvore e pelo lento processamento de *parser*.

Foi escolhido o *SAX Parser* dentre as duas técnicas, pois os documentos de entrada XMI são geralmente arquivos de grande porte. Assim, a estrutura da leitura e o tratamento dos eventos de cada *tag* XML terão que ser desenvolvidos.

Com posse do *parser* XML, chega-se ao ponto do estudo dos diferentes dicionários XMI e de suas respectivas *tags* XML.

Para implementar o *parser* genérico do modelo, o XMI foi quebrado em três arquivos:

- Arquivo XMI
- Arquivo de Metamodelo
- Arquivo de Transformação

5.1.1.2. Arquivo XMI

É o arquivo exportado pelas ferramentas de criação e edição de diagramas UML. No XMI, se encontram *tags* que representam as classes e os seus respectivos atributos, métodos, pacotes e estereótipos, bem como os diferentes tipos de relacionamento entre as classes.

A Figura 11 apresenta um trecho do arquivo XMI exportado pela ferramenta *Astah Professional 6.5*:

```
<UML:Class                                    xmi.id="me-gvpfmmdz-2tj5tv--n12rh5-
45293dd5f3a7ce872896d0760330f29c"        name="MinhaClasse1"        version="0"
unSolvedFlag="false"    isRoot="false"    isLeaf="false"    isAbstract="false"
isActive="false">
      <UML:ModelElement.namespace>
            <UML:Namespace            xmi.idref="fq-gvpfmmdz-2tj5tv--n12rh5-
45293dd5f3a7ce872896d0760330f29c"/>
      </UML:ModelElement.namespace>
      <UML:ModelElement.visibility xmi.value="public"/>
       <UML:ModelElement.clientDependency>
      <JUDE:Dependency                xmi.idref="1fq7-gvpfmmdz-2tj5tv--n12rh5-
45293dd5f3a7ce872896d0760330f29c"
xmlns:JUDE="http://objectclub.esm.co.jp/Jude/namespace/"/>
      </UML:ModelElement.clientDependency>
      <UML:GeneralizableElement.generalization>
      <UML:Generalization              xmi.idref="6d6-gvpfmmdz-2tj5tv--n12rh5-
45293dd5f3a7ce872896d0760330f29c"/>
      </UML:GeneralizableElement.generalization>
</UML:Class>
```

Figura 11 - Trecho de código do arquivo XMI.

Este trecho de arquivo XMI representa a classe de nome *MinhaClasse1*, não abstrata, definida na *tag <UML:Class>*. Repare que o atributo *xmi.id* contém o identificador único desta classe no contexto do documento.

Dentro da classe, a *tag <UML:ModelElement.namespace>* define o contexto sob a qual a classe se insere, através do atributo *xmi.idref*. A visibilidade é extraída pela *tag <UML:ModelElement.visibility>*.

A classe *MinhaClasse1* tem um relacionamento de dependência e outro de generalização com outras classes. O relacionamento de dependência é representado pela *tag <UML:ModelElement.clientDependency>*, onde a origem é a própria classe em questão e o destino é a classe cujo identificador é atribuído ao *xmi.idref* da *tag <JUDE:Dependency>*. A *tag <UML:GeneralizableElement.generalization>* explicita um relacionamento de geralização, cuja origem é *MinhaClasse1* e o destino é a classe cujo identificador é atribuído ao *xmi.idref* da *tag <UML:Generalization>*.

Vale salientar que as *tags <UML:ModelElement.namespace>, <UML:ModelElement.visibility>, <UML:ModelElement.clientDependency>, <JUDE:Dependency>* e *<UML:Generalization>* têm nomes diferentes em cada ferramenta de criação e edição de diagramas de classe. Por isso, é necessário estudar o dicionário de *tags*, descritos no *XMI Schema*, que evidenciam as sintaxes e as semânticas de todas as *tags* do documento.

5.1.1.3. Arquivo de Metamodelo

Arquivo que define o metamodelo das *tags* XML das quais a ferramenta *AnalisadorUML* é capaz de ler. Em outras palavras, é o *XMI Schema* utilizado como base de *parser*.

É possível definir variados arquivos de metamodelo dentro da ferramenta. Estes arquivos servem de entrada, assim como o arquivo XMI e o arquivo de transformação, para a leitura do modelo. Ou seja, estes três arquivos são lidos em conjunto e variam de acordo com a ferramenta de criação e edição de diagramas utilizada.

Um trecho do arquivo de metamodelo para a ferramenta *Astah Professional 6.5* é exibido na Figura 12:

```
<modelelement name="meuElementoBase">
      <attribute name="contexto" type="ref">Pai deste elemento no modelo
UML.</attribute>
      <attribute name="identificador" type="data">Identificador unico do
elemento.</attribute>
      <attribute name="nome" type="data">Nome do elemento.</attribute>
      <attribute name="estereotipos" type="ref"
multiplicity="many">Estereotipos do elemento.</attribute>
</modelelement>
<modelelement name="classe">
      <attribute name="visibilidade" type="data">Visibilidade (public,
private, protected, package).</attribute>
      <attribute name="abstrato" type="data">Indica se a classe e
abstrata.</attribute>
      <attribute name="folha" type="data">Indica se a classe nao pode ter um
filho.</attribute>
</modelelement>
```

Figura 12 - Trecho de código do arquivo de metamodelo.

Este código denota duas estruturas de elemento denominadas *meuElementoBase* e *classe* dentro do metamodelo. Estas estruturas são definidas pela *tag* <*modelelement*>.

Dentro do contexto do elemento *meuElementoBase*, são definidos os atributos contexto, identificador, nome e estereótipos. O atributo contexto é dado por referência do identificador do elemento pai, o atributo identificador é dado diretamente sem referência, assim como o atributo nome. Já o atributo estereótipos é dado por referência de identificador do estereótipo e pode receber múltiplas referências.

Dentro do contexto do elemento *classe*, são definidos os atributos visibilidade, abstrato e folha. Todos estes atributos são dados diretamente sem referência.

Como *meuElementoBase* serve de base para os demais elementos, *classe* herda todos os atributos de *meuElementoBase*.

5.1.1.4. Arquivo de transformação

Arquivo responsável por ligar o XMI com o arquivo de metamodelo, serve como um adaptador. Este arquivo possibilita o uso de qualquer XMI e de qualquer metamodelo.

A Figura 13 apresenta um trecho de código do arquivo de transformação XMI-Metamodelo:

```
<xmitransformation modelelement="meuElementoBase" xmipattern="meuElementoBase"
recurse="true">
      <trigger name="identificador" type="attrval" attr="xmi.id" />
      <trigger name="nome" type="attrval" attr="name" />
      <trigger name="nome" type="ctext" src="UML:ModelElement.name" />
      <trigger name="contexto" type="attrval" attr="namespace"/>
      <trigger                name="contexto"                type="gcattrval"
src="UML:ModelElement.namespace" attr="xmi.idref"/>
      <trigger              name="estereotipos"              type="gcattrval"
src="UML:ModelElement.stereotype"
                          attr="xmi.idref" linkbackattr="extendedelements" />
</xmitransformation>
<xmitransformation        modelelement="classe"        xmipattern="UML:Class"
recurse="true">
      <trigger name="visibilidade" type="attrval" attr="visibility" />
      <trigger               name="visibilidade"               type="cattrval"
src="UML:ModelElement.visibility" attr="xmi.value"/>
      <trigger name="abstrato" type="attrval" attr="isAbstract"/>
      <trigger                name="abstrato"                type="cattrval"
src="UML:GeneralizableElement.isAbstract" attr="xmi.value"/>
      <trigger name="folha" type="attrval" attr="isLeaf"/>
      <trigger                 name="folha"                 type="cattrval"
src="UML:GeneralizableElement.isLeaf" attr="xmi.value"/>
</xmitransformation>
```

Figura 13 - Trecho de código do arquivo XMI-Metamodelo.

O propósito é realizar um de-para das *tags* encontradas no documento do metamodelo para as *tags* do XMI. A *tag xmitransformation* transpõe o valor da *tag meuElementoBase* (atributo *xmipattern*), retirado do XMI, para o valor contido na *tag meuElementoBase* (atributo *modelelement*) do arquivo de metamodelo.

A mesma ideia vale para a *tag classe* do arquivo de metamodelo, que conterá o valor da *tag UML:Class* do arquivo XMI. Dentro da *tag xmitransformation* são feitos os adaptadores dos atributos de cada elemento passados por parâmetro (*cattrval* ou *gcattrval*) ou por valor (*attrval* ou *ctext*).

É necessário o preenchimento dos adaptadores de todos os elementos e atributos contidos no arquivo de metamodelo para as respectivas *tags* encontradas no arquivo XMI.

5.1.1.5. Leitura do Arquivo no *AnalisadorUML*

A ferramenta *AnalisadorUML* define cada modelo de diagrama de classes em um projeto. Ao criar um projeto, a ferramenta solicita como entrada o arquivo XMI, o arquivo de metamodelo e o arquivo de transformação.

Foram pré-cadastrados elementos na ferramenta, com os seus respectivos atributos, que definem o que, de fato, o *AnalisadorUML* é capaz de extrair dos modelos de diagramas de classe UML. Estes elementos são:

- *Classe:* Contexto, identificador, nome, estereótipos, visibilidade, abstrato, folha.
- *Interface:* Contexto, identificador, nome, estereótipos.
- *TipoDado:* Contexto, identificador, nome, estereótipos.
- *Atributo:* Contexto, identificador, nome, estereótipos, visibilidade, tipoAtributo, editável.
- *Operação:* Contexto, identificador, nome, estereótipos, visibilidade, abstrata, eConsulta.
- *Parâmetro:* Contexto, identificador, nome, estereótipos, tipo, tipoParâmetro.
- *Método:* Contexto, identificador, nome, estereótipos.
- *Modelo:* Contexto, identificador, nome, estereótipos.
- *Pacote:* Contexto, identificador, nome, estereótipos.
- *Subsistema:* Contexto, identificador, nome, estereótipos.
- *Associação:* Contexto, identificador, nome, estereótipos.
- *ClasseDeAssociação:* Contexto, identificador, nome, estereótipos.
- *TerminaisAssociação:* Contexto, identificador, nome, estereótipos, navegável, agregação, tipoAssociação, owner.
- *Generalização:* Contexto, identificador, nome, estereótipos, filho, pai.
- *Abstração:* Contexto, identificador, nome, estereótipos, absServidor, absCliente.
- *Dependência:* Contexto, identificador, nome, estereótipos, depServidor, depCliente.
- *Realização:* Contexto, identificador, nome, estereótipos, reaServidor, reaCliente.
- *Estereótipo:* Contexto, identificador, nome, estereótipos, elementos.
- *Diagrama:* Contexto, identificador, nome, estereótipos, tipo.
- *ElementoDiagrama:* Contexto, identificador, nome, estereótipos, elemento.

Repare que os atributos *contexto*, *identificador*, *nome*, *estereótipos* são repetidos em todos os elementos. Por isso, foi criado o *meuElementoBase* no arquivo de metamodelo, que contém exatamente estes atributos.

Com o projeto criado, o usuário define o de-para de cada elemento pré-cadastrado citado acima para o elemento contido no arquivo de metamodelo. A ferramenta já vem com os valores padrões do adaptador do metamodelo UML1.1, utilizada no *Astah Professional 6.5*.

Caso o usuário resolva utilizar outro metamodelo, a definição do de-para da ferramenta *AnalisadorUML* para o arquivo de metamodelo deverá ser refeita em tela.

Desta maneira, o *parser* dos arquivos de exportação XMI é feito por adaptadores em dois níveis. O primeiro nível é feito pelo arquivo de transformação, que transforma um arquivo XMI no formato de um arquivo de metamodelo bem definido. O segundo nível é feito pelo usuário na criação do projeto, através de um adaptador dos elementos e atributos encontrados no arquivo de metamodelo para elementos e atributos que a ferramenta é capaz de ler.

Estes dois níveis de adaptação tornam a leitura do XMI genérica, visto que é possível customizar a transformação do arquivo XMI em um arquivo de metamodelo e, também, é possível customizar a transformação do arquivo de metamodelo para os elementos de leitura da ferramenta.

Vale salientar que os elementos pré-cadastrados da ferramenta são também customizáveis: o usuário pode alterar, excluir e adicionar elementos e atributos de leitura do *AnalisadorUML*.

Para desenvolver as referências contidas nos atributos *xmi.idref*, foram desenvolvidas duas estruturas de hash. A estrutura de hash, basicamente, contém uma chave e um objeto relacionado à chave. A primeira estrutura tem como chave os atributos *xmi.id* ou *xmi.uuid* e como objeto, o elemento extraído do XMI. A segunda estrutura tem como chave o nome do elemento adicionado do contexto sob o qual está inserido e como objeto, os atributos *xmi.id* ou *xmi.uuid*. Deste modo, quando uma referência é descrita pelo nome do elemento, a segunda estrutura de hash é utilizada para obter o identificador do elemento e, então, a primeira estrutura é utilizada para obter o elemento em si do XMI. Quando uma referência é dada pelo identificador, só a primeira estrutura de hash é utilizada.

O *parser* do XMI tem como resultado final um grafo orientado, modelado como uma lista de adjacências de classes. Esta estrutura gerada é persistida em banco de dados para uso posterior e verificação do modelo.

5.2. Core Concepts

A obtenção dos *core concepts* remonta a métrica *Caminhos* citada na seção 3.4. Esta métrica mede o número de vezes que um determinado nó está presente em um caminho mais curto de um nó ao outro no grafo, para todos os nós do grafo.

O objetivo aqui é determinar quais classes são mais importantes para o diagrama, estas classes são denominadas *core concepts*.

Os *core concepts* têm um significado gerencial fundamental, pois indicam quais as classes devem receber uma atenção especial. Desenvolvedores experientes precisam implementá-las e testes rigorosos são necessários para evitar futuros problemas de refatoração ou correção de erros em cascata.

Esta métrica é calculada depois da geração do grafo orientado pela leitura do XMI. A estrutura de lista de adjacências é remontada do banco de dados para a memória e, sobre ela, é aplicado o algoritmo de Floyd-Warshall (explicado na seção 4.3). O Floyd-Warshall, além de descrever os pesos dos caminhos mais curtos entre pares de todos os nós, também dirá qual a sequência de nós intermediários que descrevem estes caminhos.

Tendo *n* como número de nós do grafo, existem no total n^2 caminhos mais curtos de, no máximo, *n* nós. O próximo passo da metodologia é contar a presença de um determinado nó nos n^2caminhos mais curtos para todos os *n* nós do grafo.

As importâncias retiradas da contagem descrita acima são persistidas no banco de dados para cada classe do modelo. A ferramenta *AnalisadorUML* possui uma tela de exibição que ordena as classes por projeto criado e por sua importância.

5.3. Componentes

A componentização das classes consiste na identificação de grupos de classe que mais se relacionam entre si. Estas classes são alocadas em componentes e persistidas em banco de dados.

Do ponto de vista de processo de software, a componentização do diagrama de classes serve para extrair diversas informações ainda na fase de design. A primeira tange à divisão do trabalho. A equipe pode ser dividida em grupos, cada qual responsável pelo desenvolvimento de um ou mais componentes. Ainda

assim, se houver mais componentes do que grupos de equipe, métricas, a serem discutidas nas próximas seções, aplicadas na componentização revelam qual componente está mais próximo de outro componente para definir sequência de atividades e tarefas.

O algoritmo utilizado para a detecção dos componentes de classes foi o *k-means*, descrito na seção 4.4. O *k-means* necessita de alguns parâmetros de entrada, são eles:

- Matriz de distâncias de todos os pares de nós.
- Número de componentes a serem gerados.
- Número de iterações para o algoritmo convergir.

Para calcular a matriz de distâncias, aplica-se o algoritmo de Floyd-Warshall sob o grafo de classes orientado gerado na leitura do modelo. Note que, nesta etapa, não se está interessado no caminho que denota a sequência dos nós intermediários e, sim, no peso total do caminho. Desta forma, o Floyd-Warshall aqui utilizado calcula os pesos dos caminhos mais curtos entre todos os pares de nós do grafo e não guarda o caminho mais curto entre todos os pares de nós.

Os pesos encontrados dos caminhos mais curtos dos pares de nós se traduzirão nas distâncias entre estes pares. Assim, a matriz de distância de todos os pares é gerada.

Com relação ao número de componentes, esta informação é dada pelo usuário na criação do projeto de análise do modelo. Esta forma se mostrou mais apropriada porque o número de desenvolvedores de um determinado projeto varia também de acordo com a disponibilidade dos profissionais dentro do contexto da empresa. Além disto, é difícil a ferramenta mensurar genericamente o que seria complexo ou não para desenvolvedores. É melhor deixar que o usuário gerente disponibilize os dados, pois ele conhece especificamente os profissionais da sua empresa.

A detecção do número de iterações para o algoritmo convergir foi feita com base na experimentação de diferentes projetos. Vários números diferentes são testados para analisar o ponto de quando o algoritmo exibe o mesmo resultado. O número seis foi obtido através da média do critério de parada dos projetos. Vale dizer que quanto mais complexo for um projeto, maior será o número de iterações necessárias para a convergência.

5.4. Estimativas de complexidade

As estimativas de complexidade são somatórios realizados na saída do algoritmo da obtenção dos *core concepts*. A complexidade de um componente é o somatório das métricas de complexidade de todas as classes do componente, enquanto que a complexidade de um projeto é o somatório das métricas de complexidade de todas as classes contidas no modelo.

Estas estimativas são necessárias na precificação de um determinado projeto. A complexidade do projeto em questão é comparada à complexidade de projeto fechados, com custo definido. Desta maneira, o gerente obtém uma estimativa de preço de determinado modelo. O estudo de Podgorelec & Heric (2007) demonstrou que existe uma relação entre a complexidade de um modelo de diagrama de classes e o desenvolvimento de códigos Java no valor de 0.64. Deste modo, uma estimativa de complexidade do modelo é transposta para uma estimativa de complexidade da dificuldade de codificação do projeto.

Estimativas de complexidade, além de precificação, auxiliam arquitetos de software na detecção de práticas equivocadas na execução do modelo.

Classes com importâncias bem maiores do que o resto das classes denotam possíveis *god classes*. Componentes com estimativas de complexidade muito elevadas em relação a outros componentes indicam possível uso incorreto de relacionamentos do tipo composição, agregação, dependência, abstração e generalização, pois estes tipos de relacionamentos aproximam as classes para pertencer a um determinado componente. Por fim, projetos com estimativa de complexidade alta na comparação a outros projetos podem significar criação de classes obsoletas no modelo.

5.5. Estimativas de coesão

Estimativas de coesão são calculadas sob os erros residuais contidos na saída de cada componente na aplicação do algoritmo *k-means*, utilizado para a componentização do grafo orientado de classes. Os erros residuais indicam quão próximas as classes do componente estão do seu centro (centróide do grupo).

Em outras palavras, os erros residuais são estimativas de coesão dos componentes gerados. Quanto maior for o erro residual, menor será a coesão do objeto estudado.

A coesão do projeto é o somatório dos erros residuais de todos os componentes alocados de um determinado projeto.

A estimativa de coesão, além de ajudar na distribuição do trabalho, auxilia na detecção do número de grupos de classe do projeto. Um componente com erro residual elevado pode ser subdividido em outros componentes através do aumento do número de grupos de classe do projeto. Além disto, projetos e componentes com grandes erros residuais expressam instabilidade alta, maior rigidez, baixo reúso e dificuldades de manutenção, visto que uma modificação dentro de uma classe acarreta em uma cascata de alterações em outras classes.

Assim, a coesão é um valor importante para variados processos de software, e isto inclui o Scrum, na estipulação das dificuldades dos sprints de refatoração e correção de erros citados na seção 2.3.

5.6. Métricas de Tamanho

As métricas de tamanho são aplicações de fórmulas básicas que estimam o tamanho de um projeto do ponto de vista das classes e dos relacionamentos entre classes. Estas métricas estimam a força bruta necessária para realizar o projeto, sendo estimativas quantitativas e não qualitativas.

A seguir, a lista de métricas de tamanho calculadas na ferramenta AnalisadorUML:

- Número de Classes por Projeto.
- Número de Relacionamentos por Projeto.
- Número de Relacionamentos de Abstração por Projeto.
- Número de Relacionamentos de Generalização por Projeto.
- Número de Relacionamentos de Dependência por Projeto.
- Número de Relacionamentos de Realização por Projeto.
- Número de Relacionamentos de Associação por Projeto.
- Número de Relacionamentos de Composição por Projeto.
- Número de Relacionamentos de Agregação por Projeto.
- Número de Componentes por Projeto.

- Número de Classes por Componente.

5.7. Métricas de Acoplamento

As métricas de acoplamento também são análises quantitativas. São calculadas fórmulas sobre cada componente gerado na componentização do grafo orientado de classes.

Estas métricas definem os relacionamentos existentes entre os componentes. Idealmente, os componentes deveriam ser estáveis, sem depender de classes externas. As dependências estariam todas exprimidas na interface do componente, que representaria a sua comunicação externa.

Métricas de acoplamento medem a dificuldade de se desenvolver tais interfaces externas. Elas são listadas na ferramenta *AnalisadorUML* da seguinte maneira:

- Número de Relacionamentos Internos por Componente.
- Número de Relacionamentos Internos de Abstração por Componente.
- Número de Relacionamentos Internos de Generalização por Componente.
- Número de Relacionamentos Internos de Dependência por Componente.
- Número de Relacionamentos Internos de Realização por Componente.
- Número de Relacionamentos Internos de Associação por Componente.
- Número de Relacionamentos Internos de Composição por Componente.
- Número de Relacionamentos Internos de Agregação por Componente.
- Número de Relacionamentos Externos por Componente.
- Número de Relacionamentos Externos de Abstração por Componente.
- Número de Relacionamentos Externos de Generalização por Componente.

- Número de Relacionamentos Externos de Dependência por Componente.
- Número de Relacionamentos Externos de Realização por Componente.
- Número de Relacionamentos Externos de Associação por Componente.
- Número de Relacionamentos Externos de Composição por Componente.
- Número de Relacionamentos Externos de Agregação por Componente.

Os relacionamentos internos do componente são relacionamentos onde a classe origem e a classe destino estão alocadas no componente. Os relacionamentos externos têm uma das classes (origem ou destino) pertencentes ao componente e a outra não pertencente.

Algumas anomalias de design podem aparecer nestas métricas. Uma destas anomalias é a detecção de relacionamentos externos de composição, agregação e dependência, fato que não é desejado para a etapa de design. Uma das formas de corrigi-la é modificar o modelo de diagrama de classes e outra forma é diminuir o número de grupos do projeto dentro da ferramenta.

5.8. Atribuições de classes a desenvolvedores

A ferramenta *AnalisadorUML* suporta a alocação de usuários desenvolvedores, gerentes e arquitetos em projetos.

Estes usuários alocados dentro do projeto, podem ser atribuídos a classes específicas extraídas do diagrama de classes do XMI. Assim, o gerente do projeto saberá responsabilizar mau uso ou inconsistências do modelo por parte dos arquitetos e futuros desempenhos de codificação por parte dos desenvolvedores.

As atribuições às classes foram implementadas no *AnalisadorUML* com o intuito do usuário gerente analisar as classes mais importantes e os componentes gerados, estudar a distribuição do trabalho e executar esta distribuição dentro da mesma ferramenta.

Desta maneira, aproveitam-se as classes extraídas do diagrama de classe e evita-se a transposição de todas as informações disponíveis para outra ferramenta.

5.9. Arquitetura e Uso

O *AnalisadorUML* foi desenvolvido em ambiente Java. O *Google Web Toolkit* foi utilizado como framework de exibição de telas, *grids* e gráficos. Para a implementação da parte *controller* e servidor se empregou o framework *spring*. Por fim, as metodologias *Data Access Object* e *Data Transfer Object* foram escolhidas para realizar a comunicação entre o banco de dados e as entidades do sistema.

A ferramenta se divide em dois módulos:

- Configuração.
- Projeto.

5.9.1. Configuração

O módulo *Configuração* é responsável por gerenciar os usuários e os seus respectivos papéis. Cada papel tem o poder de visualizar determinadas telas dentro do sistema. Outra responsabilidade do módulo é a atribuição dos pesos de cada tipo de relacionamento.

As seguintes telas pertencem ao módulo *Configuração*:

- *Usuários:* Edição, criação e remoção dos usuários do sistema. Os usuário contém login, nome, e-mail, papel e senha.
- *Papéis:* Edição, criação e remoção dos papéis do sistema. Usuários administradores podem permitir a visualização de telas específicas por parte de cada papel.
- *Parâmetros do Sistema:* Edição dos valores atribuídos aos pesos dos relacionamentos de abstração, generalização, dependência, realização, composição, agregação e associação.

5.9.2. Projeto

O módulo *Projeto* é o mais importante dentro do sistema. Ele é responsável por ler o XMI, aplicar os algoritmos e demonstrar os resultados e métricas.

Para a análise de um diagrama de classes é necessária, primeiramente, a criação de uma entidade chamada projeto. O projeto contém nome, caminho do

arquivo XMI, caminho do arquivo de metamodelo, caminho do arquivo de transformação e o número de grupos a ser utilizado na componentização.

Depois da criação do projeto, o segundo passo é definir os adaptadores do arquivo de metamodelo para os parâmetros que a aplicação é capaz de ler, descritos na seção 5.1.1.5, no contexto do projeto referente a um modelo específico. A ferramenta possui parâmetros padrões para a utilização do metamodelo UML1.1.

O terceiro passo é o agendamento dos processos de leitura do XMI, de detecção dos *core concepts* e de componentização, que define a data de execução de cada tarefa. O processo de leitura do XMI deve ser o primeiro a ser agendado. O framework *quartz* foi utilizado para desenvolver esta etapa do projeto.

Logo após o término da tarefa de leitura do XMI, é possível conferir as classes que foram extraídas do modelo e também as métricas de tamanho do projeto. Depois do término da tarefa de obtenção dos *core concepts*, é possível conferir as classes mais importantes do sistema, ordenadas por grau de importância, e algumas métricas de complexidade do projeto. Enfim, com o término da tarefa de componentização, são demonstrados os grupos de classes gerados com os seus erros residuais, bem como as métricas de acoplamento do projeto e as métricas de tamanho, complexidade e acoplamento de cada componente dentro do projeto.

O módulo *Projeto* também possibilita a alocação de usuários do sistema dentro de projetos como Gerente, Observador, Arquiteto ou Desenvolvedor. Estes usuários, por sua vez, podem ser alocados em classes dentro do projeto depois de análise do modelo.

As seguintes telas pertencem ao módulo *Projeto*:

- *Projetos:* Edição, criação e remoção de projetos. Na criação e edição de projetos, é realizado o *upload* do arquivo de XMI, do arquivo de metamodelo e do arquivo de transformação da máquina cliente para a máquina servidora.
- *Tipos de Parâmetros:* Edição, criação e remoção dos adaptadores dos elementos do arquivo de metamodelo (representado por *Nome*) com o elemento que a ferramenta é capaz de ler (representado por *Tipo Parâmetro*).

- *Atributos:* Edição, criação e remoção dos adaptadores dos atributos de cada elemento do arquivo de metamodelo (representado por *Nome*) com o atributo (representado por *Atributo Tipo Parâmetro*) do elemento (representado por *Tipo Parâmetro*) que a ferramenta é capaz de ler.
- *Usuários:* Edição, criação e remoção da alocação dos usuários dentro de projetos.
- *Agendamentos:* Edição, criação e remoção dos agendamentos dos processos de leitura do XMI, obtenção dos *core concepts* e componentização do modelo.
- *Classes:* Exibição das classes extraídas na leitura do XMI e seus respectivos relacionamentos, métodos, estereótipos e contexto.
- *Componentes:* Exibição dos componentes gerados no processo de componentização em forma de lista de classes com os seus respectivos erros residuais.
- *Core Concepts:* Exibição de cada classe com o seu respectivo grau de importância dentro do modelo, ordenado do maior grau para o menor.
- *Atribuir Classes:* Edição, criação e remoção da alocação dos usuários às classes dentro de projetos.
- *Estatísticas:* Exibição das métricas de tamanho, acoplamento e complexidade dos projetos e dos componentes.

5.9.3. Banco de Dados

O banco de dados do *AnalisadorUML* é persistido sob a plataforma do sistema gerenciador de banco de dados *PostgreSQL 9.2*. Este banco serve para armazenar informações de usuários, papéis, pesos dos relacionamentos, projetos e resultados dos processos de leitura do XMI, de obtenção dos conceitos centrais e da componentização do modelo.

A seguir a relação das principais tabelas do sistema:

- *TB_USUARIO:* Tabela que guarda informações de usuário, extraídas da tela *Configuração.Usuários*.

- *TB_PAPEL:* Tabela que guarda informações de papel, extraídas da tela *Configuração.Papéis*.
- *TB_PROJETO:* Tabela que guarda informações de projeto, extraídas da tela *Projeto.Projetos*.
- *TB_PROJETO_TIPO_PARAMETRO:* Tabela que guarda informações de elementos do adaptador entre o arquivo de metamodelo e a ferramenta *AnalisadorUML*, extraídas da tela *Projeto.TiposDeParâmetro*.
- *TB_PROJETO_TIPO_PARAMETRO_ATRIBUTO:* Tabela que guarda informações de atributos de cada elemento do adaptador entre o arquivo de metamodelo e a ferramenta *AnalisadorUML*, extraídas da tela *Projeto.TiposDeParâmetro*.
- *TB_USUARIO_PROJETO:* Tabela que guarda informações de usuários alocados em projetos, extraídas da tela *Projeto.Usuários*.
- *TB_PARAMETRO_SISTEMA:* Tabela que guarda informações dos tipos de relacionamento UML com os seus respectivos pesos, extraídas da tela *Configuração.ParâmetrosDoSistema*.
- *TB_AGENDAMENTO:* Tabela que guarda informações dos agendamentos de processos de leitura do XMI, de obtenção dos conceitos centrais e de componentização, extraídas da tela *Projeto.Agendamentos*.
- *TB_CLASSE:* Tabela que guarda informações do nome, projeto e estereótipos das classes extraídas no processo de leitura do XMI e, também, do grau de importância e do componente alocado, extraídas dos conceitos centrais e da componentização, respectivamente.
- *TB_ATRIBUTO_CLASSE:* Tabela que guarda informações do nome e visibilidade dos atributos das classes, extraídas no processo de leitura do XMI.
- *TB_METODOS_CLASSE:* Tabela que guarda informações do nome e visibilidade dos métodos das classes, extraídas no processo de leitura do XMI.
- *TB_RELACAO_CLASSE:* Tabela que guarda informações de relacionamentos UML entre classes, extraídas no processo de leitura

do XMI. Esta tabela contém colunas que representam a classe origem e a classe destino, distância e o tipo do relacionamento.

- *TB_USUARIO_CLASSE:* Tabela que guarda informações de usuários alocados em classes, extraídas da tela *Projeto.AtribuirClasses*.
- *TB_COMPONENTE:* Tabela que guarda informação de componentes e seus respectivos erros residuais, extraídas do processo de componentização do modelo de diagrama de classes.