Engenharia de Software:conceitos e processos de software.

Gilmar Luiz de Borba

O presente artigo tem por objetivo introduzir os conceitos básicos da disciplina Engenharia de Software. Além dos conceitos, são apresentados os modelos de ciclo de vida e os principais processos de software, focalizando com um pouco mais detalhe o RUP.

Segundo Melo (2002), Entre as décadas de 1940 até meados da década de 1970 os sistemas de informação, na sua maioria, eram caracterizados por limitações de hardware, aplicações de pequeno porte, desenvolvidas sem uma metodologia (ou processo de desenvolvimento) formal, pouco gerenciamento, e pouco critério para realizar o mapeamento entre os principais modelos (visões) existentes. Diante desse cenário, os produtos eram entregues quase sempre fora dos prazos, havia estouro de orçamentos, projetos eram cancelados antes da entrega, os produtos geralmente não tinham o padrão de qualidade desejado (falhas e vícios dentro do software), também havia dificuldades em realizar as devidas manutenções no software. Todos esses problemas causavam desgastes nas relações entre a equipe de desenvolvimento e os usuários. Esse cenário ficou conhecido a “Crise do Software”. Em resposta a esse cenário surgiu no final da década de 1960 o conceito de Engenharia de Software.

Conforme nos ensina Sommerville, (2011), a engenharia de software é "É uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até a sua manutenção, quando o sistema já está sendo usado." (SOMMERVILLE, 2011:5), baseado nesse conceito podemos observar a diferença entre a engenharia de software e a ciência da computação, uma vez que esta última, foca a teoria e os fundamentos, sendo, até mesmo, mais voltada para o campo da pesquisa. A engenharia de sistemas, preocupa de um modo geral, com os aspectos relativos ao desenvolvimento dos sistemas computacionais, incluindo hardware, software e processo, assim, constata-se que a engenharia de software é uma parte mais específica desse processo (engenharia de sistemas), que é mais genérica. (SOMMERVILLE, 2011:4).

Atualmente, devido á rapidez com que os sistemas são desenvolvidos, rapidez essa, originada de uma série de fatores, como: novas tecnologias, novos processos de desenvolvimento, globalização e conseqüentemente a necessidade de respostas

rápidas às necessidades dos usuários que mudam constantemente, nota-se que, de acordo com Sommerville (2011), lidar com o aumento de diversidade, demandas pela diminuição do tempo para a entrega e desenvolvimento de software confiável é o principal desafio da engenharia de software.

O software deve ser desenvolvido (projeto e codificação) de maneira a atender às necessidades dos clientes (à medida da sua evolução). Esse cenário é preocupante, uma vez que, a mudança pode ser vista como a principal constante no desenvolvimento de software. Concluímos que o software deve ter como característica a facilidade de manutenção, conceito conhecido também como manutenibilidade. Outra característica é a confiança e proteção, o software, quando confiável não deve causar prejuízos (sejam eles físicos ou econômicos). O software produzido, deve também ter uma boa capacidade de resposta e tempo de processamento adequado para o objetivo ao qual se propõe a alcançar, essa é outra característica do software de qualidade, e é denominada, eficiência. Entre outras características do software de qualidade, Sommerville (2011), aponta a aceitabilidade, ou seja, o software deve ser aceitável para o tipo de usuário para o qual foi projetado, assim, deve ser compreensível, usável e compatível com outros sistemas usados por ele.

Nas atividades da engenharia de software é comum o uso do termo processo, uma vez que, esses processos devem ser estudados pelo engenheiro de software para que, se possível, possam ser automatizados, ou escritos em alguma linguagem de programação. Segundo Pádua (2005), um processo é

> um conjunto de passos particularmente ordenados, construídos por atividades, métodos, práticas e transformações, usados para atingir uma meta. Esta meta geralmente está associada a um ou mais resultados concretos finais, que são os produtos da execução do processo. (PÁDUA, 2005,11).

Na engenharia de software esse conceito abrange as atividades que permeiam o ciclo de vida dos sistemas, como por exemplo: o desenvolvimento, a manutenção, aquisição e a contratação de software. Neste sentido pode-se dizer que o processo pode responder perguntas como o que é feito, como é feito, por quem é feito e finalmente o que produzirá. Os processos de desenvolvimento de software atuais são regidos por normas que especificam o que deve ser feito, definindo os processos fundamentais, suas fases, seus fluxos e atividades, metas e resultados.

Conforme relata Bezerra (2007), devido à grande complexidade dos sistemas, complexidade esta originada pela sobreposição das atividades relacionadas aos procedimentos, software, hardware e pessoal, um grande número de projetos não chegam ao final. De acordo com dados levantados pela Chaos Report (1994), somente 10% dos projetos terminam dentro do prazo. Sendo assim é importante usar os processos de desenvolvimento de software para acompanhar as mudanças, gerenciar requisitos, e controlar continuamente a qualidade do software. Todo esse controle é baseado na arquitetura e no ciclo de vida dos processos de desenvolvimento de software. Entre os ciclos de vida de processos de desenvolvimento de software existentes, podemos citar:

O Modelo Codifica-remenda. É um modelo particularmente caótico, normalmente sem a presença de um processo formal definido, os erros aparecem e são corrigidos com pouco critério, sem documentação. Há pouco ou nenhum controle gerencial e técnico. O Modelo em Cascata. Nesse modelo há uma seqüência pré-definida na execução dos processos, permitindo a inserção de marcos, o que vem a facilitar a gestão do projeto. Este processo se interpretado literalmente torna-se rígido, lento e burocrático. Sua arquitetura fornece pouca transparência tanto para o lado do desenvolvimento quanto para o usuário final. O Modelo em Espiral. No modelo em espiral há uma série de iterações, cada uma correspondendo a uma volta na espiral que a partir dos resultados obtidos fornecem dados para a nova iteração. Este modelo permite o desenvolvimento de produtos à curto prazo com novos recursos agregados a partir de cada nova iteração. Por outro lado requer uma gestão sofisticada. O Modelo de Prototipagem evolutiva. Trata-se da construção de uma série de versões chamadas de protótipos. Os protótipos, ao longo do processo, cobrem cada vez mais novos requisitos até atingir o produto desejado. É um modelo flexível, iterativo (baseado no modelo em espiral). Da mesma forma do modelo em espiral, requer gestão sofisticada. Exemplo: XP (Extreme Programing). O Modelo de Entrega evolutiva. Este modelo reúne características dos modelos em cascata e prototipagem evolutiva. Ele permite que os usuários avaliem o produto em pontos estratégicos do processo e conseqüentemente realimentá-lo com as novas implementações. Possibilita bom gerenciamento do projeto. A desvantagem está em desenhar uma arquitetura robusta de forma a manter a integridade do produto final durante as liberações parciais.

Complementando os modelos de ciclos de vida descritos anteriormente merecem destaque ainda o modelo dirigido por prazo e o modelo dirigido por ferramenta CASE. O modelo de processo de desenvolvimento dirigido por prazo define que, o que se consegue fazer dentro de um prazo é o que determinará o produto. Os prazos são determinados de acordo com os interesses do contratante e o produto é entregue baseado neste. PÁDUA (2005) descreve esse modelo:

> Na prática, os prazos costumam ser definidos de forma política, e o "produto" que se entrega no final do prazo é apenas um resultado parcial. Ele será completado aos trancos e barrancos em desenvolvimento posterior, disfarçado de "manutenção". (PÁDUA, 2005:15).

Um último modelo de processo, denominado modelo dirigido por ferramenta CASE normalmente impõe processos rígidos adequados à ferramenta escolhida. Conseqüentemente a qualidade desses processos dependerá da qualidade da ferramenta.

Exemplos de Processos de Software

A indústria de software através de organizações globais como ISO[1], IEC[2] e IEEE[3] tem procurado prover meios de tornar o processo de desenvolvimento de software mais maduro e com maior qualidade. Isto é feito a partir da proposição de novos modelos e padrões que abrangem todo ciclo de vida, desde a concepção até a entrega do produto. Como exemplos de processos, podemos citar: o UP (Unified Process), proposto pelos três[4] amigos. Este processo usa a UML, é dirigido por caso de uso, é centrado em uma arquitetura, é incremental e interativo e usa o modelo em espiral. O UP é dividido no tempo a partir de fases (concepção, elaboração, construção e transição) e fluxos (requisitos, Análise, desenho, implementação e testes). O EUP (Enterprise Unified Process) é uma extensão do processo unificado, possui uma fase adicional de produção (produto em operação e manutenção) e a partir desta, permite implementar novos fluxos. O PSP (Personal Software Process) incentiva a prática individual de engenheiros de software, tem por objetivo melhorar a previsibilidade, a qualidade e o ciclo de vida dos produtos. O PSP é focado em atingir

[1] ISO: International Organization for Standardization.
[2] IEC: International Electrotechnical Commission.
[3] IEEE: Institute of Electrical and Electronics Engineers.

[4] Rumbaugh, Jacobson e Booch.

um grau elevado de maturidade, foi definido a partir dos conceitos do CMM[5] e para atingir essa maturidade foi percebido que é necessário melhorar os processos a partir dos seus desenvolvedores individuais.O processo TSP (Team Software Process) é focado para a formação de equipes de desenvolvimento direcionado em métricas e inspeções. Possui fases como lançamento, estratégia, planejamento, requisitos, desenho e implementação. O processo visa os seguintes resultados: criatividade, gerir planos e reduzir custos.

O PRAXIS (**PR**ocesso para **A**plicativos e**X**tensíveis **I**nterativo**S**) é um processo inicialmente projetado para fins didáticos. Segundo PÁDUA (2005), o PRAXIS foi

> desenhado para suportar projetos realizados individualmente ou por pequenas equipes com duração de seis meses a um ano. Com isso pretende-se que ele seja utilizável para projetos de fim de curso de graduação ou similares, ou projetos de aplicação de disciplinas da engenharia de software. (PÁDUA, 2005:21).

O PRAXIS é baseada na orientada a objetos e usa a notação UML. O ICONIX, desenvolvido pela ICONIX Software Engeneering, não é tão burocrático quanto o RUP. É um processo relativamente simples, também utiliza a UML e obrigatoriamente usa rastreabilidade de requisitos.

O XP (Extreme Programming) foi construído baseado na necessidade de melhorar o desempenho de projetos. A partir de experiências anteriores constatou-se que alguns princípios eram sempre respeitados em projetos vencedores, entre eles destacam-se: a boa comunicação, a simplicidade, o feedback e a coragem. Com a implementação de práticas como jogos de planejamento, pequenas versões, programação em par, reuniões em pé, entre outras, este modelo de desenvolvimento de software tornou-se preferido por uma grande comunidade de usuários. O XP surgiu com o Manifesto Ágil, que é composto por quatro declarações de valores que, de um modo geral, relacionam os indivíduos e as interações entre eles; o software em funcionamento; colaboração com o cliente e a resposta rápida às mudanças. Entre os princípios trazidos nesse manifesto podemos citar: satisfação do cliente; receber mudanças de requisitos mesmo em estágios mais avançados do desenvolvimento; entrega do software mesmo com poucas semanas de desenvolvimento; equipes de negócio e desenvolvimento devem trabalhar juntas; dar motivação e apoio à equipe; ter software funcionando mostra progresso, entre outros.

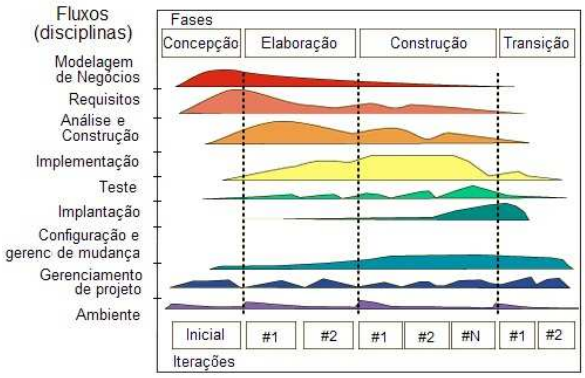
[5] Capabitlity Maturuty Model

O RUP (Rational Unified Process) trata-se de produto de processo. Foi desenvolvido e é mantido pela Rational Software que fornece um conjunto de ferramentas para o desenvolvimento de software, segundo KRUCHTEN (2003) o

> Rational Unified Proccess é um processo de engenharia de software. Ele fornece uma abordagem disciplinada para assumir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo é assegurar a produção de software de alta qualidade que satisfaça as necessidades de seus usuários finais dentro de prazo e orçamento previsíveis. (KRUCHTEN, 2003: 15).

A idéia de desenvolver um produto de processo de software parte do princípio que “processos de software são softwares também”. (OSTERWEIL apud KRUCHTEN, 2003: 16). Essa abordagem traz benefícios como: manter o processo sempre atualizado, o usuário tem acesso a versões do processo, facilidade de melhorias e gerenciamento. Entre as muitas características do RUP, podemos citar: o desenvolvimento iterativo; o gerenciamento de requisitos; usar uma arquitetura baseada em componentes e fazer a modelagem usando a UML. O desenvolvimento iterativo é uma das características mais importantes desse processo de desenvolvimento uma vez que leva em conta as mudanças de requisitos, permite o gerenciamento de riscos, mudanças e correções a cada iteração.

O RUP (Rational Unified Process) possui uma arquitetura baseada em duas dimensões, que permite definir as atividades, metas, marcos e resultados de cada processo. A primeira são as fases (uma organização ao longo do tempo) que representam aspectos dinâmicos . A segunda dimensão são os fluxos ou disciplinas que representam aspectos estáticos do processo. A Figura a seguir mostra esta estrutura:



Este processo traz diversos benefícios para os usuários, dentre eles, podemos citar: a manutenção da qualidade do próprio software, grande aceitação por grande parte dos envolvidos no processo de desenvolvimento, facilidade de configuração e gerenciamento de mudanças, facilidade de entendimento do negócio (o desenvolvimento é orientado a caso de uso) e evitar o retrabalho.

Referências Bibliográficas:

KRUCHTEN, Philippe. **Introdução ao RUP - Rational Unified Process**. Rio de Janeiro: Ciência Moderna, 2003.

BEZERRA, E**. Princípios de Análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

MELO, Ana Cristina. **Desenvolvendo aplicações com UML.** Rio de Janeiro: Brasport, 2002.

FILHO, Wilson de Pádua. Engenharia de Software – Fundamentos, Métodos e Padrões. LTC, 2001.

SOMMERVILLE, Ian**. Engenharia de Software**. 9ª. Ed. São Paulo. Pearson, Prentice Hall, 2011.