Modelos de Processos de Engenharia de Software

Rafael Orivaldo Lessa, Edson Orivaldo lessa Junior

Princípios da Engenharia de Software – Universidade do Sul de Santa Catarina (UNISUL)
Caixa Postal 88132-000 – Palhoça – SC – Brasil

edsonlessajunior@yahoo.com.br, rafinha_lessa@yahoo.com.br

Abstract. *The software engineering appeared to correct problems with regard to the development of software projects, from this sprouting models of processes and guide of developments had been created for improve of the process of development of software. The article will show to some models of processes and a guide calling SWEBOK, that was created by members of the which had scientific community to the vast material that involves the software engineering.*

Resumo. *A engenharia de software surgiu para corrigir problemas com relação ao desenvolvimento de projetos de software, a partir desse surgimento modelos de processos e guia de desenvolvimentos foram criados para otimização do processo de desenvolvimento do software. O artigo mostrará alguns modelos de processos e um guia chamando de SWEBOK, que foi criado por membros da comunidade cientifica devido ao vasto material que envolve a área.*

1. Engenharia de Software

O software de computadores é hoje uma tecnologia importante no âmbito mundial, tendo um crescimento rápido desde a década de 1950, com esse ríspido crescimento começou a ocorrer problemas relacionados a correções, adaptações, aperfeiçoamento é ainda o processo de manutenção que consome mais recurso e mais pessoas que na criação de novos software. Com todos esses problemas novos conceitos começaram ser criados com o da Engenharia de Software.

A Engenharia de software segundo Fritz Bauer “é a criação e a utilização de sólidos princípios de engenharia a fim de obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em maquinas reais”. Já IEEE desenvolveu uma definição mais abrangente que é “(1) aplicação de uma abordagem sistemática, disciplinada e quantificável, para desenvolvimento, operação e manutenção do software, isto é a aplicação da engenharia ao software. (2) Os estudos de abordagens como as de (1)”. (PRESSMAN, 2006)

Todas essas definições estão inseridas no livro de Pressman que ressalva que a engenharia de software é uma tecnologia em camadas (Ferramentas, Métodos, Processo, Foco na qualidade) e a organização devem se apoiar num compromisso de qualidade.

2. Técnicas de engenharia de Software

A Engenharia de Software é uma técnica de sucesso para projetos de software, pôr em sua maioria, não são aplicadas na produção de Software. No entanto, ainda há problemas em produzir software complexo, que atenda às expectativas dos usuários e que seja entregue dentro do prazo e do orçamento estabelecido. Muitos projetos de software ainda têm problemas, e isso levou alguns críticos, como Pressman (PRESSMAN, 2006), a sugerirem que a Engenharia de Software está em um estado de aflição crônica.

À medida que a nossa capacidade de produzir software aumentou, também cresceu a complexidade dos sistemas requeridos. Novas tecnologias que resultam da convergência de sistema de computadores e de comunicação, trazem novas questões para os engenheiros de software. Por essa razão e pelo fato de muitas empresas não aplicarem as técnicas de Engenharia de Software de maneira eficaz, apresenta diversos problemas. A situação não é tão ruim como os pessimistas sugerem, mas, com certeza, há espaço para melhorias.

Um modelo de processo de software é uma representação abstrata de um processo de software. Cada modelo de processo representa um processo a partir de uma perspectiva particular, de uma maneira que proporciona apenas informações parciais sobre o processo (SOMMERVILLE 1995). Abaixo constam alguns modelos de processos de softwares que serão abordados no artigo:

- O modelo em cascata;
- Desenvolvimento evolucionário;
- Desenvolvimento iterativo
    - Modelo de desenvolvimento espiral;
    - Modelo de desenvolvimento incremental.
- Desenvolvimento baseado em componentes

Existem outros modelos além dos citados acima, que Pressman (2006) coloca em seu livro, mas o artigo não entrará em detalhes nesses modelos ele são:

- Modelo Incremental;
- Modelo RAD;
- Modelo de Desenvolvimento Concorrente;
- Modelo de Métodos Formais;
- Processo Unificado.

2.1 Cascata, Linear ou Clássico

O modelo cascata descreve um método de desenvolvimento que é linear e seqüencial. Na primeira vez que uma fase de desenvolvimento é completada, o desenvolvimento prossegue para a próxima fase e não há retorno. A vantagem do desenvolvimento

cascata é que ele permite controle departamental e gerencial. Um planejamento pode ser atribuído com prazo final para cada estágio de desenvolvimento e um produto pode prosseguir no processo de desenvolvimento, teoricamente ser entregue no prazo. O desenvolvimento move do conceito, através do projeto (design), implementação, teste, instalação, descoberta de defeitos e termina com a operação e manutenção. Cada fase de desenvolvimento prossegue em uma ordem estrita, sem qualquer sobreposição ou passos iterativos. (PRESSMAN, 2006)

A desvantagem do desenvolvimento em cascata é que ele não permite muita flexibilidade ou revisão. A primeira vez que uma aplicação está em estágio de teste é muito difícil retornar e mudar alguma coisa que não foi bem pensada no estágio conceitual.

O modelo cascata inicia de uma abordagem orientada a projetos para a engenharia de software. O projeto é considerado uma tarefa claramente delineada para a qual os resultados desejados podem ser determinados completamente e sem ambigüidade.

Pressman (1995) aponta alguns problemas identificados neste ciclo de vida:

- Os projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Alguma iteração sempre ocorre e traz problemas na aplicação do paradigma
- Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente. O ciclo de vida clássico exige isso e tem dificuldade de acomodar a incerteza natural que existe no começo de muitos projetos
- O cliente deve ter paciência. Uma versão de trabalho dos programas não estará disponível até um ponto tardio do cronograma do projeto. Um erro crasso, se não for detectado até que o programa de trabalho seja revisto, pode ser desastroso. Peters (2001) aponta vantagens e desvantagens:

Vantagens: Permitir a gerência do baseline, que identifica um conjunto fixo de documentos produzidos como resultado de cada fase do ciclo de vida.

Desvantagens: não é capaz de estabelecer como efetuar engenharia reversa de um sistema existente e faltam noções de prototipação rápida e desenvolvimento incremental.

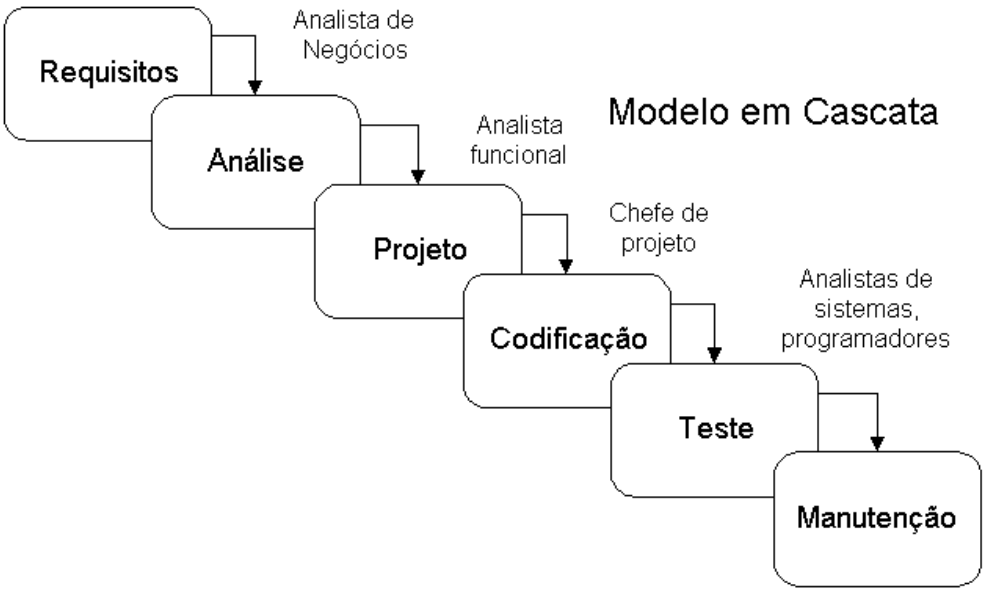

Figura 1. Modelo de processo em Cascata

2.2 Prototipação

Prototipação, é a montagem de protótipos, ela pode ser classificada de acordo com uma variedade de dimensões. A abordagem de prototipação tem um número de vantagens importante a oferecer, das quais nós indicamos a mais importante delas aqui. Primeira todo o requisitos de sistema não tem que ser completamente determinado antecipadamente e pode mesmo ser trocada durante o curso do projeto. Segundo, a entrega de prototipação clara, definições de sistema entendível e especificações para o usuário final. Como conseqüência, o envolvimento e satisfação do usuário final são fortemente aumentados. Finalmente, prototipação faz isso possível para rapidamente testar o ambiente de desenvolvimento voltado para a funcionalidade, performance, interface com banco de dados, etc. (IBM, 2002).

Porém algumas desvantagens podem ser apontadas como, por exemplo, a modelagem é iniciada antecipadamente, sem ter uma atenção devotada suficientemente para a analise de uma situação corrente e desejada, reconhecimento do problema e formulação do problema que são pelo menos tão importantes como a própria solução. Especialmente na prototipação evolucionária o perigo da falha do projeto existe: toda iteração ajusta o protótipo de uma forma que menos da funcionalidade desejada ou funcionalidade supérflua é incorporada dentro do protótipo (IBM, 2002). Um perigo final é que a prototipação pode lidar com entusiasmo do usuário final. O processo de prototipação pode dar ao usuário final a impressão que praticamente qualquer sugestão pode ser implementada, não importa qual estágio do processo de desenvolvimento se está. Além disso, para os usuários não está claro o porquê da demora para entregar a aplicação final depois que uma versão demo do sistema foi exibida.

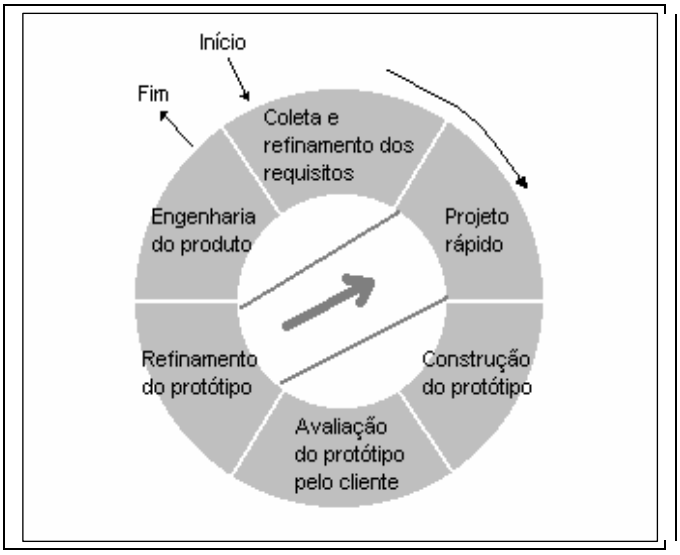

Figura 2. Modelo de processo de Prototipação

2.3 Modelo de Processo de Software Evolucionário

Estudos mostraram que o software, como todos os sistemas complexos, evoluem durante um período de tempo e os requisitos do negócio e do produto mudam frequentemente a medida que o desenvolvimento prossegue dificultando um caminho direto para um produto final (PRESSMAN, 2006).

Os modelos que serão citados à frente são interativos e caracterizam-se pela forma como se desenvolve versões cada vez mais completas do software.

2.3.1 Espiral

O modelo espiral foi desenvolvido para abranger as melhores características tanto do ciclo de vida clássico como da prototipação, acrescentando, ao mesmo tempo, um novo elemento, a análise de riscos que falta a esses paradigmas. O modelo define quatro importantes atividades representadas por quatro quadrantes:

1. Planejamento: determinação dos objetivos, alternativas e restrições.

2. Análise de riscos: análise de alternativas e identificação/resolução de riscos.

3. Engenharia: desenvolvimento do produto no “nível seguinte”.

4. Atualização feita pelo cliente: avaliação dos resultados da engenharia.

Ele usa uma abordagem “evolucionária” à engenharia de software, capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada fase evolutiva. O modelo espiral usa a prototipação como um mecanismo de redução de riscos, mas, o que é mais importante, possibilita que o desenvolvedor aplique a abordagem de prototipação em qualquer etapa da evolução do produto. Ele mantém a abordagem de passos sistemáticos sugerida pelo ciclo de vida clássico, mas incorpora-a numa estrutura iterativa que reflete mais realisticamente o mundo real. O modelo espiral exige uma consideração direta dos riscos técnicos em todas as etapas do projeto e, se

adequadamente aplicado, deve reduzir os riscos antes que eles se tornem problemáticos (Pressman, 2006).

O modelo de ciclo de vida espiral apresentado por Boehm em 1988 combina as características positivas da gerência (documento associado às fases do ciclo) do modelo de cascata com as fases sobrepostas encontradas no modelo incremental e, também, com as versões anteriores de um sistema do modelo de prototipação. O modelo em espiral parte do princípio de que a forma do desenvolvimento de software não pode ser completamente determinada de antemão. (Pressman, 2006).

A prototipação é vista como um meio de redução de riscos, a permitir que se descubram os problemas potenciais antes de se comprometer com um sistema completo. O modelo caracteriza-se como um gerador de modelo de processo. Cada ciclo do modelo em espiral possui quatro atividades principais:

- Elaborar objetivos, restrições e alternativas para entidades de software.
- Avaliar alternativas com relação aos objetivos e restrições, e identificar as principais fontes de riscos.
- Elaborar a definição das entidades de software em um projeto.
- Planejar o próximo ciclo. Abortar um projeto se ele apresentar um alto fator de risco.

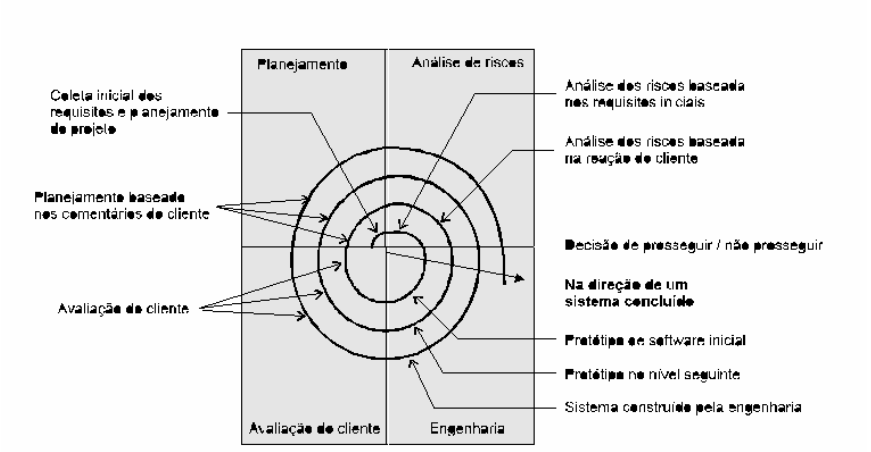

Figura 3. Modelo de processo em Espiral

2.3.2 Incremental

O modelo incremental segundo Pressman (2006) combina elementos do modelo cascata sendo aplicado de maneira interativa. O modelo de processo incremental é interativo igual à prototipagem, mais diferente a prototipagem o incremental tem como objetivo apresentar um produto operacional a cada incremento realizado.

Esse modelo é muito útil quando a empresa não possui mão de obra disponível no momento para uma implementação completa, dentro do prazo estipulado.

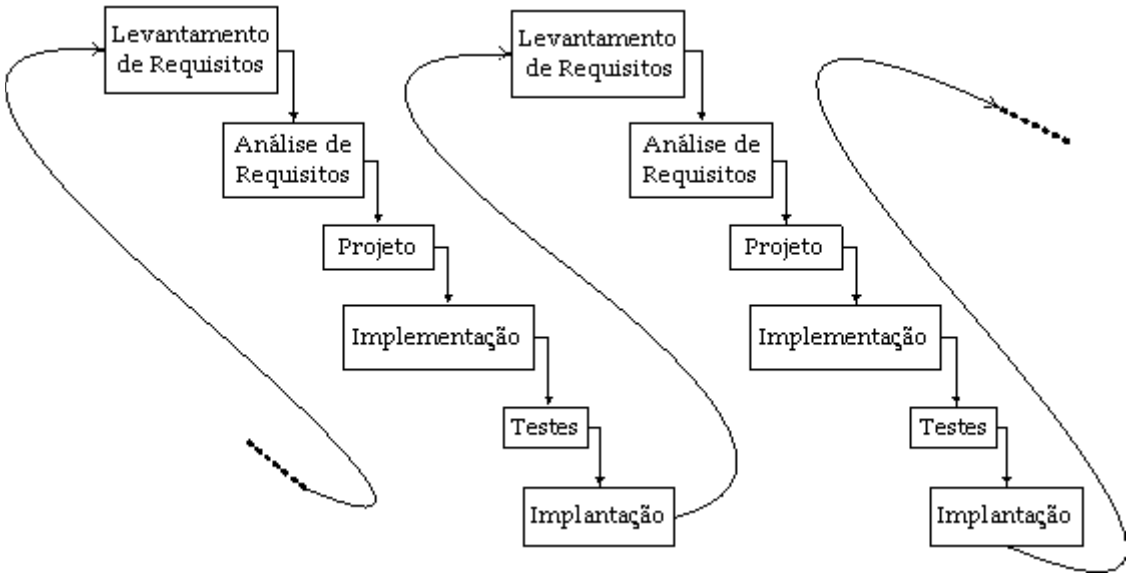

Figura 4. Modelo de processo Incremental

2.3.3 Desenvolvimento baseado em componentes

Desenvolvimento baseado em componentes ou component-based development (CBD) também é conhecido como component-based software engineering (CBSE) ou simplesmente componente de software (BROWN, 1997).

Pressman (2006) não define o que é um componente e restringe-se a dizer que o modelo de desenvolvimento baseado em componentes utiliza paradigma de orientação a objetos baseando-se em uma classe como código reutilizável, ou seja, o componente.

Em orientação a objetos uma classe encapsula dados e algoritmos e este último também pode ser usado para manipular os dados.

Pressman (2006) caracteriza esse modelo como incorporador do modelo espiral com uma abordagem iterativa para a criação de software. Através desta abordagem uma biblioteca de classes é construída com as classes identificadas no desenvolvimento do software e a partir de então toda iteração da espiral deverá verificar o conteúdo da biblioteca que pode ser reutilizado ou identificar se novas classes devem ser inseridas na biblioteca para posterior reuso.

3 SWEBOK

Engenharia de software possui vasto material e várias técnicas para sucesso de software, então o IEEE montou um comitê que se reuniram e montaram um guia chamado SWEBOK que desmembra a engenharia de software em dez áreas de conhecimento que são:

- Requisitos de Software
- Projeto (Design) de Software

- Construção de Software
- Teste de Software
- Manutenção de software
- Gerência de Configuração de Software
- Gerência de Engenharia de Software
- Processos de Engenharia de Software
- Ferramentas e Métodos de Engenharia de Software
- Qualidade de Software

Cada uma das áreas de conhecimento é decomposta em itens, que são os assuntos de conhecimentos a serem considerados pelo projeto. Conforme o projeto deixa claro, os engenheiros de software não devem receber somente os conhecimentos estabelecidos pelas respectivas áreas, mas também outros conhecimentos dos cursos com os quais a Engenharia de Software se relaciona. Foi montado sobre três versões onde a ultima definiu-se como final. São elas:

- Homem de palha
- Homem de pedra (1998-2001)
- Homem de ferro (2003)

Por a engenharia de software ser uma área muito abrangente o SWEBOK não consegue abranger todas as áreas de conhecimento. Um problema que o guia enfrenta é que a engenharia de software continua mudando continuamente inspirada em novas tecnologias e praticas, tornando o guia muitas vezes defasado.

O SWEBOK ainda correlaciona áreas que ele não considera que estejam dentro do escopo da engenharia de software, e sim disciplinas de apoio que são:

- Engenharia da computação;
- Ciência da computação;
- Gerenciamento;
- Matemática;
- Gerência de projetos;
- Gerenciamento da qualidade;
- Ergonomia de software;
- Engenharia de sistemas.

Mesmo com algumas limitações o SWEBOK é um ótimo guia que apóia o desenvolvimento de software, é que pode ser usado como um diferencial e apoio nesse desenvolvimento. (SWEBOK, 2004)

4 Conclusão

A Engenharia de Software é um processo importante no desenvolvimento de Software, mesmo não possuindo uma forma especifica para realização de processos todos os fatores pode funcionar em cada empresa tem que optar pela melhor técnica de desenvolvimento, a melhor opção e mesclarem as técnicas.

O SWEBOK é uma das melhores opções, pois se trata de um guia que foi constituído por vários pesquisadores, portanto é um material completo, porem inacabado, já que novas técnicas continuam surgindo, mas contem as principais funções de desenvolvimento.

Pode-se salientar que a Engenharia de software é um critico fator de sucesso da empresas de desenvolvimento de software, esses conceitos são organizados e agregam viabilidade de produção de software com qualidade, prazo e custos.

Referências Bibliográficas

BROWN, ALAN W., On Components and *Objects: The Fundation of Component-Based Development,* Assessment of Software Tools and Tecnology, Procedings Fifth International Symposium on Proceedings - IEEE, 1997.

IBM; Practicing Object-Oriented Analysis and Design- ERC2.2.; IBM Education and Training; 2002L.

PRESSMAN, ROGER S., *Engenharia de Software- (3ª edição)*, São Paulo, Ed. Makron Books, 1995.

PRESSMAN, ROGER S., *Engenharia de Software- (6ª edição)*, São Paulo, Ed. McGrawHill, 2006.

PETERS, JAMES F., *Engenharia de Software: Teoria e Prática*, Rio de Janeiro, Editora Campus, 2001.

SOMMERVILLE, I. Software Engineering (International Computer Science Series). 5a Edição. Reading: Addison-Wesley, 1995.

SWEBOK 2004, Guide for the Software Engineering Body of Knowledge, 2004 version, IEEE Computer Society, California, EUA.