1
Introdução

O design e a arquitetura de projetos fazem parte da etapa de modelagem dentro do processo de desenvolvimento de software e, posteriormente, servem para facilitar a manutenção.

A modelagem é feita antes da codificação. Fazer uso de um modelo assegura o sucesso de desenvolvimento do software na verificação da completeza e da corretude das funcionalidades de negócio (Object Management Group, 2012).

A UML[1] é a linguagem mais utilizada na etapa de design. Um dos seus aspectos mais importantes é o poder de escolha do detalhamento que se dá ao software. Eleva-se o nível de abstração até onde se deseja (Object Management Group, 2012).

São treze os tipos de diagrama definidos para o UML, segundo Object Management Group (2012), e estes são divididos em três categorias:

- *Diagramas estruturais*: Diagramas de classe, objeto, componente, estrutura de composição, pacote e de desenvolvimento.
- *Diagramas de comportamento*: Diagramas de caso de uso, atividade e máquina de estados.
- *Diagramas de interação*: Diagramas de sequência, comunicação, tempo e de resumo de interação.

O emprego destes diagramas nas fases de concepção e elaboração de projetos é feito em larga escala, principalmente para problemas mais complexos, onde é necessário um maior entendimento através de um modelo. Assim, a UML ganhou notoriedade na indústria, mas será que ele atende a todos os requisitos da fase inicial de um projeto?

Na fase inicial de um projeto são necessárias informações de cunho geral tanto para o âmbito do desenvolvimento quanto para o âmbito gerencial.

---

[1] Unified Modeling Language.

Como o esboço do modelo do software já se encontra pronto nesta fase através de diagramas UML seria interessante extrair destes diagramas informações de estimativas de complexidade, coesão, acoplamento, custo e trabalho.

À medida que a UML se transformou em um padrão de linguagem de design, muitas ferramentas passaram a adotá-la, o que causou um problema onde a linguagem é um padrão, mas o mesmo não acontece com as ferramentas que a utilizam.

O desafio é elaborar uma ferramenta compatível com todo este ferramental UML e prover as informações extras para um gerente de desenvolvimento custear e dividir o trabalho através de diferentes estimativas de complexidade e de coesão do projeto.

1.1. Motivação

Os arquitetos de software, geralmente os responsáveis por implementar os diagramas UML, têm em suas mãos a modelagem do software. Porém quando estes diagramas são analisados, percebe-se que eles têm um embasamento muito mais técnico no ponto de vista de arquitetura do desenvolimento por expôr os relacionamentos entre as entidades do sistema.

Apesar dos diagramas UML terem se tornado padrões de design no processo de desenvolvimento de um software, a própria etapa de arquitetura e design sofre com a entrada de metodologias ágeis.

O ímpeto de entregar os projetos de maneira mais rápida contraria a qualidade do software. Pela norma ISO 9000, a qualidade é o grau em que um conjunto de características inerentes a um produto, processo ou sistema cumpre com os requisitos inicialmente estipulados para estes (Guerin & Rice, 1996). O uso de modelos para o entendimento do problema pode evitar retrabalhos e visa à manutenção da qualidade do software desde sua etapa inicial.

De acordo com Abrantes & Travasos (2011), as tecnologias ágeis, como o Scrum e o XP, primam por dois pilares no atendimento aos requisitos:

- *Small releases*: Esta prática encurta o ciclo de desenvolvimento dos entregáveis para aumentar o *feedback* do cliente. Dado que os requisitos mudam frequentemente, os ciclos curtos de entregáveis evitam retrabalho de alta magnitude em futuras mudanças de

requisitos já desenvolvidos. O resultado é a produção de um software utilizável, que agrega valor de negócio ao cliente.

- *Simple design*: A ênfase desta prática é desenvolver o design da solução mais simples, implementável naquele momento. Complexidade desnecessária e código extra devem ser removidos, sem a adição de funcionalidades extras. O foco é resolver o problema de hoje ao invés de considerar possíveis mudanças no futuro.

As técnicas de *small releases* e *simple design* são contraditórias ao uso da modelagem por definir que o trabalho despendido na elaboração dos diagramas não compensa. Neste ponto, o produto final da arquitetura e modelagem do sistema não agrega informações suficientes, ou seja, o custo-benefício é baixo.

A motivação deste trabalho é justamente agregar informações extras aos diagramas através de análises e métricas. Sendo assim, o diagrama, que é um dos produtos finais da fase de arquitetura de software, vai ganhar mais corpo e talvez fique mais atraente para determinados usos em diferentes processos.

1.2. Especificação do Problema

1.2.1. Enunciado

A proposta dessa dissertação é agregar informações gerenciais em um dos diagramas UML, o diagrama de classes. A ideia é utilizar um conjunto de algoritmos para, dado um diagrama de classes de entrada, retirar algumas informações do ponto de vista de distribuição de trabalho, precificação e estimativas de complexidade e de coesão do projeto ainda em fase de design, o que evita o retrabalho e surpresas negativas em fases posteriores do processo de desenvolvimento de software. Outro ponto é a extração de informações dos diagramas de classe que evidenciem vícios de má modelagem por parte de arquitetos de software.

O primeiro tópico abordado é a descoberta de quais são as classes mais importantes, ditas *core concepts*, do sistema. Do ponto de vista gerencial, se supõe que as classes mais importantes deveriam ser implementadas por desenvolvedores

mais experientes e os testes em cima destas classes precisariam de uma atenção especial.

O segundo assunto é a determinação dos grupos de classes do sistema, que consiste em coletar as classes que têm mais proximidade entre si no diagrama e alocá-las em componentes. Isto é interessante para criar grupos de desenvolvedores dentro do projeto para implementação em conjunto e, também, para especializar um determinado desenvolvedor em um grupo de classes específico do projeto. Estes componentes e suas implicações ajudam um gerente de projetos na distribuição do trabalho.

De posse destas duas informações, infere-se, através de métricas a serem descritas posteriormente, as estimativas de complexidade[2] e de coesão do projeto como um todo e de cada componente integrante. A avaliação da coesão e da complexidade auxiliará na estimativa de precificação de um projeto através da comparação com projetos anteriores com medidas similares dessas propriedades. Ou seja, se um projeto já fechado teve complexidade total 100 e um custo de R$100.000,00, outro projeto ainda em fase de análise que tem complexidade total estimada em 120, deveria custar em torno de R$120.000,00.

Por fim, na realização do confronto dos *core concepts* e dos componentes extraídos pelo estudo, com os *core concepts* e os componentes esperados como resposta, o arquiteto de software pode eliminar erros de modelagem tais como classes desproporcionalmente grandes e complexas ou mau uso dos relacionamentos dentro do diagrama de classe.

1.2.2. Proposta

- Desenvolver um leitor do tipo de arquivo padrão de exportação XMI das ferramentas de criação e edição de diagramas de classe.
- Armazenar o diagrama de classes em uma estrutura de grafo orientado, onde os nós são as classes e as arestas são os relacionamentos entre as classes.

---

[2] Complexidade neste ponto seria uma medida estimada da dificuldade de desenvolver um determinado componente ou projeto de acordo com o diagrama de classes.

- Aplicar o algoritmo de caminhos mais curtos de todos os nós a todos os nós conhecido como *Floyd Warshall*[3], onde as distâncias entre os nós são calculados em função do tipo de relacionamento das arestas do grafo.
- Aplicar o procedimento de obtenção dos *core concepts* com base em contagem dos nós presentes em caminhos mais curtos.
- Aplicar o algoritmo de clusterização denominado *k-means* para a componentização do grafo.
- Calcular estimativas de complexidade através de fórmulas aplicadas aos valores de importância de cada classe, obtidas na concepção dos *core concepts*.
- Calcular estimativas de coesão através de fórmulas aplicadas sobre os erros residuais, obtidos na saída da clusterização.
- Calcular métricas de tamanho e acoplamento em cima do grafo.
- Possibilitar ao usuário realizar todos estes passos de maneira customizável. A ferramenta *AnalisadorUML* foi desenvolvida em ambiente web para dar suporte aos passos descritos acima e para a avaliação dos resultados.
- Utilizar a ferramenta para, dada uma base amostral de diagramas de classe, determinar os pesos de cada tipo de relacionamento do grafo. Deste modo, melhores resultados de componentes e de *core concepts* serão gerados.
- Observar as contribuições feitas pelo *AnalisadorUML* em diferentes processos de software de projetos com complexidades variadas.

A Figura 1 sintetiza o fluxo das entidades que compõem o estudo:

[3] O algoritmo de Floyd Warshall aqui utilizado é o que guarda todos os caminhos entre os nós e não apenas a distância entre eles.

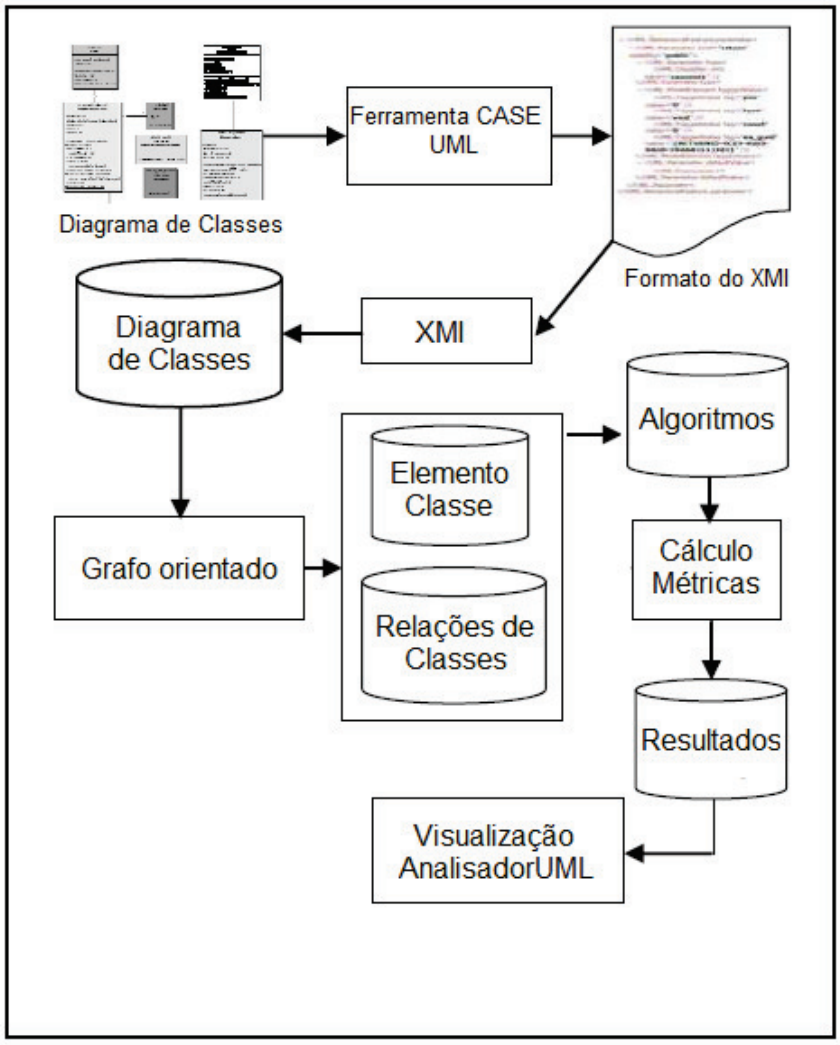

Figura 1 - Estrutura do *AnalisadorUML*.

1.3. Organização do Trabalho

O trabalho está organizado para que o leitor entenda primeiramente quem tem interesse no trabalho. Isto é evidenciado no capítulo *Aplicabilidade*. Depois disto, no capítulo *Referências e Base Técnica*, são debatidos todos os estudos relacionados a este, bem como o estado da arte. O terceiro capítulo são os *Conceitos Chaves*; nele se encontram as definições de diagrama de classes e seus relacionamentos, do XMI, de clusterização pelo algoritmo *k-means* e de caminhos mais curtos pelo algoritmo *Floyd Warshall*.

Com todas as entidades do trabalho definidas, o capítulo *Abordagem* demonstra como foi desenvolvido o *parser* de leitura dos diagramas de classe, a construção do grafo orientado de classes, a obtenção dos *core concepts* e da

componentização, como foram calculadas as estimativas de complexidade e coesão e, por fim, as métricas de tamanho e acoplamento.

O capítulo *Aplicação* começa com a base amostral criada para atribuir pesos aos relacionamentos do diagrama de classes, reflete sobre os pesos atribuídos e comenta os resultados encontrados para diferentes tipos de projeto. Finalmente, o último capítulo, *Conclusão*, indica possíveis trabalhos futuros e as dificuldades e benefícios de se utilizar a ferramenta tema do trabalho.