Projeto de Sistemas de Informação

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico

Engenharia de software

Responsável pelo Conteúdo:
Profa. Ms. Fábio Peppe Beraldo

Revisão Textual:
Prof. Ms. Claudio Brites

UNIDADE Engenharia de software

- Introdução
- O profissional da engenharia de software
- Certificação
- Impacto na globalização
- O processo de desenvolvimento de software

Objetivo de APRENDIZADO

Falaremos nesta unidade de Engenharia de Software, sua conceitualização, detalhes da profissão e modelos de trabalho. Nesse contexto, a unidade apresenta uma introdução ao "o que é" e como funciona em alguns países a Engenharia de Software, além de diversos modelos e métodos de trabalho no desenvolvimento de softwares.

Este guia de estudo demonstra a forma de organizar suas atividades para cumprir os propósitos da disciplina *online* objetiva salientar algumas informações importantes sobre esta Unidade, bem como sobre o estudo *online* em si.

Primeiramente, é necessária a conscientização de que a exigência – tanto de estudo quanto de avaliação – da disciplina *online* é, no mínimo, a mesma que a presencial.

Assim, é necessário organizar seu tempo para ler atentamente o material teórico e assistir os vídeos, se houver, inclusive do material complementar.

Organize-se também de forma a não deixar para o último dia a realização das atividades de sistematização (pontuada) e fórum (não pontuado). Podem ocorrer imprevistos e, encerrada a Unidade, encerra-se a possibilidade de obter a nota relativa a cada atividade.

Para ampliar seu conhecimento, bem como aprofundar os assuntos discutidos, pesquise, leia e consulte os livros indicados nas Referências e/ou na Bibliografia.

As Referências estão indicadas ao final dos textos de conteúdo de cada Unidade.

Contextualização

Engenharia de Software é o estudo e a aplicação de engenharia no design, desenvolvimento e manutenção de softwares. A engenharia de software trata os programadores como praticantes de uma abordagem de engenharia alinhada com as conotações de previsibilidade, precisão, risco mitigado e profissionalismo.

Nesta unidade da disciplina Projeto de Sistemas de Informação, teremos uma visão de como a engenharia de software funciona, seu campo profissional e seus modelos.

Introdução

Engenharia de Software é o estudo e a aplicação de engenharia no design, desenvolvimento e manutenção de softwares. As definições mais comuns para a engenharia de software abrangem:

- Aplicação de uma abordagem sistemática e disciplinada, quantificável para o desenvolvimento, operação e manutenção de softwares;
- Preocupação com todos os aspectos da produção de software;
- O uso de princípios sólidos de engenharia, a fim de obter economicamente softwares que sejam confiáveis e funcionem de forma eficiente em máquinas reais;
- Uma gama de atividades que antes eram chamadas de programação de computadores e análise de sistemas;
- Aspectos da prática da programação de computadores, ao contrário da teoria de programação de computadores, que é chamada de informática;
- Abordagem específica para a programação de computadores, que insiste para que seja tratada como uma disciplina de engenharia em vez de uma arte ou um ofício, e defende a codificação das práticas recomendadas.

A engenharia de software trata os programadores como praticantes de uma abordagem de engenharia alinhada com as conotações de previsibilidade, precisão, risco mitigado e profissionalismo. Essa perspectiva levou a pedidos de licenciamento, certificação e de corpos de conhecimento codificados como mecanismos de amadurecimento do campo. Até mesmo o termo "Software de Artesanato ou Feito a Mão" foi proposto por um corpo de desenvolvedores de softwares como uma alternativa que enfatizaria as habilidades de codificação e de responsabilização dos próprios desenvolvedores de softwares.

Os conceitos de "engenharia de software" e "manutenção de software" são considerados hoje meras analogias ao fenômeno da engenharia de software, destinados para a ciência da computação.

O profissional da engenharia de software

Os Estados Unidos iniciou em 2013 exames de engenharia, permitindo assim que os engenheiros de software fossem licenciados e reconhecidos. O licenciamento é obrigatório e, por enquanto, tem sido discutido de forma bastante controversa. Em algumas partes dos EUA, como o Texas, o uso do termo *engenheiro* é regulamentado por lei e reservado apenas para pessoas que tenham uma licença de engenheiro professional. O IEEE informa que a licença de engenheiro profissional não é necessária a menos que o indivíduo trabalhe para o público. Licenças de engenheiro profissional são específicas dependendo do país.

A maioria dos engenheiros de software trabalha como empregado ou contratado. Os engenheiros de software podem atuar em empresas, organizações, agências governamentais, civis, militares e sem fins lucrativos. Alguns engenheiros de software trabalham de forma autônoma como *freelancers*. Algumas organizações têm especialistas para realizar cada uma das tarefas do processo de desenvolvimento de software; outras organizações exigem engenheiros de software para fazer muitas delas ou todas. Em grandes projetos, as pessoas podem se especializar em apenas um papel. Em projetos pequenos, as pessoas podem preencher várias ou todas as funções ao mesmo tempo. Especializações incluem: a indústria (analistas, arquitetos, desenvolvedores, testadores, suporte técnico, analistas, gerentes de middleware) e a academia (educadores, pesquisadores).

A maioria dos engenheiros de software e programadores trabalham 40 horas por semana, mas cerca de 15% dos engenheiros de software e 11% dos programadores trabalharam mais de 50 horas por semana. Lesões nessas ocupações são raras, mas, como outros trabalhadores que passam longos períodos em frente a um terminal de computador digitando, engenheiros e programadores são suscetíveis a fadiga ocular, desconforto nas costas, na mão e no punho e problemas como a síndrome do túnel do carpo.

Certificação

O Instituto de Engenharia de Software oferece certificações sobre temas específicos como: segurança, melhoria de processos e arquitetura de software. Empresas como Apple, IBM, Microsoft e outras também patrocinam os seus próprios exames de certificação. Muitos programas de certificação de TI são orientados para tecnologias específicas e geridos pelos fornecedores dessas tecnologias. Esses programas de certificação são adaptados para as instituições que empregam pessoas que usam essas tecnologias.

Desde 2006, o IEEE certificou mais de 575 profissionais de software com o *Software Development Certified Professional* (SDCP). Em 2008, eles adicionaram uma certificação de nível básico conhecida como *Certified Software Development Associate* (CSDA).

No Reino Unido, a British Computer Society desenvolveu uma certificação profissional reconhecida legalmente chamada *Chartered IT Professional* (CITP), disponível para membros qualificados (MBCS). Os engenheiros de software puderam aderir ao Instituto de Engenharia e Tecnologia e assim se beneficiarem do status de *Engenheiro Chartered*. No Canadá, a Sociedade de Processamento de Informações desenvolveu uma certificação profissional reconhecida legalmente, chamada de Sistemas de Informação Profissional (ISP). Em Ontário, Canadá, engenheiros de software se formam a partir de programas credenciados: *Engenharia Accreditation Board* (CEAB), *Professional Engineers Ontário* (PEO), Exame de Prática Profissional (PPE).

Um outro tipo de problema enfrentado é que a grande maioria dos profissionais que trabalham no campo possuem diploma em Ciências da Computação, não em Engenharia de Software. Dado o difícil caminho de certificação para os titulares de graus não-engenheiros, estes nunca se preocuparam em buscar a licença.

Impacto na globalização

O impacto inicial da terceirização e o custo relativamente mais baixo de recursos humanos internacionais no desenvolvimento de países do terceiro mundo levaram a uma migração maciça de atividades de desenvolvimento de software de empresas na América do Norte e na Europa para a Índia e, mais tarde para China, Rússia e outros países em desenvolvimento. Tal abordagem teve algumas falhas, principalmente a diferença de fuso horário que impediu a interação entre clientes e desenvolvedores, fora a grande transferência de trabalho. Isso teve impacto negativo sobre muitos aspectos da profissão de engenharia de software, por exemplo: alguns alunos no mundo desenvolvido evitaram estudar qualquer área relacionada à engenharia de software por causa do medo de *offshore outsourcing* (importação de produtos de softwares ou serviços de outros países) e de serem substituídos por trabalhadores de com vistos estrangeiros. Embora as estatísticas não mostrem atualmente uma ameaça à própria engenharia de software, as carreiras em programação de computadores não parecem ter sido afetadas.

No entanto, a capacidade de alavancar recursos *offshore* de forma inteligente e perto da costa por meio do fluxo de trabalho tem melhorado a capacidade operacional global de muitas organizações – quando os norte-americanos estão saindo do trabalho, os asiáticos estão apenas chegando para trabalhar; quando os asiáticos estão saindo do trabalho, os europeus estão chegando para trabalhar –, isso fornece uma capacidade contínua de ter supervisão humana em processos críticos de negócios 24 horas por dia, sem pagar compensação de horas extras ou interromper uma necessidade humana fundamental: os padrões de sono.

Enquanto o *outsourcing* global tem várias vantagens, o desenvolvimento pode ter sérias dificuldades resultantes da distância entre os desenvolvedores – devido aos elementos-chave dessa distância, identificados como geográficos, temporais, culturais e de comunicação (que inclui o uso de diferentes línguas e dialetos em locais diferentes). A pesquisa foi realizada em áreas de desenvolvimento global de software ao longo dos últimos 15 anos e um extenso corpo de trabalho relevante foi publicado, o qual destaca os benefícios e problemas associados com a atividade complexa – tal qual acontece com outros aspectos da pesquisa em engenharia de software que está em curso nesta e em áreas afins.

O processo de desenvolvimento de software

A Engenharia de software Assistida por Computador (CASE) é uma ferramenta utilizada para apoiar as atividades do processo de software. No entanto, devido à grande diversidade de processos de software para diferentes tipos de produtos, a eficácia de ferramentas CASE é limitada. Não há uma abordagem ideal para o processo de criação do software ainda não desenvolvido. Algumas atividades fundamentais, como a especificação de software, design, validação e manutenção, são comuns a todas as atividades do processo.

Em Síntese

Um modelo de processo de software é uma abstração do processo de software, é também chamado de paradigma de processo. Existem vários modelos de processos, sendo os mais utilizados: o modelo em cascata (*waterfall*), modelo de iterativo, modelo espiral, modelo V e modelo Big Bang. Estes são largamente utilizados na prática corrente da engenharia de software – para grandes sistemas, eles são utilizados em conjunto.

O modelo Waterfall

Foi o primeiro modelo SDLC a ser usado amplamente em engenharia de software para garantir o sucesso do projeto. No modelo waterfall, todo o processo de desenvolvimento de software é dividido em fases distintas. No modelo em cascata, normalmente, o resultado dos atos de uma fase são a entrada para a próxima fase, sequencialmente.

PRÓS	CONTRAS
• Simples e fácil de entender e usar; • Fácil de gerenciar, por conta da rigidez do modelo – cada fase tem produtos específicos e um processo de revisão; • As fases são processadas e completa-se uma de cada vez; • Funciona bem para projetos menores onde as exigências são muito bem compreendidas; • Etapas claramente definidas; • Marcos bem entendidos; • Fácil de organizar tarefas; • Processos e resultados estão bem documentados.	• Nenhum software de trabalho é produzido até o final durante o ciclo de vida; • Quantidade elevada de risco e incerteza; • Não é um bom modelo para projetos complexos e de orientação a objetos; • Modelo pobre para projetos de duração longa e contínua; • Não é adequado para projetos onde os requisitos estão em risco de moderado a alto de mudar. Assim, o risco e a incerteza são altas com esse modelo de processo; • É difícil medir o progresso em estágios; • Não é possível acomodar a mudança de requisitos; • Nenhum software de trabalho é produzido até o final do ciclo de vida; • Ajustar escopo durante o ciclo de vida pode terminar um projeto; • A integração é feita como um “big bang”, no final, o que não permite identificar cedo qualquer gargalo, desafio tecnológico ou de negócio.

O modelo iterativo

O processo iterativo começa com a simples implementação de um subconjunto dos requisitos de software e de forma iterativa se aumenta as versões, evoluindo até que o sistema completo seja implementado. A cada iteração, as modificações de design são feitas e novas capacidades funcionais são adicionadas. A ideia básica por trás desse método é desenvolver um sistema por meio de repetidos ciclos (iterativos) e em pequenas porções por vez (incremental).

O desenvolvimento iterativo e incremental é uma combinação de ambos: projeto iterativo ou método iterativo e modelo de compilação incremental para o desenvolvimento. Durante o desenvolvimento de software, mais do que uma repetição do ciclo de desenvolvimento de software pode estar em curso ao mesmo tempo.

A chave para o sucesso do uso de um ciclo de vida de desenvolvimento de software iterativo é a rigorosa validação de requisitos, e a verificação e a análise de cada versão do software dentro desses requisitos em cada ciclo do modelo. À medida que o software evolui através de ciclos sucessivos, os testes devem ser repetidos e estendidos para que se verifique cada versão do software.

PRÓS	CONTRAS
• Algumas funcionalidades do trabalho podem ser desenvolvidas de forma rápida e logo no início do ciclo de vida; • Os resultados são obtidos no início e periodicamente; • O desenvolvimento paralelo pode ser planejado; • O progresso pode ser medido; • Menos caro para alterar o escopos/requisitos; • Testes e depurações durante uma menor iteração é fácil; • Os riscos são identificados e resolvidos durante a iteração, e cada iteração é um marco de fácil gestão; • Mais fácil de gerenciar riscos – a parte alta do risco é feita primeiro; • Com cada produto operacional, algum incremento é entregue; • Problemas, desafios e riscos identificados a partir de cada incremento podem ser utilizados / aplicados para o próximo incremento; • A análise de risco é melhor; • Ele suporta mudanças de requisitos; • O tempo de operação inicial é menor; • Mais adequado para projetos grandes e de missão crítica; • Durante o ciclo de vida de software, é produzido precoce, que facilita a avaliação do cliente e feedback.	• Podem ser necessários mais recursos; • Embora o custo de mudança seja menor, não é muito adequado para a evolução das necessidades. • É necessário mais atenção da administração; • Na arquitetura de sistema ou de design, problemas podem surgir porque nem todos os requisitos estão reunidos no início de cada ciclo de vida; • Definindo incrementos, pode-se exigir definições do sistema completo; • Não é adequado para projetos menores; • Complexidade de gerenciamento é maior; • O fim do projeto não pode ser conhecido – o que é um risco; • São necessários recursos altamente qualificados para análise de risco; • O progresso do projeto é altamente dependente da fase de análise de risco.

O modelo espiral

Em Síntese

É uma combinação do modelo de processo de desenvolvimento iterativo e sequencial e o modelo de desenvolvimento, ou seja, o modelo em cascata linear com alta ênfase em análise de risco. Ele permite as versões incrementais do produto, ou o refinamento progressivo por meio de cada iteração ao redor da espiral. O modelo espiral tem quatro fases, um projeto de software passa repetidamente por essas fases em iterações chamadas espirais.

- **Identificação**

Esta fase começa com a coleta dos requisitos de negócios na espiral da linha de base. Nos espirais posteriores, como o produto amadurece, a identificação de requisitos de sistema, requisitos de subsistemas e os requisitos de unidade são feitos.

Isto também inclui a compreensão dos requisitos do sistema de comunicação contínua entre o cliente e o analista de sistemas. No final da espiral do produto, ele é implantado no mercado identificado.

- **Design**

Esta fase de projeto começa com o projeto conceitual na espiral da linha de base e envolve: projeto arquitetônico, projeto lógico de módulos, design de produto físico e design final nas espirais subsequentes.

- **Construção**

Refere-se à produção do produto de software real em cada espiral. Na espiral da linha de base, quando o produto é apenas pensado e o projeto está sendo desenvolvido, um POC (*Proof of Concept*) é desenvolvido para obter feedback do cliente.

- **Avaliação e análise de riscos**

Visa estimar e monitorar viabilidade e gestão de riscos técnicos – como o não cumprimento do cronograma e a superação de custo. Depois de testar a acumulação, no final da primeira iteração, o cliente avalia o software e fornece feedback.

PRÓS	CONTRAS
• Novas exigências podem ser acomodadas; • Permite o uso extensivo de protótipos; • Os requisitos podem ser capturados com mais precisão; • Os usuários veem o sistema mais cedo; • O desenvolvimento pode ser dividido em partes menores e as peças de maior risco podem ser desenvolvidas antes, o que ajuda na melhor gestão de riscos.	• O gerenciamento é mais complexo; • O fim do projeto não pode ser conhecido mais cedo; • Não é adequado para projetos de pequeno ou de baixo risco, e pode ser caro para pequenos projetos; • O processo é complexo; • Grande número de etapas intermediárias, exige documentação excessiva.

O modelo V

No modelo V, a fase de testes é realizada em paralelo a fase de desenvolvimento. Portanto, há fases de verificação de um lado do "V" e fases de validação do outro lado. A fase de Codificação une os dois lados do modelo V. A seguir, segue uma descrição de cada uma das três fases:

- **Fase de verificação**

A Fase de Verificação é dividida em:

- **Análise de requisitos de negócio:**

Esta é a primeira fase do ciclo de desenvolvimento, onde os requisitos do produto são compreendidos a partir da perspectiva do cliente. Esta fase envolve a comunicação detalhada com o cliente para entender suas expectativas e exigências exatas. É uma atividade muito importante e precisa ser bem gerida, por conta da maioria dos clientes não ter certeza sobre o que exatamente precisa. O planejamento do projeto de teste de aceitação é feito nessa fase, os requisitos de negócios podem ser usados como entrada para o teste de aceitação;

- **Design de sistemas:**

Depois de ter os requisitos do produto claros e detalhados, é hora de projetar o sistema completo. O projeto do sistema seria constituído de compreensão e detalhamento da configuração completa do hardware e comunicação para o produto em desenvolvimento. O plano de teste do sistema é desenvolvido com base no projeto do sistema. Fazer isso em um estágio anterior deixa mais tempo depois para a execução de teste real;

- **Projeto de arquitetura:**

Especificações de arquitetura são compreendidas e projetadas nessa fase. Normalmente é proposta mais de uma abordagem técnica, baseada sobre a viabilidade técnica e financeira de ser tomada a decisão final. O projeto do sistema é dividido em módulos, mas ocupando funcionalidades diferentes. Isso também é conhecido como *High Level design* (HLD);

- **Design de módulo:**

Nesta fase, o design interno detalhado para todos os módulos do sistema é especificado, conhecido como *Low Level Design* (LLD). É importante que a concepção seja compatível com os outros módulos da arquitetura do sistema e dos outros sistemas externos. Os testes de unidade são uma parte essencial de qualquer processo de desenvolvimento e ajudam a eliminar as falhas e os erros máximos em uma fase bastante precoce. Os testes de unidade podem ser projetados nesse estágio, baseados nos projetos de módulos internos.

- **Fase de codificação**

A codificação real dos módulos do sistema projetada na fase de concepção é retomada na fase de codificação. A linguagem de programação mais adequada é definida com base no sistema e nos requisitos arquitetônicos. A codificação é realizada com base nas diretrizes de codificação e nos padrões. O código passa por inúmeras revisões e é otimizado para melhor desempenho antes da versão final, marcada para o repositório.

- **Fase de validação**

A Fase de Validação é dividida em:

- **Testes de unidades**

Os testes de unidades projetados na fase de projeto do módulo são executados no código durante essa fase de validação. O teste de unidade é realizado em nível de código e ajuda a eliminar erros em um estágio inicial, embora nem todos os defeitos possam ser descobertos;

- **Testes de integração**

Os testes de integração são associados com a fase de projeto arquitetônico. São realizados para testar a convivência e comunicação dos módulos internos dentro do sistema;

- **Teste do sistema**

Testes de sistema estão diretamente associados com a fase de projeto do sistema. São a verificação de todas as funcionalidades do sistema e da comunicação do sistema em desenvolvimento com os sistemas externos. A maioria dos problemas de software e de compatibilidade de hardware podem ser descobertos durante a execução desse tipo de teste;

- **Testes de aceitação**

Os testes de aceitação são associados com a fase de análise de requisitos de negócio e envolvem testar o produto no ambiente do usuário. Eles descobrem os problemas de compatibilidade com outros sistemas disponíveis no ambiente do usuário. Também encontram as questões não funcionais – como carga –, de desempenho, e defeitos no ambiente real do usuário.

PRÓS	CONTRAS
• Este é um modelo altamente disciplinado e fases são concluídas uma de cada vez; • Funciona bem para projetos menores onde as exigências são muito bem compreendidas; • Simples e fácil de entender e usar; • Fácil de gerenciar devido à rigidez do modelo – cada fase tem produtos específicos e um processo de revisão.	• Alto risco e incerteza; • Não é um bom modelo para projetos complexos e de orientação a objetos; • Pobre modelo para projetos de duração longa e contínua; • Não é adequado para projetos onde os requisitos sofrem risco moderado a alto de mudar; • Depois que um aplicativo está em fase de testes, é difícil voltar atrás e mudar uma funcionalidade; • Nenhum software de trabalho é produzido até o final durante o ciclo de vida.

O modelo Big Bang

O Modelo Big Bang foi construído para focar todos os recursos possíveis no desenvolvimento de software e de codificação, com muito pouco ou nenhum planejamento. Os requisitos são entendidos e implementados como eles vêm, quaisquer alterações necessárias podem ou não exigir renovar o software completo.

Esse modelo é ideal para pequenos projetos com um ou dois desenvolvedores trabalhando em conjunto, e também é útil para projetos acadêmicos ou de prática. É um modelo ideal para o produto em que as exigências não são bem compreendidas e a data de lançamento final não é dada.

PRÓS	CONTRAS
• Este é um modelo muito simples; • Pouco ou nenhum planejamento exigido; • Fácil de gerenciar; • Poucos recursos são necessários; • Dá flexibilidade para os desenvolvedores; • É uma boa ajuda de aprendizagem para os recém-chegados ou estudantes.	• Muito alto o risco e a incerteza; • Não é um bom modelo para projetos complexos e de orientação a objetos; • Pobre modelo para projetos de duração longas e contínuas; • Pode vir a ser muito caro se os requisitos forem incompreendidos.

Material Complementar

Explore

PRESSMAN, R. S.ENGENHARIA De Software. 6ª ed. Porto Alegre: Grupo A, 2010. (e-book)

STEPHEN R. S. Engenharia de Software. 8ª ed. Porto Alegre: Grupo A, 2008. (e-book)

PADUA, W. Engenharia de Software, 3ª ed. Rio de Janeiro: Grupo GEN, 2008. (e-book)

KALINOVSKY, A. Java Secreto: técnicas de descompilação, patching e engenharia reversa. São Paulo: Pearson, 2009. (e-book)

PFLEEGER, S. L. Engenharia de Software: teoria e prática - 2º ed. São Paulo: Pearson, 2009. (e-book)

Referências

KALINOVSKY, A. **Java secreto**: técnicas de descompilação, patching e engenharia reversa. São Paulo: Pearson, 2009.

PADUA, W. **Engenharia de software**. 3. ed. Rio de Janeiro: Grupo GEN, 2008.

PFLEEGER, S. L. **Engenharia de software**: teoria e prática. 2. ed. São Paulo: Pearson, 2009.

PRESSMAN, R. S. **Engenharia de software**. 6. ed. Porto Alegre, RS: Grupo A, 2010.

SCHACH, S. R. **Engenharia de software**: os paradigmas clássicos e orientado a objetos. 7. ed. São Paulo: Bookman, 2009.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Addison-Wesley, 2007.

STEPHEN R. S. **Engenharia de software**. 8. ed. Porto Alegre, RS: Grupo A, 2008.

WAZLAWICK, R. S. **Análise e projeto de sistemas de informação orientados a objetos**. 2. ed. Rio de Janeiro: Elsevier, 2011.

Anotações

Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000

Universidade Cruzeiro do Sul

UNICID Universidade Cidade de S. Paulo

UNIFRAN Universidade de Franca

UDF Centro Universitário

Módulo Centro Universitário