2
Aplicabilidade

As contribuições esperadas desse trabalho têm relação com diferentes processos de software. Então, primeiro será definido onde haverá contribuição do trabalho nos processos de software como um todo para depois definir onde haverá contribuição no *Rational Unified Process* e no *Scrum*.

2.1.
Processo de Software

Processo de software é um conjunto de atividades e resultados associados que produzem um software (Sommerville, 2004).

Uma abordagem mais específica definiria processo de software como um conjunto de atividades, ligadas por padrões de relacionamento entre elas, pelas quais se as atividades operam corretamente e de acordo com os padrões requeridos, o resultado desejado é produzido. O resultado desejado é um software de alta qualidade e baixo custo. Obviamente, um processo que não aumenta a produção (não suporta projetos de software complexos) ou não pode produzir software com boa qualidade, não é um processo adequado (Sass, 2011).

Segundo Schwartz (1975), existem quatro grandes fases em um processo de software:

- *Especificação de Requisitos*: Tradução da necessidade ou requisito operacional para uma descrição da funcionalidade a ser executada.
- *Projeto de Sistema*: Tradução destes requisitos em uma descrição de todos os componentes necessários para codificar o sistema.
- *Programação (Codificação)*: Produção do código que controla o sistema e realiza a computação e a lógica envolvida.
- *Verificação e Integração*: Verificação da satisfação dos requisitos iniciais pelo produto produzido. Embora esta etapa seja mais intensa após a codificação, a verificação e a validação devem ocorrer durante todas as etapas do processo de desenvolvimento.

Definidas as fases, é preciso definir as atividades de um processo de software. As principais atividades são a especificação, o projeto, a implementação, a validação e a manutenção/evolução. Esta última precisa ser avaliada em todas as fases (Schwartz, 1975; Pressman, 1997; Sommerville, 1995).

Para realizar tais tarefas, especificam-se modelos que representam uma visão abstrata dos elementos do processo e sua dinâmica.

É neste momento que o trabalho contribui, pois nesta etapa, ainda na modelagem do sistema, já se pode ter estimativas de precificação, de divisão de trabalho, de complexidade e de coesão. Estas informações facilitam o planejamento da etapa de projeto de sistema e de codificação e evitam possíveis surpresas de dificuldade de trabalho.

Do ponto de vista de processo de software mais geral, este trabalho seria utilizado pelo arquiteto de software desenvolvedor do modelo para refinamento do diagrama de classes, através da eliminação de inconsistências e defeitos arquiteturais, e pelo gerente de projeto, responsável pela distribuição das tarefas e por dar preço ao projeto.

2.2. Rational Unified Process (RUP)

O *Rational Unified Process* é um processo de software desenvolvido pela *Rational Software Corporation* e adquirido posteriormente pela IBM. Consiste em um framework de apoio a processos abrangente, que provê atribuições de tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a qualidade do software para respeitar as necessidades do cliente dentro de orçamentos e durações pré-determinados (Rational, 2002). Por trás deste framework, existe uma disciplina com várias regras a serem respeitadas.

Segundo a Rational (2002), para definir prazos e o horizonte de tempo, o RUP faz a divisão do projeto em quatro fases com foco em pessoa, projeto, produto e processo (os quatro P's):

- *Concepção*: A concepção trabalha com ênfase no escopo do sistema, contém os fluxos de trabalho para que os *stakeholders* concordem com as definições, arquitetura e planejamento do projeto. Esta etapa geralmente é curta e não fecha os requisitos do sistema.

- *Elaboração*: A elaboração é o projeto do sistema, responsável por complementar os casos de uso originados na concepção. Esta fase desenvolve outros diagramas UML, para explicitar a arquitetura do sistema e começar a construção do manual do usuário.
- *Construção*: A construção corresponde à implementação. São geradas versões alfa e beta para testes.
- *Transição*: A transição é a entrega para uso da versão compilada e testada do software.

Vale salientar que estas fases não são executadas em cascata, e sim, em iterações.

A contribuição deste trabalho no RUP está na fase de elaboração do projeto. Esta etapa é responsável por desenvolver o diagrama de classes.

A ferramenta *AnalisadorUML* pode ser utilizada para fornecer informações de custo através da estimativa de complexidade e de coesão do projeto, visto que o orçamento do projeto é um fator determinado antes da construção, ou seja, é preciso ser definido na elaboração.

Outro auxílio é na fase de construção, no que tange à distribuição das tarefas dadas pela componentização e pelos *core concepts*. Atribuem-se classes e componentes mais complexos a desenvolvedores mais experientes, e divide-se a equipe em grupos, cada qual responsável por codificar um determinado componente.

Os usuários desta ferramenta, assim como nos processos gerais de software, seriam o arquiteto de software e a gerência de projetos.

Os desenvolvedores também poderiam utilizar a ferramenta para facilitar a comunicação entre a equipe, visto que o desenvolvedor saberia qual componente implementar e quais desenvolvedores estão alocados em componentes contextualmente próximos.

2.3. Scrum

Scrum é um framework ágil para a gerência de projetos. Originalmente foi formalizado para projetos de desenvolvimento de software, mas funciona bem para qualquer escopo de trabalho (Scrum Alliance, 2011). Ao invés de fornecer completa e detalhadamente a maneira como tudo deve ser feito no projeto, muito

desse trabalho é deixado para a equipe de desenvolvimento de software. Isso porque a equipe saberá a melhor forma de resolver os problemas apresentados (Mountain Goat, 2011).

Este processo se baseia no manifesto ágil. A metodologia é baseada em princípios como: equipes pequenas, requisitos pouco estáveis ou desconhecidos e iterações curtas para promover visibilidade para o desenvolvimento (Soares, 2004). Por causa destas características, o Scrum não atende satisfatoriamente a projetos complexos, o que faz com que o desenvolvimento de diagramas de classe tenha baixo custo-benefício nestes ambientes.

Os requisitos, mesmo que instáveis, são definidos no documento *Product Backlog*. Para resolver os itens descritos neste documento, o *Scrum* realiza iterações denominadas *sprints*. Os *sprints* quebram os requisitos do *Product Backlog* em problemas menores e os especificam no documento *Sprint Backlog*. Para codificar os itens do *Sprint Backlog* são realizadas reuniões entre o *Scrum Master* e os desenvolvedores para descrever estes itens em tarefas pequenas e executáveis, cujos prazos e responsabilidades são definidos através de diversas metodologias. A partir daí, começa a codificação.

Além disto, são utilizados no *Scrum* os *sprints* de refatoração e de correção de erros. Muitas vezes, existe a necessidade de se refatorar projetos, complexos ou não, e corrigir um conjunto de erros dentro de um *sprint* específico, através da abordagem ágil. O trabalho contribui para o *Scrum* neste aspecto.

Como o código a ser refatorado ou corrigido já está feito, aplica-se a engenharia reversa para gerar o diagrama de classes UML automaticamente do código. Com posse do diagrama, a ferramenta *AnalisadorUML* gera os *core concepts*, os componentes, as estimativas de complexidade e coesão e as métricas de tamanho e acoplamento de maneira mais eficaz, pois todos os detalhes do código estarão presentes no diagrama de classes, tornando-o mais completo.

Assim, quando o *Scrum Master* verificar a possibilidade de uma refatoração ou correção de erros, ele e a equipe de desenvolvimento estudarão as classes principais que sofrerão alterações. Destas classes, a ferramenta explicita informações como complexidade, quais classes estão mais relacionadas (contexto da classe) e qual a complexidade e coesão do contexto. Com estas métricas avaliadas, a atribuição de custos, a divisão do trabalho e a ordenação das classes que serão refatoradas ou corrigidas terão suas definições simplificadas.