GOVERNO DO ESTADO DO CEARÁ

Secretaria da Ciência, Tecnologia e Educação Superior

UNIVERSIDADE ESTADUAL VALE DO ACARAÚ

Engenharia de Software

NEaD UVA

Núcleo de Educação a Distância

Universidade Estadual Vale do Acaraú

Engenharia de Software

O QUE É "PRÁTICA" EM ENGENHARIA DE SOFTWARE?

De modo geral, a *prática* é uma coleção de conceitos, princípios, métodos e ferramentas da qual um engenheiro de software faz uso diariamente. A prática permite aos gerentes gerenciar projetos de software e, aos engenheiros de software, construir programas de computador.

Por que é importante?

A prática fornece os detalhes de que você vai precisar para seguir caminho. Preenche um modelo de processo de software com as receitas técnicas e gerenciais necessárias para fazer o serviço. Ela transforma uma abordagem aleatória, não enfocada, com alguma coisa que é mais organizada, efetiva e provável de alcançar sucesso.

Quais são os passos? Três elementos da prática aplicam-se independente do modelo de processo escolhido. São eles: *conceitos, princípios e métodos.* Um quarto elemento da prática -- as ferramentas -- dão apoio à aplicação dos métodos.

A ESSÊNCIA DA PRÁTICA

1. *Entenda o problema* (comunicação e análise)
2. *Planeje uma solução* (modelagem e projeto de software)
3. *Execute o plano* (geração de código)
4. *Examine o resultado quanto à precisão* (teste e garantia de qualidade)

PRINCÍPIOS CENTRAIS

1. A Razão Por que Tudo Existe: Um sistema de software existe por uma razão. Todas as decisões devem ser tomadas com isso em mente.

2. KISS (*Keep It Simple, Stupid* - Mantenha a Coisa Simples!): Ter um sistema mais simples de entender e de manter. Na verdade projetos mais elegantes costumam ser os mais simples. Simples também não significa "rápido e sujo". De fato, frequentemente, simplificar precisa de muito raciocínio e trabalho em várias iterações.

3. Mantenha a Visão: Uma visão clara e bem definida do sistema é essencial para o sucesso de um projeto de software.

4. Mantenha a Visão
(O Que Você Produz Outros Vão Consumir): Sempre especifique, projete e implemente sabendo que mais alguém terá que entender o que você está fazendo. De um modo ou de outro alguém mais vai usar, manter, documentar ou precisará entender o seu sistema.

5. Esteja Aberto para o Futuro: Um sistema com um longo tempo de vida tem maior valor. Nos ambientes de computação atuais, em que as especificações mudam de um momento para outro e as plataformas de hardware ficam obsoletas depois de apenas alguns meses os tempos de vida do software são tipicamente medidos em meses em vez de anos. Para fazer isso com sucesso, eles precisam estar prontos para se adaptar a essas e outras modificações.

6. Planeje com Antecedência o Reuso: Reuso poupa tempo e esforço. Planejar o reuso com antecedência reduz o custo e aumenta o valor tanto dos componentes reusáveis quanto do sistema ao qual eles são incorporados.

PRÁTICAS DE COMUNICAÇÃO

Antes que os requisitos do cliente possam ser analisados, modelados ou especificados, eles precisam ser coletados por meio de uma atividade de *comunicação* (também chamada de *levantamento de requisitos*).
A comunicação efetiva (entre os técnicos, o cliente e outros interessados e com os gerentes) está entre as atividades mais desafiadoras com as quais se confronta um engenheiro de software. Nesse contexto apresentamos 10 princípios e conceitos para a comunicação com o cliente.

Princípio 01: Escute – Tente se concentrar nas palavras do interlocutor em vez de na formulação de sua resposta a essas perguntas. Peça esclarecimentos se algo não estiver claro, sempre tendo o cuidado de evitar interrupções constantes.

Princípio 02: Prepare-se antes de se comunicar – Dedique um tempo para entender o problema antes de se encontrar com os outros. Estude sobre o negócio e esteja familiarizado com os jargões mais utilizados na atividade.

Princípio 03: Alguém deve facilitar a atividade – Toda reunião de comunicação deve ter um líder e suas principais funções são: manter o rumo e objetivo da conversa, mediar conflitos e garantir que os outros princípios sejam seguidos.

Princípio 04: Comunicação *face-a-face* é melhor – E funciona ainda melhor quando ocorre apoiada com algum tipo de material complementar como uma apresentação, figuras ou um documento que sirva como foco para a discussão.

Princípio 05: Faça anotações e documente as decisões – Anote todos os pontos e decisões importantes que foram tomadas para que nada se perca, ou que mais tarde você não lembre. Uma boa prática é a elaboração de atas de reunião.

Princípio 06: Busque colaboração – Durante o desenvolvimento de um sistema a colaboração e o consenso ocorrem quando o conhecimento coletivo dos membros da equipe é utilizado para se produzir o produto de software desejado.

Princípio 07: Mantenha o foco, conduza a discussão em partes – Quanto mais pessoas estiverem envolvidas em uma reunião, mais provavelmente a discussão tende a ser direcionada para assuntos paralelos. O facilitador deve manter a discussão em partes, abandonando um tópico apenas depois que o mesmo tiver sido resolvido. (Mas atenção ao Princípio 9)

Princípio 08: Se algo não está claro, desenhe uma figura – A comunicação verbal tem um limite. Um desenho (foto, gráfico ou vídeo) pode normalmente fornecer o esclarecimento necessário quando as palavras não conseguem fazer o serviço.

Princípio 09: Quando você concorda com algo, prossiga; se você não pode concordar com algo, prossiga; se uma característica não está clara e não pode ser esclarecida no momento, prossiga – A comunicação como qualquer atividade da engenharia de software, leva tempo. Em vez de entrar em um caminho sem fim, as pessoas devem reconhecer que muitos tópicos requerem discussão (veja o princípio 2) e que prosseguir é às vezes, o melhor modo de conseguir agilidade na comunicação.

Princípio 10: Negociação não é um concurso ou um jogo - Funciona melhor quando *ambas as partes ganham.* Em algumas

situações o engenheiro de software e o cliente precisarão negociar. Negociar funções e características, prioridades, datas de entrega e valores. As negociações devem partir do princípio conhecido como "ganha-ganha", onde ninguém deve sair perdendo.

PRÁTICAS DE PLANEJAMENTO

A atividade de comunicação ajuda uma equipe de software a definir seus objetivos e metas gerais. No entanto, entender essas metas e objetivos não é mesmo que definir um plano para atingi-los (ou na maioria das vezes chamado de "*plano de ação*").

Por que é importante?

O planejamento inclui um conjunto de práticas gerenciais e técnicas que permite à equipe de software definir um roteiro enquanto ela se move em direção a sua estratégia e seus objetivos.

Princípio 01: Entenda o escopo do projeto – é impossível usar um roteiro se você não souber onde está indo. O escopo oferece à equipe um destino. Você precisa saber qual seu objetivo final.

Princípio 02: Envolva o cliente na atividade de planejamento – o cliente define prioridades e oferece restrições ao projeto. Para acomodar tais realidades, os planejadores precisam negociar a ordem das entregas, prazos e outros tópicos relacionados ao projeto. Esse princípio é importante, pois o cliente pode acabar percebendo que nem tudo o que definiu é realmente necessário.

Princípio 03: Reconheça que o planejamento é iterativo – um plano de projeto nunca é gravado na pedra, isto é, quando se inicia o trabalho é provável que as coisas se modifiquem. Assim, o plano deve ser adaptado a estas mudanças. Assim como em todas as áreas, um planejamento "estratégico" pode (e algumas vezes, deve!) ser alterado quando se verifica que o que antes foi definido não está mais em acordo com os objetivos do projeto.

Princípio 04: Estime com base no que é sabido – a intenção de estimativa é indicar o esforço, o custo e a duração das tarefas com base em um entendimento atual da equipe quanto ao trabalho a ser executado. Se a informação não for suficiente ou não for confiável, as estimativas serão também não confiáveis.

Princípio 05: Considere os riscos à medida que você define o plano – se a equipe definir os riscos que tem grande impacto e alta probabilidade de ocorrerem é necessário um plano de contingência (Planos de ação caso a equipe se encontre em algum desses cenários de risco - Crie alguns planos "B")

Princípio 06: Seja realista – em um dia de trabalho, sempre ocorrem ruídos, ou seja, omissões e ambiguidades. Logo, modificações ocorrerão, pois mesmo os melhores profissionais cometem erros. Essas e outras realidades devem ser consideradas quando um plano for estabelecido.

Princípio 07: Ajuste a granularidade à medida que você define o plano – a granularidade trata-se do nível de detalhes que são introduzidos à medida que um plano de projeto é desenvolvido. O planejamento de atividades a serem executadas a curto prazo é apresentado em granularidade fina, isto é, com riqueza de detalhes. Já tarefas a serem executadas a longo prazo são planejadas com granularidade grossa, ou seja, oferece tarefas de trabalho mais amplas. Logo, a granularidade move-se de fina para grossa à medida que a linha do tempo se afasta da data atual. Atividades que não vão ocorrer nos próximos meses não exigem granularidade fina, pois podem ocorrer variações de cenário.

Princípio 08: Defina como você pretende garantir a qualidade – o plano deve identificar como a equipe pretende garantir a qualidade, se serão utilizadas revisões técnicas formais, ou outras formas de acompanhamento devem ser explicitadas no plano. Algumas vezes chamados de "*indicadores de desempenho*".

Princípio 09: Descreva como você pretende acomodar as mudanças – diante de modificações descontroladas, mesmo os melhores planejamentos pode ser comprometidos. (O negócio do cliente também pode sofrer alterações)

Princípio 10: Acompanhe o plano com frequência e faça ajustes quando necessário – é importante acompanhar o projeto diariamente, identificando áreas problemáticas e situações em que o que foi planejado se destorce da realidade. Quando o desvio for encontrado, o plano deve ser ajustado.

PRÁTICAS DE MODELAGEM

Criamos modelos para obter um melhor entendimento da entidade real a ser construída. Quando a entidade é uma coisa física (por exemplo, um edifício, avião ou máquina) podemos construir um modelo que é idêntico em forma e aspecto, mas em menor escala. No entanto, quando a entidade é um software, nosso modelo precisa assumir uma forma diferente.
Ele precisa ser capaz de representar a informação que o software transforma, a arquitetura e funções que permitem que essa transformação ocorra, as características que os usuários desejam e o comportamento do sistema à medida que a transformação ocorre.

Por que é importante?

Modelos de software são construídos para uma visualização do sistema a ser construído, permitindo uma melhor compreensão e entendimento. Eles podem também ser utilizados para a especificação e para a documentação do software. O modelo quando utilizado para especificação possibilita uma descrição precisa do será desenvolvido pelos programadores.

No trabalho de engenharia de software, duas classes de modelo são criadas: modelo de análise e modelos de projetos. Os *modelos de análise* representam os requisitos do cliente mostrando o software em três domínios diferentes: o domínio de informação, o domínio funcional e o domínio comportamental.
Os *modelos de projeto* representam características de software que ajudam os profissionais a construí-lo efetivamente: a arquitetura, a interface do usuário e detalhes em nível de componentes.

PRINCÍPIOS DA MODELAGEM DE ANÁLISE

Princípio 01: O domínio de informação de um problema precisa ser representado e entendido – o domínio de informação abrange os dados que fluem para dentro do sistema (vindos de usuários finais, de outros sistemas ou de dispositivos externos), os dados que fluem para fora do sistema (pela interface do usuário, por interfaces de rede, relatórios, gráficos e outros meios) e os depósitos de dados que coletam e organizam os objetos de dados persistentes (banco de dados).

Princípio 02: As funções a serem desenvolvidas pelo software devem ser definidas – As funções do software podem ser descritas em vários níveis de abstração diferentes que vão desde declaração geral de objetivo até uma descrição detalhada dos elementos de processamento que precisam ser invocados.

Princípio 03: O comportamento do software precisa ser representado – O comportamento do software é guiado por suas interações com o ambiente externo. Entradas fornecidas pelos usuários finais, dados de controle fornecidos por um sistema externo ou monitoramento de dados coletados em uma rede, todos fazem que o software se comporte de um modo específico.

Princípio 04: Os modelos que mostram informação, função e comportamento devem ser particionados de um modo que revele detalhes em forma de camadas (hierarquia) – Problemas

complexos são difíceis de serem resolvidos como um todo. Por isso, usamos uma estratégia de dividir para conquistar. Um problema complexo, grande, é dividido em subproblemas até que cada subproblema seja relativamente fácil de entender. Esse conceito é chamado de ***particionamento*** e é uma estratégia-chave na modelagem de análise**.**

Princípio 05: A tarefa de análise deve ir da informação essencial até os detalhes de implementação – A modelagem de analise começa descrevendo um problema na perspectiva do usuário final. A "essência" do problema sem qualquer consideração de como uma solução será implementada. Os detalhes de implementação (normalmente descritos como parte do modelo de projeto) indicam como a essência será implementada.

PRINCÍPIOS DA MODELAGEM DE PROJETO

Princípio 01: O projeto deve estar relacionado ao modelo de análise – O modelo de análise descreve o domínio de informação do problema, funções visíveis ao usuário, o comportamento do sistema e um conjunto de classes. O modelo de projeto traduz essa informação em uma arquitetura em nível de componente que são a realização das classes de análise.

Princípio 02: Sempre considere a arquitetura do sistema a ser construído – A arquitetura do software é o esqueleto do sistema a ser construído. Ela afeta as interfaces, estruturas de dados, fluxo de controle e comportamento do programa, o modo pelo qual o teste pode ser conduzido, a manutenibilidade do sistema resultante e muito mais. Por todas essas razões, o projeto deve começar com considerações arquiteturais. Apenas depois de a arquitetura ser estabelecida, os tópicos em nível de componente devem ser considerados.

Princípio 03: O projeto de dados é tão importante quanto o projeto de funções de processamento – O projeto dos dados é um elemento essencial do projeto arquitetural. A maneira pela qual os objetos de dados são realizados no projeto não pode ser deixada ao acaso. Um projeto de dados bem estruturado ajuda a simplifica o fluxo do programa, tornar o projeto e implementação dos componentes de software mais fáceis e deixa o processamento global mais eficiente.

Princípio 04: As interfaces precisam ser projetadas com cuidado – A maneira como os dados fluem entre os componentes de um sistema tem muito a ver com a eficiência de processamento, a propagação de erros e a simplicidade de projeto. Uma interface bem projetada torna a integração mais fácil e ajuda o testador na validade das funções dos componentes**.**

Princípio 05: O projeto de interface do usuário deve estar sintonizado com as necessidades do usuário final. No entanto, em cada caso, ele deve enfatizar a facilidade de uso – A interface do

usuário é a manifestação visível do software. Não importa quão sofisticadas sejam suas funções internas, quão abrangente suas estruturas de dados, quão bem projetada sua arquitetura, um projeto de interface ruim conduz, muitas vezes, à percepção de que o software é "ruim".

Princípio 06: O projeto em nível de componente deve ser funcionalmente independente – A independência funcional é uma medida da "objetividade" de um componente de software. A funcionalidade fornecida por um componente deve ser *coesiva* – isto é, deve enfocar uma e apenas uma função dou subfunção.

Princípio 07: Os componentes devem ser fracamente acoplados uns aos outros e ao ambiente externo –À medida que o nível de acoplamento aumenta, a probabilidade de propagação de erros também aumente e a manutenibilidade global do software diminui. Assim, o acoplamento deve ser mantido tão baixo quanto for razoável.

Princípio 08: Representações de projeto devem ser facilmente compreensíveis – O objetivo do projeto é comunicar a informação para os profissionais que vão gerar o código, para aqueles que vão testar o software, e para outros que podem vir a manter o software no futuro. Se o projeto for difícil de entender, ele não servirá como um meio de comunicação efetivo.

Princípio 09: O projeto deve ser desenvolvido iterativamente. A cada iteração, o projetista deve lutar por maior simplicidade – Como quase todas as atividades criativas, o projeto ocorre iterativamente. As primeiras iterações trabalham para refinar o projeto e corrigir erros, mas as últimas iterações devem procurar tornar o projeto o mais simples possível.

PRÁTICA DE CONSTRUÇÃO

A atividade de *construção* compreende um conjunto de tarefas de codificação e teste que levam ao software operacional que está pronto para ser entregue ao cliente ou ao usuário final. No trabalho moderno de engenharia de software, a codificação pode ser: (1) a criação direta de código-fonte em linguagem de programação; (2) a geração automática de código-fonte usando uma representação intermediária análoga ao projeto do componente a ser construído; (3) a geração automática de código executável usando linguagem de programação de quarta geração (por exemplo, Visual C++).

PRINCÍPIOS E CONCEITOS DE CODIFICAÇÃO

Os princípios e conceitos que dirigem a tarefa de codificação são estilo de programação, linguagem de programação e métodos de programação rigorosamente definidos. No entanto, há um número

de princípios fundamentais que podem ser enunciados:

Princípios de preparação: Antes de escrever uma linha de código, certifique-se de:

1.Entender o problema que está tentando resolver.
2.Entender os princípios e conceitos do projeto.
3.Escolher uma linguagem de programação que satisfaça às necessidades do software.
4.Selecionar um ambiente de programação que fornece ferramentas para facilitar o seu trabalho.
5. Criar um conjunto de testes unitários que será aplicado tão logo o componente que você está codificando seja completado.

Princípios de codificação: Quando começar a escrever o código, certifique-se de:

1. Restringir os seus algoritmos seguindo a prática de programação estruturada.
2. Selecionar estruturas de dados que atendam às necessidades do projeto.
3. Entender a arquitetura do software e criar interfaces que sejam consistentes com ela.
4.Conservar a lógica condicional tão simples quanto possível.
5. Criar ciclos aninhados de modo que sejam facilmente testáveis.
6. Selecionar nomes significativos de variáveis e seguir outras normas locais de codificação.
7. Escrever código que é autodocumentado.
8. Criar uma disposição visual que auxilie o entendimento.

Princípios de validação: Depois que completar seu primeiro passo de codificação, certifique-se de:

1. Conduzir uma inspeção de código quanto adequado.
2. Realizar testes unitários e corrigir os erros descobertos.
3. Refabricar o código.

PRINCÍPIOS DE TESTE

Princípio 01: Todos os testes devem estar relacionados aos requisitos– O objetivo do teste de software é descobrir erros. Segue-se daí que os defeitos mais severos (do pondo de vista do cliente) são aqueles que fazem que o programa deixe de satisfazer aos seus requisitos.

Princípio 02: Os testes devem ser planejados com antecedência– Todos os testes podem ser planejados e projetados antes que qualquer código tenha sido gerado, ou seja, o planejamento de testes (capítulo 13) pode começar assim que o modelo de análise for completado.

Princípio 03: O princípio de Pareto se aplica ao teste de software– Colocado simplesmente, o princípio de Pareto implica que 80% de todos os erros descobertos durante o teste estarão, provavelmente, relacionados a 20% de todos os componentes do programa. O problema, sem dúvida, é isolar esses componentes suspeitos e testá-los rigorosamente.

Princípio 04: O teste deve começar "no varejo" e progredir até o teste "no atacado"– Os primeiros testes planejados e executados concentram-se nos componentes individuais. À medida que o teste progride, o foco se desloca numa tentativa de encontrar erros em conjuntos integrados de componentes e, finalmente, em todo o sistema.

Princípio 05: Testes exaustivos não são possíveis –A quantidade de permutações de caminho, mesmo para um programa de tamanho moderado, é excepcionalmente grande. Por essa razão, é impossível executar todas as combinações de caminhos durante o teste. È possível, no entanto, cobrir adequadamente a lógica e garantir que todas as condições do projeto em nível de componente, tenham sido exercitadas.

IMPLANTAÇÃO

Princípio 01: As expectativas do cliente quanto ao software devem ser geridas– O engenheiro de software deve ser cuidadoso com o envio de mensagens conflitantes ao cliente (por exemplo, prometer mais do que você pode realmente entregar).

Princípio 02: Um pacote completo de entrega deve ser montado e testado– Um CD-ROM ou outra mídia contendo todo o software executável, arquivos de dados de suporte, documentos de suporte e outras informações relevantes deve ser montado e rigorosamente testado. Todos os documentos de instalação e outras características operacionais devem ser seriamente exercitados em todas as possibilidades de configuração de computação (isto é, hardware, sistemas operacionais, arranjos de redes, periféricos)

Princípio 03: Um regime de suporte deve ser estabelecido antes de o software ser entregue–Um usuário final espera receptividade e informação segura quando uma questão ou um problema surge. Se o suporte é *ad hoc (finalidade especifica)*ou, pior, inexistente, o cliente ficará insatisfeito imediatamente. O suporte deve ser planejado, o material de suporte preparado e mecanismos adequados de registros devem ser estabelecidos, de modo que a equipe de software possa conduzir uma avaliação categórica das espécies de suporte necessárias.

Princípio 04: Materiais institucionais adequados devem ser fornecidos aos usuários finais–A

equipe de software entrega mais do que o software em si. Ajuda de treinamento adequado deve ser desenvolvida e materiais de auxílio.

Princípio 05: Software defeituoso deve ser corrigido primeiro e, depois entregue–Pressionados pelo tempo, algumas organizações de software entregam incrementos de baixa qualidade com um aviso ao cliente de que os defeitos "serão corrigidos na versão seguinte". Isso é um erro. Há um ditado no negócio de software que diz:"os clientes esquecerão que você entregou um produto de alta qualidade alguns dias depois, mas eles nunca esquecerão os problemas que um produto de baixa qualidade lhes causou. O software os lembra a cada dia"

RESUMO

A prática de engenharia de software engloba conceitos, princípios, métodos e ferramentas que engenheiros de software aplicam durante o processo de software. Cada projeto de engenharia de software é diferente, no entanto, um conjunto de princípios e tarefas genéricas aplica-se a cada atividade de arcabouço de processo independentemente do projeto ou do produto.

BIBLIOGRAFIA

LEITE, Jair C. **Engenharia de Software** [*homepage* na Internet]**.** Natal-RN: UFRN; 3 de Junho de 2007. Acesso em 20 de Janeiro de 2013. Disponível em: http://engenhariadesoftware.blogspot.com.br/2007/06/modelos-de-software.html