Engenharia de Software: uma visão geral

Ana Lúcia de Oliveira Tavares[1], Ana Paula Eckel[1], Cateane Scarpa[1], Závia Roselita Vendrame[1]

[1]Curso de Especialização em Engenharia de Projetos de Software
Universidade do Sul de Santa Catarina (UNISUL) – Palhoça, SC – Brasil

{analucia.oliveira, anaeckel, cateane}@gmail.com, zavia@ionics.com.br

Abstract. *This article intends to present the Software Engineering with its main aspects (life cycles and knowledge areas) as an attempt to offer a general vision about this disciplines, for those that are involved in the management and development process can use this paradigm for process and product improvement, with direct benefits for the organizations, its customers and collaborators.*

Resumo. *Este artigo pretende apresentar a Engenharia de Software com seus principais aspectos (ciclos de vida e áreas de conhecimento) numa tentativa de oferecer uma visão geral sobre esta disciplina, para que aqueles que estejam envolvidos no processo de gestão e desenvolvimento, possam vir a utilizar este paradigma para a melhoria do processo e do produto, com benefícios diretos para as organizações, seus clientes e colaboradores..*

1. Introdução

Nos últimos 20 anos, o hardware deixou de ser o item mais caro na implementação de um sistema, enquanto que o custo relacionado ao software cresceu e se tornou o principal item no orçamento da computação. Isso se deve principalmente pela crescente complexidade dos problemas a serem resolvidos pelos softwares. Aliado a isso, alguns problemas inerentes ao processo de desenvolvimento de um software começaram a surgir: as estimativas de prazo e de custo freqüentemente são imprecisas, a produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços e, a qualidade de software às vezes é menos que adequada, ocorrendo freqüentemente a insatisfação do usuário. A chave para se vencer esses problemas e dificuldades acima relatados é a larga utilização de uma abordagem de engenharia ao desenvolvimento de software, aliada a uma contínua melhoria das técnicas e ferramentas, no intuito também de melhorar a produtividade da equipe.

Assim, podemos destacar duas tendências para justificar o uso da Engenharia de Software: primeiro, o software é um item de alto custo e em progressivo aumento; segundo, que os softwares têm um importante papel no bem-estar da sociedade. Dessa forma, a Engenharia de Software assume papel crítico para garantir que tarefas, dados, pessoas e tecnologias estejam apropriadamente alinhadas para produzir um sistema efetivo e eficiente.

2. Processo de desenvolvimento caótico

Em uma conferência em 1968 - *Software Engineering*: *Concepts and Techniques. Proceedings of the NATO Conferences* - Ronald Graham comentou "construímos sistemas como os irmãos Wright construíam aviões - constrói-se de uma vez só, empurra-se para o despenhadeiro, deixa bater e começa tudo outra vez" (NAUR & RANDELL, 1968). Vemos que após 38 anos muito se parece a maneira como são feitos os softwares em grande parte das empresas, principalmente pequenas empresas, onde freqüentemente o programador é também o condutor do projeto.

A crise do software persiste até hoje, erros em estimativas, dificuldade no domínio da área de conhecimento específica do software proposto, especificações obscuras, requisitos mal feitos e mal interpretados, conflitos nos objetivos e mudanças intermináveis e mal controladas são apenas poucos, mas motivos importantes ou suficientes para fazer com que um projeto falhe.

O que percebemos é que após a aprovação do contrato de um novo projeto, os programadores começam o desenvolvimento em cima de um esboço de requisitos do cliente, sem planejamento, sem interatividade com o cliente, ou com esta interatividade fora de controle e não documentada.

As mudanças dos requisitos são feitas de forma informal, via telefone, e-mail e anotações em papéis. A falta de comunicação na equipe de desenvolvimento com a gerência do projeto e o cliente é falha, gerando mais retrabalho (quando normalmente já não existe tempo). A gerência de configurações das versões do software é praticamente nula.

As organizações, tanto públicas como em setores privados, freqüentemente são forçadas a cancelar projetos de software que excederam demais suas estimativas originais de custo e entrega ou estimativas de tempo de conclusão, ou simplesmente não conseguiram alcançar as funcionalidades mínimas desejadas. Os custos de tais cancelamentos de projeto têm sido espantosos nos últimos anos, devendo em parte ao aumento da complexidade das novas tecnologias de software. (EWUSI-MENSAH, 2003). Porém, como disse Jim Johnson, do The Standish Group, "quando um projeto falha, raramente a causa é técnica". Em geral, a causa está na utilização ou não de metodologias de desenvolvimento e em como elas interagem com os envolvidos em um projeto de software (GOULD & FREEMAN, 2006).

A utilização de modelos de processos é fundamental para o bom andamento do projeto e do produto. Visto que o produto de software tem um tempo de desenvolvimento grande se comparado ao tempo de vida útil deste no mercado. Por isso, quanto menor o tempo de desenvolvimento maior a chance dele permanecer no mercado e retornar o investimento feito.

3. Engenharia de Software

John Tukey, especialista em estatística, em 1958 criou o termo software (SWEBOK, 2004). Usando um de seus artigos, citava software como "... rotinas interpretativas cuidadosamente planejadas, compiladores, e outros aspectos da programação automativa...". Segundo Pressman (1995), softwares são “Instruções (programas de computador) que, quando executadas, produzem a função e o desempenho desejados”.

O termo Engenharia de Software surgiu ao final da década de 60, em meio a uma crise no desenvolvimento de software. Nesta época, o interesse por métodos padronizados a serem usados no local de trabalho com o intuito de criar aplicações de qualidade, já haviam se tornado um interesse importante para muitas organizações. Abordagens foram definidas para identificar como a produção de software de qualidade podia ser alcançada nas organizações, visto que a maioria das aplicações foram produzidas naquele tempo sob as regras de desenvolvimento da empresa, nenhuma padronização verdadeira nem qualidade eficiente eram realmente atingíveis para aplicações de software. De fato, a mistura de novas abordagens e recomendações, possivelmente só adicionaram mais confusão e contribuíram ainda mais para à crise na indústria de software (DEEK *et al*, 2005).

Então, ainda nos anos 60, ocorreram duas importantes conferências - onde originou-se o termo Engenharia de Software - realizadas em reconhecimento à esta crise que estava ocorrendo. Foi uma reação a fracassos de projeto; perdas econômicas; atrasos na entrega; mercados competitivos; e uma exigência crescente para funcionalidade, qualidade, e confiabilidade com menor custo possível (DEEK *et al*, 2005).

Mas qual foi o propósito de se usar o termo Engenharia?

O termo Engenharia foi usado justamente para associar o "conceito" Engenharia ao desenvolvimento de software, em outras palavras é ter uma abordagem sistemática, disciplinada e quantificada ao desenvolvimento, operação e manutenção de software (IEEE, 1990).

Segundo ASEE (2000), a engenharia é aplicação de princípios matemáticos e científicos, experiência, julgamento e bom senso para trazer coisas que beneficiam as pessoas. A Engenharia de Software segue este mesmo raciocínio, tendo como objetivo definir e exercitar processos, métodos, ferramentas e ambientes para construção de software que satisfaça necessidades de cliente e usuário dentro de prazos e custos previsíveis.

A Engenharia de Software propõe estratégias de desenvolvimento, chamadas modelos de ciclo de vida de desenvolvimento de software ou modelos de processo. Estes modelos de ciclo de vida, como o nome diz, ajudam o desenvolvimento do início ao fim do projeto.

3.1. Processo Metódico

A Engenharia de Software compreende um conjunto de etapas que envolvem métodos, ferramentas e procedimentos. Essas etapas muitas vezes são citadas como Paradigmas da Engenharia de Software. Um paradigma de Engenharia de Software é escolhido tendo-se como base a natureza do projeto e da aplicação, os métodos e ferramentas a serem usados, os controles e os produtos que precisam ser entregues. Quatro paradigmas têm sido amplamente discutidos e debatidos: Ciclo de Vida Clássico ou Cascata, Prototipação, Modelo Espiral e Técnicas de Quarta Geração.

3.1.1 Ciclo de Vida Clássico

É o modelo mais antigo utilizado. É um método sistemático e seqüencial, em que o resultado de uma fase se constitui na entrada da outra fase. Foi modelado de acordo com

o ciclo da engenharia convencional e, segundo Pressman (1995), abrange as seguintes fases:

- Análise de Engenharia de Sistema: conhecer o sistema e através dele estabelecer os requisitos que devam fazer parte do software;
- Análise de Requisitos de Software: revisão de informações e requisitos e especificações das funcionalidades, desempenho e interface;
- Projeto: constituído de pontos distintos: estrutura de dados, arquitetura do software, detalhes procedimentais e caracterização de interface. Traduz os requisitos levantados para avaliação e qualidade do software antes da codificação;
- Codificação: transformação do projeto para que possa ser interpretado pela máquina. Se o projeto for bem detalhado a codificação torna-se praticamente mecânica;
- Testes: etapa onde são verificados os erros e se o código produz o resultado desejado;
- Manutenção: correção de erros e adaptação do software ao ambiente onde será instalado.

Dentre as vantagens que podemos citar é que o modelo tem uma abordagem disciplinada e dirigida a documentação, tendo como problemas a dificuldade de seguir o fluxo, a determinação de requisitos iniciais (incerteza natural) e o resultado efetivo aparece apenas no final.

Segundo Pressman (1995), "O ciclo de vida clássico continua sendo o modelo procedimental mais amplamente usado pela Engenharia de Software. Embora tenha fragilidade, ele é significativamente melhor do que uma abordagem casual ao desenvolvimento de software".

3.1.2. Prototipação

A prototipação é usada para identificar os requisitos e pode ser a melhor escolha quando o cliente definiu apenas objetivos gerais para o software, sem requisitos detalhados de entrada, processamento ou saída, ou o desenvolvedor pode não ter certeza daquilo que precisa ser desenvolvido.

A prototipação tem início com a coleta dos requisitos, onde desenvolvedor e o cliente definem os objetivos gerais do software. Um projeto é elaborado e a partir dele é construído um protótipo para avaliação, proporcionando ao cliente e desenvolvedor a compreensão daquilo que precisa ser feito.

A vantagem da prototipação é a facilidade para se determinar os requisitos iniciais e a garantia de atingir as necessidades do cliente, enquanto que as desvantagens são a implementação rápida do protótipo comprometido, a determinação do fim do desenvolvimento e tendência a utilizar o protótipo como produto final.

A Prototipação é um ciclo de vida eficiente quando as regras entre cliente e desenvolvedor são definidas no início do processo e há concordância em que o protótipo, por ser um protótipo, deve ser descartado (PRESSMAN, 1995).

3.1.3. Modelo Espiral

Segundo Pressman (1995), este modelo foi desenvolvido pela Engenharia de Software para abranger as melhores características do ciclo de vida clássico e da prototipação, acrescentando ao mesmo tempo, um novo elemento - a análise de riscos - que falta a esses outros paradigmas.

O Modelo Espiral define quatro importantes atividades:

- Planejamento: determinação dos objetivos, alternativas e restrições;
- Análise de Riscos: análise de alternativas e identificação/resolução dos riscos;
- Engenharia: desenvolvimento do produto no "nível seguinte";
- Avaliação feita pelo cliente: avaliação dos resultados da engenharia.

Com cada iteração ao redor da espiral (iniciando-se ao centro e avançando para fora), versões progressivamente mais completas do software são construídas. Durante o primeiro giro ao redor da espiral, os objetivos, as alternativas e as restrições são definidos e os riscos são identificados e analisados. Se a análise dos riscos indicar que há incertezas nos requisitos, a prototipação pode ser usada no quadrante da engenharia para ajudar tanto o desenvolvedor quanto o cliente.

O modelo espiral usa a prototipação como um mecanismo de redução de riscos e mantém uma abordagem de passos sistemáticos sugerida pelo ciclo de vida clássico e ainda incorpora uma estrutura iterativa que reflete mais realisticamente o mundo real.

3.1.4. Técnicas de Quarta Geração

O paradigma Técnicas de Quarta Geração (4GT) da Engenharia de Software concentra-se na capacidade de se especificar software a uma máquina em um nível que esteja próximo à linguagem natural ou de se usar uma notação que comunique uma função significativa.

O 4GT inicia-se com uma etapa de coleta de requisitos. Idealmente, o cliente descreveria os requisitos, e estes seriam diretamente traduzidos num protótipo operacional. Mas isso não é idealmente possível. O cliente pode não ter certeza daquilo que é exigido, pode ser ambíguo ao especificar fatos que são conhecidos e pode ser inviável especificar as informações de maneira que uma ferramenta 4GT possa receber. Para pequenas aplicações, talvez seja possível passar diretamente da etapa de coleta das exigências para a implementação, utilizando uma linguagem de quarta geração (4GL).

Para transformar a implementação de uma 4GT num produto, o desenvolvedor deve realizar testes cuidadosos, desenvolver uma documentação significativa e executar todas as demais atividades de "transição" que também são exigidas em outros paradigmas da Engenharia de Software.

As técnicas de quarta geração já se tornaram uma parte importante do desenvolvimento de software na área de aplicação de sistemas de informação e a demanda de software continuará em ascensão, porém o software produzido com métodos e paradigmas convencionais contribuirá cada vez menos para todo o software desenvolvido. As técnicas de quarta geração preencherão a lacuna (PRESSMAN, 1995).

3.1.5. Combinando paradigmas

Os paradigmas podem e devem ser combinados de forma que as potencialidades de cada um possam ser obtidas num único projeto. Os paradigmas da Engenharia de Software podem ser combinados num único esforço de desenvolvimento de software. O trabalho inicia-se com a definição dos objetivos, alternativas e restrições. A partir desse ponto qualquer caminho pode ser seguido (PRESSMAN, 1995).

3.2 Áreas de Conhecimento segundo o SWEBOK

O SWEBOK é um guia de uso e aplicação das melhores práticas de Engenharia de Software, informado, sensato e razoável. Ele foi desenvolvido com conhecimentos recolhidos no período de 4 décadas e revisado por inúmeros profissionais de diversos países envolvidos com a Engenharia de Software. Seu principal objetivo foi estabelecer um conjunto apropriado de critérios e normas para a prática profissional da Engenharia de Software. Neste guia, a Engenharia de Software foi dividida em 10 áreas de conhecimentos, também conhecidas por KAs (*Knowledge Areas*), que serão descritas na seqüência (SWEBOK, 2004).

3.2.1 Requisitos de Software

Os requisitos expressam a necessidade e restrições colocadas sobre o produto de software que contribuem para a solução de algum problema do mundo real. Esta área envolve elicitação, análise, especificação e validação dos requisitos de software (SWEBOK, 2004).

Segundo Breitman e Sayão (2005), os requisitos de software são classificados em:

- Requisitos funcionais: correspondem a funcionalidade do software e o que o sistema deve fazer;
- Requisitos não funcionais: expressam restrições e características que o software deve atender ou ter;
- Requisitos inversos: definem estados e situações que nunca podem acontecer.

Pressman (2002) cita que “se você não analisa, é altamente provável que construa uma solução de software muito elegante que resolve o problema errado”. Esta atitude pode resultar em perda de tempo e dinheiro, pessoas frustradas e clientes insatisfeitos.

3.2.2 *Design* de Software

Segundo o SWEBOK (2004), esta área envolve definição da arquitetura, componentes, interfaces e outras características de um sistema ou componente. Visualizado como um processo, esta área é uma etapa do ciclo de vida da Engenharia de Software, onde os requisitos são analisados para produzir uma descrição da arquitetura do software.

Para Pressman (2002), *design* de software "é um processo iterativo através do qual os requisitos são traduzidos num documento para construção do software."

3.2.3 Construção de Software

Refere-se a implementação do software, verificação, testes de unidade, testes de integração e *debugging*. Esta área está envolvida com todas as áreas de conhecimento, porém, está fortemente ligada às áreas de *Design* e Teste de Software, pois o processo de construção abrange significativamente estas duas áreas (SWEBOK, 2004).

As áreas correlatas à construção de software, segundo o SWEBOK (2004) são:

- Fundamentos: minimizar a complexidade, antecipar mudanças, construir para verificar e padrões de construção;
- Gerenciamento da construção: modelos, planejamento e métricas;
- Considerações práticas: *design*, linguagens, codificação, testes, reutilização, qualidade e integração.

É importante que as funcionalidades do software sejam testadas durante todo o processo de desenvolvimento, não deixando apenas para a etapa de testes.

3.2.4 Teste de Software

Teste de software é uma atividade executada para avaliar a qualidade do produto, buscando identificar os defeitos e problemas existentes (SWEBOK, 2004).

Para Pressman (2002), "teste de software é um elemento crítico da garantia de qualidade de software e representa a revisão final da especificação, projeto e geração de código."

Segundo Coelho (2005) os tipos de teste são:

- Teste funcional: verifica as regras de negócio, condições válidas e inválidas;
- Teste de recuperação de falhas: as falhas são provocadas diversas vezes a fim de verificar a eficiência da recuperação;
- Teste de desempenho: verifica o tempo de resposta e processamento para configurações diferentes;
- Teste de segurança e controle de acesso: verifica a funcionalidade dos mecanismos de proteção de acesso e de dados;
- Teste de *interfaces* com o usuário: verifica navegação, consistência e padrões;
- Teste de volume: verifica até aonde o software suporta.

A etapa de teste de software é relevante para que os erros possam ser encontrados e corrigidos antes que o software seja entregue ao cliente.

3.2.5 Manutenção de Software

Esta área de conhecimento é definida como a totalidade das atividades requeridas para fornecer suporte custo-efetivo a um sistema de software, que pode ocorrer antes ou depois da entrega. Antes da entrega do software são realizadas atividades de planejamento e depois, modificações são feitas com o objetivo de corrigir falhas, melhorar o desempenho ou adaptá-las a um ambiente externo (SWEBOK, 2004).

Os tipos de modificações durante a fase de manutenção, segundo Pressman (2002) são:

- Manutenção corretiva: modifica o software para corrigir erros;
- Manutenção adaptativa: altera o software para acomodar mudanças no seu ambiente externo;
- Manutenção perfectiva: aprimora o software (solicitações do cliente);
- Manutenção preventiva (reengenharia): modifica o software a fim de torná-los mais fáceis de serem corrigidos, adaptados e melhorados.

Em torno de 60% do esforço despendido por uma organização de desenvolvimento é referente à manutenção de software. Este percentual continua crescendo à medida que mais softwares são produzidos. Manutenção de software não é só concertar erros. Apenas 20% do trabalho de manutenção é referente à correção de falhas e, os outros 80% refere-se à adaptações ao ambiente externo e a melhorias solicitadas pelos usuários (PRESSMAN, 2002).

3.2.6 Gerenciamento de Configuração de Software

Conforme Cunha *et al* (2004), o GCS (Gerenciamento de Configuração de Software) é um processo que provê recursos para a identificação, controle da evolução e auditagem dos artefatos de software criados durante o desenvolvimento do projeto. De grosso modo, é o controle de versões do software.

A finalidade do GCS é estabelecer e manter a integridade dos produtos de software durante todo seu ciclo de vida (SWEBOK, 2004):

- Identificar a configuração do software em um dado momento;
- Controlar sistematicamente as mudanças de configuração;
- Manter a integridade e a rastreabilidade da configuração ao longo do ciclo de vida do software;
- Controlar a integridade dos artefatos compostos, levando em conta cada um dos componentes do software;
- Registrar e controlar o estado do processo de alteração.

Porém, sua aplicação em empresas de desenvolvimento de software é complexa, e às vezes inviabilizada pelos gastos. Uma forma de contornar isso é desenvolver uma metodologia de gerenciamento de configuração que leve em conta somente aspectos relevantes para a realidade da empresa, descartando aqueles que são menos utilizados.

3.2.7 Gerenciamento de Engenharia de Software

Segundo SWEBOK (2004), Gerenciamento de Engenharia pode-se definir como a aplicação das atividades de gerenciamento: planejamento, coordenação, medição, monitoração, controle e documentação, garantindo que o desenvolvimento e a gerência de software sejam sistemáticos, disciplinados e qualificados.

O Gerenciamento de Engenharia é tratado sob dois aspectos:

- Engenharia de Processo: refere-se às atividades empreendidas para geração de políticas, padrões e objetivos organizacionais consistentes;
- Engenharia de Mensuração: refere-se à atribuição de valores e rótulos às atividades referentes à Engenharia de Software.

O gerenciamento de processo e a mensuração são importantes em todas as áreas de conhecimento, mas o Gerenciamento de Engenharia trata esses aspectos de forma mais direta. Um grande aliado desta área de conhecimento é o Gerenciamento de Projetos, que pode ser visto a seguir.

3.2.7.1 Gerenciamento de Projetos de Software

Projeto é um empreendimento temporário, de elaboração progressiva e com o objetivo de criar um produto ou serviço único (PMBOK, 2004).

- Temporário: o projeto possui início e fim bem definidos e pode ser de curta ou longa duração. O projeto chega ao fim quando os seus objetivos são atingidos;
- Elaboração progressiva: o desenvolvimento ocorre em etapas e continua por incrementos;
- Produto ou serviço único: cada projeto é exclusivo.

Segundo o PMBOK (2004), o gerenciamento de projetos "é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender aos seus requisitos". É realizado através de cinco grupos de processos: iniciação, planejamento, execução, monitoramento e controle, e encerramento.

Para Pressman (2002), “a gestão do projeto envolve o planejamento, a monitoração e controle do pessoal, processo e eventos que ocorrem à medida que o software evolui de um conceito preliminar para uma implementação operacional".

O gerenciamento de projetos auxilia as organizações a atenderem as necessidades de seus clientes, padronizando tarefas do dia a dia e reduzindo o número de tarefas, que muitas vezes são esquecidas (PMI-SC, 2006).

Por fim, gerenciar projetos é manter o equilíbrio entre escopo, qualidade, custos, recursos e tempo (PMBOK, 2004). É papel do gerente de projetos avaliar os riscos e impactos associados a qualquer mudança em um desses fatores.

3.2.8 Engenharia de Processo de Software

A Engenharia de Processo pode ser interpretada como uma visão geral sobre questões relacionadas ao processo de Engenharia de Software, principalmente as atividades relacionadas à definição, implementação, avaliação, mensuração, gerenciamento, mudanças e melhorias do processo de ciclo de vida de software (SWEBOK, 2004).

O objetivo da Engenharia de Processo de Software é implementar processos novos e melhores, seja no escopo individual, de projeto ou organizacional.

3.2.9 Ferramentas e Métodos de Software

Ferramentas de desenvolvimento de software são ferramentas criadas para auxiliar no ciclo de vida do software. Essas ferramentas normalmente automatizam algumas atividades do processo de desenvolvimento, fazendo com que o analista concentre-se nas atividades que exigem maior trabalho intelectual (SWEBOK, 2004).

Métodos de Engenharia de Software impõe estrutura sobre a atividade de desenvolvimento e manutenção de software com o objetivo de torná-la sistemática e mais propensa ao sucesso (SWEBOK, 2004).

Esta área de conhecimento tem como objetivo pesquisar ferramentas e métodos que aumentem a produtividade dos desenvolvedores enquanto reduzem a ocorrência de falhas no desenvolvimento (FERNANDES, 2003).

3.2.10 Qualidade de Software

A qualidade de software não pode ser entendida como perfeição. Qualidade é um conceito multidimensional, realizado por um conjunto de atributos, representando vários aspectos relacionados ao produto: desenvolvimento, manutenção e uso. Qualidade é algo factível, relativo, dinâmico e evolutivo, adequando-se ao nível dos objetivos a serem atingidos (SIMÃO, 2002).

Um dos principais objetivos da Engenharia de Software é melhorar a qualidade dos produtos de software, ela visa estabelecer métodos e tecnologias para construir produtos de software de qualidade dentro dos limites de tempo e recursos disponíveis. A qualidade de software está diretamente ligada com a qualidade do processo através do qual o software é desenvolvido, portanto, para se ter qualidade em um produto de software é necessário ter um processo de desenvolvimento bem definido, que deve ser documentado e acompanhado (SWEBOK, 2004).

A avaliação da qualidade de produtos de software normalmente é feita através de modelos de avaliação de qualidade. Esses modelos descrevem e organizam as propriedades de qualidade do produto em avaliação. Os modelos de avaliação mais aceitos e usados no mercado são:

- CMMI (*Capability Maturity Model Integration*), proposto pelo CMM (*Capability Maturity Model*);
- Norma ISO/IEC 9126, proposta pela ISO (*International Organization for Standardization*).

As organizações desenvolvedoras desses modelos de qualidade fornecem selos de qualidade para as empresas que se submetem à avaliações e estiverem dentro dos padrões propostos. Esses selos são muito valorizados pelas empresas que compram software, e representam um diferencial competitivo no mercado. Porém, nem todas as empresas têm condições financeiras de bancar os custos de uma aquisição de um selo de qualidade, pois implantar um processo de qualidade em uma empresa envolve custos elevados. Contudo, é possível implantar boas práticas e desenvolver um processo de desenvolvimento organizado adaptando modelos de desenvolvimento conhecidos, despendendo menos recursos e provendo um mínimo de sistematização no desenvolvimento de software, a fim de se ter maior qualidade.

4. Conclusões

Atualmente o desenvolvimento de sistema encontra-se em um panorama bastante preocupante, pois a maioria das organizações envolvidas no processo, não estão utilizando nenhum dos paradigmas da Engenharia de Software, e deixando de lado um forte conjunto de atividades que assegurariam a qualidade do produto. O uso da Engenharia de Software é uma tarefa difícil e extensa, recheada de métodos, que tornam sua utilização uma atividade para especialistas. Entretanto, não se deve relegar sua aplicação.

Os softwares de qualidade precisam de vários passos para o seu desenvolvimento, como uma detalhada análise de requisitos, escolha de um modelo adequado, hardware e software para o auxílio do desenvolvimento e projeto de *interface* bem definidos.

O intuito deste artigo foi despertar a comunidade de desenvolvimento de sistemas para a Engenharia de Software. Procurou-se oferecer um breve panorama sobre como a disciplina deve ser aplicada, seus recursos, paradigmas e métodos. Tentou-se demonstrar as vantagens em se utilizar a Engenharia de Software no processo de desenvolvimento de um sistema, enfatizando a melhoria da qualidade dos processos e produtos gerados, com o objetivo final de melhorar a qualidade software.

Diante do cenário apresentado no artigo, as práticas de Engenharia de Software deveriam ser aplicadas em todas as empresas, tornando cada vez mais eficiente o processo de desenvolvimento e aumentando a qualidade do produto. Vale ressaltar que todo o processo de Engenharia de Software tem um custo envolvido. As empresas podem e devem, adaptar os processos de desenvolvimento a sua realidade, mas o fundamental é ter um processo bem definido.

5. Referências

ASEE. **American Society for Engineering Education.** 2000.

AZEVEDO, R. S. N. **Um Sistema de Gestão da Qualidade para Micro, Pequenas, e Médias Empresas de software a partir da ISO 9001 e ISO/IEC.** 2005. Dissertação (Mestrado em Informática Aplicada) - Programa de Pós-Graduação em Informática, Fundação Educacional Edson de Queiroz, Universidade de Fortaleza, Fortaleza.

BREITMAN, K; SAYÃO, M. **Gerência de Requisitos. XIX Simpósio Brasileiro de Engenharia de Software: Mini curso.** Uberlândia, 2005. Disponível em: <http://www.sbbd-sbes2005.ufu.br/arquivos/Gerencia_Requisitos2.pdf> Acesso em: 03 dez. 2006.

COELHO, R. **Teste de Software. XIX Simpósio Brasileiro de Engenharia de Software: Mini curso.** Uberlândia, 2005. Disponível em: <http://www.sbbd-sbes2005.ufu.br/arquivos/minicursoSBES2005_TestesSoftware.pdf> Acesso em: 03 dez. 2006.

CUNHA, J. R. D.; PRADO, A. F.; SANTOS, A. C.; Souza R. M. N. **Uma Abordagem para o Processo de Gerenciamento de Configuração de Software.** Revista Eletrônica de Sistemas de Informação. 4ª Edição. 2004. Disponível em: <http://www.inf.ufsc.br/resi/edicao04/artigo05.pdf >. Acesso em: 28 nov. 2006.

DEEK, F. P.; MCHUGH, J. A.; ELJABIRI, O. M. **Strategic software engineering: an interdisciplinary approach.** Auerbach, 2005, pp.173-188.

EWUSI-MENSAH, K. **Software Development Failures: Anatomy of Abandoned Projects.** Cambridge, Mass.: MIT Press, 2003.

FERNANDES, J. H. C. **Qual a Prática do Desenvolvimento de Software?** Revista Ciência e Cultura. Vol 55. Nº2. Abril/Maio/Junho de 2003. pp 29-33. Editora da Sociedade Brasileira para o Progresso da Ciência. ISSN 0009-6725. Disponível em: <http://www.dimap.ufrn.br/~jorge/textos/papers/PraticaDeSoftware.pdf>. Acesso em: 29 nov. 06.

FERREIRA, M. P. **Desenvolvimento de software alinhado aos objetivos estratégicos do negócio: proposta de uma metodologia.** 2002. 149 f. Dissertação (Mestrado em Engenharia de Produção) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Produção. Florianópolis. Disponível em <http://150.162.90.250/teses/PEPS2492.pdf>. Acesso em: 30 nov. 2006.

GOULD, M.; FREEMAN, R. **The Art of Project Management: A Competency Model For Project Managers.** Boston University Corporate Education Center. Disponível em: <http://emarketing.propoint.com/propoint/pdf/PPI_TheArtOfProjectManagement.pdf>. Acesso em: 06 dez. 2006.

IEEE. **Standard Glossary of Software Engineering Terminology, IEEE std 610.12-1990.** 1990. Disponível em: <http://www.swen.uwaterloo.ca/~bpekilis/public/SoftwareEngGlossary.pdf>. Acesso em: 26 nov. 2006

ISO. **ISO/IEC 9126. Software engineering – Product quality – Part 1: Quality model**. 2001. Disponível em: <http://www.iso.org/>. Acesso em: 26 nov. 2006.

NAUR, P.; RANDELL, B. E.. **Software Engineering: Report on a Conference Sponsored by the NATO Science Committee.** Technical report, NATO, pp. 31, Garmisch, Germany, 7th to 11th October 1968.

PMBOK. **Um guia do Conjunto de Conhecimentos em Gerenciamento de Projetos.** Uma Norma Nacional Americana ANSI/PMI 99-001-2004. 3. ed.

PMI-SC. **Project Manager Institute. Gerenciamento de Projetos: GP e o sucesso de organizações.** Joinville – SC. Disponível em: <http://www.pmisc.org.br>. Acesso em: 29 nov. 2006.

PRESSMAN, R. S. **Engenharia de Software.** São Paulo: Makron Books, 1995.

PRESSMAN, R. S. **Engenharia de Software.** 5. ed. Rio de Janeiro: McGraw-Hill, 2002.

SIMÃO, R. P. S. **Características de Qualidade para Componentes de Software.** 2002. Dissertação (Mestrado em Informática Aplicada) - Programa de Pós-Graduação

em Informática, Fundação Educacional Edson de Queiroz, Universidade de Fortaleza, Fortaleza.

SWEBOK. **Guide to the Software Engineering Body of Knowledge**. 2004 Version. A project of the IEEE Computer Society Professional Practices Committee. Disponível em: <http://www.swebok.org/>. Acesso em: 15 nov. 2006.