Curso: Sistemas de Informação	
Disciplina: Padrões de projeto	**Período:** 8
Professor: Luiz Gustavo Dias	**Tipo:** Estudo Complementar
Objetivo: Princípios da Engenharia que norteiam a prática	

1. PRINCÍPIOS FUNDAMENTAIS

1.1 Princípios que Orientam o Processo:

Princípio 1. Seja ágil. Não importa se o modelo de processo que você escolheu é prescritivo ou ágil, os princípios básicos do desenvolvimento ágil devem comandar sua abordagem. Todo aspecto do trabalho deve enfatizar a economia de ações – mantenha a abordagem técnica tão simples quanto possível, mantenha os produtos tão concisos quanto possível e tome decisões localmente sempre que possível.

Princípio 2. Concentre-se na qualidade em todas as etapas. A condição final de toda atividade, ação e tarefa do processo deve se concentrar na qualidade do produto.

Princípio 3. Esteja pronto para adaptações. Processo não é uma experiência religiosa e não há espaço para dogmas. Quando necessário, adapte sua abordagem às restrições impostas pelo problema, pelas pessoas e pelo próprio projeto.

Princípio 4. Monte uma equipe eficiente. O processo e a prática de engenharia de software são importantes, mas o fator mais importante são as pessoas. Forme uma equipe que se organize automaticamente, que tenha confiança e respeito mútuos.

Princípio 5. Estabeleça mecanismos para comunicação e coordenação. Os projetos falham devido à omissão de informações importantes e/ ou devido a falha dos envolvidos, na coordenação de seus esforços para criar um produto final bem sucedido. Esses são itens de gerenciamento e devem ser tratados.

Princípio 6. Gerencie mudanças. A abordagem pode ser tanto formal quanto informal, entretanto devem ser estabelecidos mecanismos para gerenciar a maneira como as mudanças serão solicitadas, avaliadas, aprovadas e implementadas.

Princípio 7. Avalie os riscos. Uma série de coisas pode dar errada quando um software é desenvolvido. É essencial estabelecer planos de contingência. Alguns

desses planos de contingência formarão a base das tarefas de engenharia de segurança.

Princípio 8. Gere artefatos que forneçam valor para outros. Crie apenas artefatos que proporcionarão valor para outro processo, atividades, ações ou tarefas. Todo artefato produzido como parte da prática da engenharia de software será repassado para alguém. Uma lista de funções e características requisitadas será repassada para a pessoa (ou pessoas) que vai desenvolver um projeto, o projeto será repassado para aqueles que gerarão o código e assim por diante. Certifique-se de que o artefato contenha a informação necessária, sem ambiguidades ou omissões.

1.2 Princípios que Orientam a Prática

Princípio 1. Divida e conquiste. De forma mais técnica, a análise e o projeto sempre devem enfatizar a separação por interesses (SoCs, separation of concerns). Um problema será mais fácil de resolver se for subdividido em conjuntos de interesses. Na forma ideal, cada interesse fornece uma funcionalidade distinta a ser desenvolvida e, em alguns casos, validada, independentemente de outros negócios.

Princípio 2. Compreenda o uso da abstração. Em essência, abstrair é simplificar algum elemento complexo de um sistema comunicando o significado em uma única frase. Quando se usa a abstração planilha, presume-se que se compreenda o que vem a ser uma planilha, a estrutura geral de conteúdo que uma planilha apresenta e as funções típicas que podem ser aplicadas a ela. Na prática de engenharia de software, usam-se muitos níveis diferentes de abstração, cada um incorporando ou implicando um significado que deve ser comunicado. No trabalho de análise de projeto, uma equipe de software normalmente inicia com modelos que representam altos níveis de abstração (por exemplo, planilha) e, aos poucos, refina tais modelos em níveis de abstração mais baixos (por exemplo, uma coluna ou uma função de soma).

O objetivo de uma abstração é eliminar a necessidade de comunicar detalhes. Algumas vezes, efeitos problemáticos surgem devido ao “vazamento” desses detalhes. Sem o entendimento dos detalhes, a causa de um problema não poderá ser facilmente diagnosticada.

Princípio 3. Esforce-se pela consistência. Seja criando um modelo de análise, desenvolvendo um projeto de software, gerando código-fonte ou criando casos de teste, o princípio da consistência sugere que um contexto conhecido facilita o uso do software. Consideremos, por exemplo, o projeto de uma interface para o usuário de uma WebApp. A colocação padronizada do menu de opções, o uso padronizado de um esquema de cores e de ícones identificáveis colaboram para uma interface ergonomicamente boa.

Princípio 4. Concentre-se na transferência de informações. Software trata da transferência de informações: do banco de dados para um usuário, de um sistema judiciário para uma aplicação na Web (WebApp), do usuário para uma interface gráfica (GUI), de um sistema operacional de um componente de software para outro; a lista é quase infinita. Em todos os casos, a informação flui por meio de uma interface, e, como consequência, há a possibilidade de erros, omissões e ambiguidade. A implicação desse princípio é que se deve prestar especial atenção à análise, ao projeto, construção e testes das interfaces.

Princípio 5. Construa software que apresente modularidade efetiva. A separação por interesse (Princípio 1) estabelece uma filosofia para software. A modularidade fornece um mecanismo para colocar a filosofia em prática. Qualquer sistema complexo pode ser dividido em módulos (componentes), porém a boa prática de engenharia de software demanda mais do que isso. A modularidade deve ser efetiva. Isto é, cada módulo deve se concentrar exclusivamente em um aspecto bem restrito do sistema – deve ser coeso em sua função e/ou apresentar conteúdo bem preciso. Além disso, os módulos devem ser interconectados de uma maneira relativamente simples – cada módulo deve apresentar baixo acoplamento com outros módulos, fontes de dados e outros aspectos ambientais.

Princípio 6. Verifique os padrões. Brad Appleton propõe que:

O objetivo dos padrões é criar uma fonte literária para ajudar os desenvolvedores de software a solucionar problemas recorrentes, encontrados ao longo de todo o desenvolvimento de software. Os padrões ajudam a criar uma linguagem que pode ser compartilhada de tal forma que possa transmitir conteúdos e experiências acerca dos problemas e de suas soluções. Codificar formalmente tais soluções e as suas relações permite que se armazene com sucesso a base do conhecimento, a qual

define nossa compreensão sobre boas arquiteturas, correspondendo às necessidades de seus usuários.

Padrões de projeto podem ser utilizados em problemas mais amplos de engenharia e de integração de sistemas, permitindo que os componentes de sistemas complexos evoluam independentemente.

Princípio 7. Quando possível, represente o problema e sua solução sob perspectivas diferentes. Ao analisar um problema e sua solução sob uma série de perspectivas diferentes, é mais provável que se obtenha uma melhor visão e, assim, os erros e omissões sejam revelados. Por exemplo, um modelo de requisitos pode ser representado usando um ponto de vista orientado a cenários, um ponto de vista orientado a classes ou um ponto de vista comportamental. Cada um deles fornece uma perspectiva diferente do problema e de seus requisitos.

Princípio 8. Lembre-se de que alguém fará a manutenção do software. No longo prazo, à medida que defeitos forem descobertos, o software será corrigido, adaptado de acordo com as alterações de seu ambiente e estendido conforme solicitação de novas funcionalidades por parte dos envolvidos. As atividades de manutenção podem ser facilitadas se for aplicada uma prática de engenharia de software consistente ao longo do processo.

2. PRINCÍPIOS DAS ATIVIDADES METODOLÓGICAS

2.1 Princípios da Comunicação

Princípio 1. Ouça. Concentre-se mais em ouvir do que em se preocupar com respostas. Peça esclarecimento, se necessário, e evite interrupções constantes. Nunca se mostre contestador, tanto em palavras quanto em ações (por exemplo, revirar olhos ou balançar a cabeça) enquanto uma pessoa estiver falando.

Princípio 2. Prepare-se antes de se comunicar. Dedique tempo para compreender o problema antes de se reunir com outras pessoas. Se necessário, faça algumas pesquisas para entender o jargão da área de negócios em questão. Caso seja sua a responsabilidade conduzir uma reunião, prepare uma agenda com antecedência.

Princípio 3. Alguém deve facilitar a atividade. Toda reunião de comunicação deve ter um líder (um facilitador) para manter a conversa direcionada e produtiva, mediar qualquer conflito que ocorra e garantir que outros princípios sejam seguidos.

Princípio 4. Comunicar-se pessoalmente é melhor. No entanto, costuma ser mais produtivo quando alguma outra representação da informação relevante está presente. Por exemplo, um participante pode fazer um desenho ou um esboço de documento que servirá como foco para a discussão.

Princípio 5. Anote e documente as decisões. As coisas tendem a cair no esquecimento. Algum participante desta reunião deve servir como “gravador” e anotar todos os pontos e decisões importantes.

Princípio 6. Esforce-se para conseguir colaboração. Colaboração e consenso ocorrem quando o conhecimento coletivo dos membros da equipe é usado para descrever funções e características do produto ou sistema. Cada pequena colaboração servirá para estabelecer confiança entre os membros e chegar a um objetivo comum.

Princípio 7. Mantenha o foco; crie módulos para a discussão. Quanto mais pessoas envolvidas, maior a probabilidade de a discussão saltar de um tópico a outro. O facilitador deve manter a conversa modular, abandonando um assunto somente depois de ele ter sido resolvido.

Princípio 8. Faltando clareza, desenhe. A comunicação verbal flui até certo ponto. Um esboço ou um desenho pode permitir maior clareza quando palavras são insuficientes.

Princípio 9. (a) Uma vez de acordo, siga em frente. (b) Se não chegar a um acordo, siga em frente. (c) Se uma característica ou função não estiver clara e não puder ser elucidada no momento, siga em frente. A comunicação, assim como qualquer outra atividade da engenharia de software, toma tempo. Em vez de ficar interagindo indefinidamente, os participantes precisam reconhecer que muitos assuntos exigem discussão (veja o Princípio 2) e que “seguir em frente” é, algumas vezes, a melhor maneira de ser ágil na comunicação.

Princípio 10. Negociação não é uma competição nem um jogo. Funciona melhor quando as duas partes saem ganhando. Há muitas ocasiões em que é necessário negociar funções e características, prioridades e prazos de entrega. Se a equipe interagiu adequadamente, todas as partes envolvidas têm um objetivo comum. Mesmo assim, a negociação exigirá compromisso de todos.

2.2 Princípios do Planejamento

Princípio 1. Entenda o escopo do projeto. É impossível usar um mapa se você não sabe aonde está indo. O escopo indica um destino para a equipe de software.

Princípio 2. Inclua os envolvidos na atividade de planejamento. Os envolvidos definem prioridades e estabelecem as restrições de projeto. Para adequar essas características, os engenheiros muitas vezes devem negociar a programação de entrega, cronograma e outras questões relativas ao projeto.

Princípio 3. Reconheça que o planejamento é iterativo. Um plano de projeto jamais é gravado em pedra. Depois que o trabalho se inicia, muito provavelmente ocorrerão alterações. Consequentemente, o plano deverá ser ajustado para incluir as alterações. Além do mais, os modelos de processos incremental e iterativo exigem replanejamento após a entrega de cada incremento de software, de acordo com os feedbacks recebidos dos usuários.

Princípio 4. Faça estimativas baseadas no que conhece. O objetivo da estimativa é dar indicações de esforço, custo e prazo para a realização, com base na compreensão

atual do trabalho a ser realizado. Se a informação for vaga ou não confiável, as estimativas serão igualmente não confiáveis.

Princípio 5. Considere os riscos ao definir o plano. Caso tenha identificado riscos de alto impacto e alta probabilidade, um planejamento de contingência será necessário. Além disso, o plano de projeto (inclusive o cronograma) deve ser ajustado para incluir a possibilidade de um ou mais desses riscos ocorrerem. Leve em conta a provável exposição decorrente de perdas ou comprometimentos de bens do projeto.

Princípio 6. Seja realista. As pessoas não trabalham 100% de todos os dias. Sempre há interferência de ruído em qualquer comunicação humana. Omissões e ambiguidades são fatos da vida. Mudanças ocorrem. Até mesmo os melhores engenheiros de software cometem erros. Essas e outras realidades devem ser consideradas ao se estabelecer um plano de projeto.

Princípio 7. Ajuste particularidades ao definir o plano. Particularidades referem-se ao nível de detalhamento introduzido conforme o plano de projeto é desenvolvido. Um plano com alto grau de particularidade fornece considerável detalhamento de tarefas planejadas para incrementos em intervalos relativamente curtos para que o rastreamento e controle ocorram com frequência. Um plano com baixo grau de particularidade resulta em tarefas mais amplas para intervalos maiores. Em geral, particularidades variam de altas para baixas, conforme o cronograma de projeto se distancia da data atual. Nas semanas ou meses seguintes, o projeto pode ser planejado com detalhes significativos. As atividades que não serão realizadas por muitos meses não exigem alto grau de particularidade (muito pode ser alterado).

Princípio 8. Defina como pretende garantir a qualidade. O plano deve determinar como a equipe pretende garantir a qualidade. Se forem necessárias revisões técnicas, deve-se agendá-las. Se a programação em pares for utilizada, isso deve estar definido explicitamente dentro do plano.

Princípio 9. Descreva como acomodar as alterações. Mesmo o melhor planejamento pode ser prejudicado por alterações sem controle. Deve-se identificar como as alterações serão integradas ao longo do trabalho de engenharia. Por exemplo, o cliente pode solicitar uma alteração a qualquer momento? Se for solicitada uma mudança, a equipe é obrigada a implementá-la imediatamente? Como é avaliado o impacto e o custo de uma alteração?

Princípio 10. Verifique o plano com frequência e faça os ajustes necessários. Os projetos de software atrasam uma vez ou outra. Portanto, é bom verificar diariamente seu progresso, procurando áreas ou situações problemáticas, nas quais o que foi programado não está em conformidade com o trabalho realizado. Ao surgir um descompasso, deve-se ajustar o plano adequadamente.

2.3 Princípios da Modelagem

Princípio 1. O objetivo principal da equipe de software é construir software, não criar modelos. Agilidade significa entregar software ao cliente no menor prazo possível. Os modelos que fazem isso acontecer são criações valiosas; entretanto, os que retardam o processo ou oferecem pouca novidade devem ser evitados.

Princípio 2. Seja objetivo – não crie mais modelos do que precisa. Todo modelo criado deve ser atualizado quando ocorrem alterações. E, mais importante, todo modelo novo demanda tempo que poderia ser despendido em construção (codificação e testes). Portanto, crie somente modelos que facilitem mais e diminuam o tempo para a construção do software.

Princípio 3. Esforce-se ao máximo para produzir o modelo mais simples possível. Não exagere no software. Mantendo-se modelos simples, o software resultante também será simples. O resultado será um software mais fácil de ser integrado, testado e mantido. Além disso, modelos simples são mais fáceis de compreender e criticar, resultando em uma forma contínua de feedback (realimentação) que otimiza o resultado final.

Princípio 4. Construa modelos que facilitem alterações. Considere que os modelos mudarão, mas, ao considerar tal fato, não seja relapso. Por exemplo, uma vez que os requisitos serão alterados, há uma tendência de dar pouca atenção a seus modelos. Por quê? Porque se sabe que mudarão de qualquer forma. O problema dessa atitude é que, sem um modelo de requisitos razoavelmente completo, criar-se-á um projeto (modelo de projeto) que invariavelmente vai deixar de lado funções e características importantes.

Princípio 5. Estabeleça um propósito claro para cada modelo. Toda vez que criar um modelo, pergunte se há motivo para tanto. Se você não for capaz de dar justificativas sólidas para a existência do modelo, não desperdice tempo com ele.

Princípio 6. Adapte os modelos que desenvolveu ao sistema à disposição. Talvez seja necessário adaptar a notação ou as regras do modelo ao aplicativo; por exemplo, um aplicativo de videogame pode exigir uma técnica de modelagem diferente daquela utilizada em um software embarcado e de tempo real que controla o motor de um automóvel.

Princípio 7. Crie modelos úteis, mas esqueça a construção de modelos perfeitos. Ao construir modelos de requisitos e de projetos, um engenheiro de software atinge um ponto de retornos decrescentes. Isto é, o esforço necessário para fazer o modelo absolutamente completo e internamente consistente não vale os benefícios resultantes. Estaríamos sugerindo que a modelagem deve ser descuidada ou de baixa qualidade? A resposta é: não. Mas a modelagem deve ser conduzida tendo-se em vista as próximas etapas de engenharia de software. Iterar indefinidamente para tornar um modelo “perfeito” não supre a necessidade de agilidade.

Princípio 8. Não se torne dogmático quanto à sintaxe do modelo. Se ela consegue transmitir o conteúdo, a representação é secundária. Embora todos os integrantes de uma equipe devam tentar usar uma notação consistente durante a modelagem, a característica mais importante do mo- delo reside em transmitir informações que possibilitem a próxima tarefa de engenharia. Se um modelo viabilizar isso com êxito, a sintaxe incorreta pode ser perdoada.

Princípio 9. Se os instintos dizem que um modelo não está correto, mesmo parecendo correto no papel, provavelmente há motivos para se preocupar. Se você for um engenheiro experiente, confie em seus instintos. O trabalho com software nos ensina muitas lições – muitas das quais em um nível subconsciente. Se algo lhe diz que um modelo de projeto parece falho, embora não haja provas explícitas, há motivos para dedicar tempo extra, examinando o modelo ou desenvolvendo outro diferente.

Princípio 10. Obtenha feedback o quanto antes. Todo modelo deve ser revisado pelos membros da equipe de software. O objetivo das revisões é proporcionar feedback que seja usado para corrigir erros de modelagem, alterar interpretações errôneas e adicionar características ou funções omitidas inadvertidamente.

2.3.1Princípios da Modelagem de Requisitos

Princípio 1. O universo de informações de um problema deve ser representado e compreendido. O universo de informações engloba os dados constantes no sistema (do usuário, de outros sistemas ou dispositivos externos), os dados que fluem para fora do sistema (via interface do usuário, interfaces de rede, relatórios, gráficos e outros meios) e a armazenagem de dados que coleta e organiza objetos de dados persistentes (isto é, dados que são mantidos permanentemente).

Princípio 2. As funções executadas pelo software devem ser definidas. As funções do software oferecem benefício direto aos usuários e também suporte interno para fatores visíveis aos usuários. Algumas funções transformam dados que fluem no sistema. Em outros casos, as funções exercem certo nível de controle sobre o processamento interno do software ou sobre elementos de sistema externo. As funções podem ser descritas em diferentes níveis de abstração, desde afirmação geral até uma descrição detalhada dos elementos de processo que devem ser requisitados.

Princípio 3. O comportamento do software (consequência de eventos externos) deve ser representado. O comportamento de um software é comandado por sua interação com o ambiente externo. Dados fornecidos pelos usuários, informações referentes a controles provenientes de um sistema externo ou dados de monitoramento coletados de uma rede fazem o software se comportar de maneira específica.

Princípio 4. Os modelos que representam informação, função e comportamento devem ser divididos de modo a revelar detalhes em camadas (ou de maneira hierárquica). A modelagem de requisitos é a primeira etapa da solução de um problema de engenharia de software. Permite que se entenda melhor o problema e se estabeleçam bases para a solução (projeto). Os problemas complexos são difíceis de resolver em sua totalidade. Por essa razão, deve-se usar a estratégia “dividir e conquistar”. Um problema grande e complexo é dividido em subproblemas até que cada um seja relativamente fácil de ser compreendido. Esse conceito é denominado fracionamento ou separação por interesse e é uma estratégia-chave na modelagem de requisitos.

Princípio 5. A análise deve partir da informação essencial para os detalhes da implementação. A modelagem de análise se inicia pela descrição do problema sob o ponto de vista do usuário. A “essência” do problema é descrita sem levar em

consideração como será implementada uma solu- ção. Por exemplo, um jogo de videogame exige que o jogador “instrua” seu protagonista sobre qual direção seguir para continuar, conforme se envolve em situações perigosas. Essa é a essência do problema. O detalhamento da implementação (em geral descrito como parte do modelo de projeto) indica como a essência (do software) será implementada. No caso do videogame, talvez fosse usada entrada de voz. Por outro lado, poderia ser digitado um comando no teclado, um joystick (ou mouse) poderia ser apontado em uma direção específica, um dispositivo sensível ao movimento poderia ser agitado no ar ou poderia ser usado um dispositivo que lê diretamente os movimentos do corpo do jogador.

2.3.2 Princípios da Modelagem de Projeto

Princípio 1. O projeto deve ser alinhado para o modelo de requisitos. O modelo de requisitos descreve a área de informação do problema, funções visíveis ao usuário, desempenho do sistema e um conjunto de classes de requisitos que empacota objetos de negócios com os métodos a que servem. O modelo de projeto traduz essa informação em uma arquitetura, um conjunto de subsistemas que implementam funções mais amplas e um conjunto de componentes que são a concretização das classes de requisitos. Os elementos da modelagem de projetos devem ser alinhados para a modelagem de requisitos.

Princípio 2. Sempre considere a arquitetura do sistema a ser construído. A arquitetura de software é a espinha dorsal do sistema a ser construído. Afeta interfaces, estruturas de dados, desempenho e fluxo de controle de programas, a maneira pela qual os testes podem ser conduzidos, a manutenção do sistema realizada e muito mais. Por todas essas razões, o projeto deve começar com as considerações arquiteturais. Só depois de a arquitetura ter sido estabelecida devem **ser considerados os elementos relativos aos componentes.**

Princípio 3. O projeto de dados é tão importante quanto o projeto das funções de processamento. O projeto de dados é um elemento essencial do projeto da arquitetura. A forma como os objetos de dados são percebidos no projeto não pode ser deixada ao acaso. Um projeto de dados bem estruturado ajuda a simplificar o fluxo do programa e torna mais fácil a elaboração do projeto e a implementação dos componentes de software, tornando mais eficiente o processamento como um todo.

Princípio 4. As interfaces (tanto internas quanto externas) devem ser projetadas com cuidado. A forma como os dados fluem entre os componen- tes de um sistema tem muito a ver com a eficiência do processamento, com a propagação de erros e com a simplicidade do projeto. Uma interface bem elaborada facilita a integração e auxilia o responsável pelos testes quanto à validação das funções dos componentes.

Princípio 5. O projeto da interface do usuário deve ser voltado às necessidades do usuário, mas ele sempre deve enfatizar a facilidade de uso. A interface do usuário é a manifestação visível do software. Não importa quão sofisticadas sejam as funções internas, quão amplas sejam as estruturas de dados, quão bem projetada seja a arquitetura; um projeto de interface deficiente leva à percepção de que um software é "ruim".

Princípio 6. O projeto no nível de componentes deve ser funcionalmente independente. Independência funcional é uma medida para a "mentalidade simplificada" de um componente de software. A funcionalidade entregue por um componente deve ser coesa – isto é, concentrar-se em uma, e somente uma, função ou subfunção.

Princípio 7. Os componentes devem ser relacionados livremente tanto entre componentes quanto com o ambiente externo. O relacionamento é obtido de várias maneiras – via interface de componentes, por meio de mensagens, por meio de dados em geral. À medida que o nível de correlação aumenta, a tendência para a propagação do erro também aumenta, e a manutenção geral do software decresce. Portanto, a dependência entre componentes deve ser mantida o mais baixo possível.

Princípio 8. Representações de projetos (modelos) devem ser de fácil compreensão. A finalidade dos projetos é transmitir informações aos desenvolvedores que farão a codificação, àqueles que irão testar o software e a outros que possam vir a dar manutenção futuramente. Se o projeto for de difícil compreensão, não servirá como meio de comunicação efetivo.

Princípio 9. O projeto deve ser desenvolvido iterativamente. A cada iteração, o projetista deve se esforçar para obter maior grau de simplicidade. Como todas as atividades criativas, a elaboração de um projeto ocorre de forma iterativa. As primeiras iterações são realizadas para refinar o projeto e corrigir erros; entretanto, as iterações finais devem dirigir esforços para tornar o projeto tão simples quanto possível.

Princípio 10. A criação de um modelo de projeto não exclui uma abordagem ágil. Alguns proponentes do desenvolvimento de software ágil insistem em que o código é a única documentação de projeto necessária. Contudo, o objetivo de um modelo de projeto é ajudar outros que deverão manter e evoluir o sistema. É extremamente difícil entender o objetivo de nível mais alto de um trecho de código ou suas interações com outros módulos em um moderno ambiente de execução (run-time) e multiprocessos (multithread).

2.4 Princípios da Construção

2.4.1 Princípios da Codificação

Antes de escrever uma linha de código, certifique-se de que:

• Compreendeu bem o problema a ser solucionado.

• Compreendeu bem os princípios e conceitos básicos sobre o projeto.

• Escolheu uma linguagem de programação adequada às necessidades do software a ser desenvolvido e ao ambiente em que ele vai operar.

• Selecionou um ambiente de programação que forneça ferramentas para tornar seu trabalho mais fácil.

• Elaborou um conjunto de testes de unidade que serão aplicados assim que o componente codificado estiver completo.

Ao começar a escrever código:

• Restrinja seus algoritmos seguindo a prática de programação estrutura-da.

• Pense na possibilidade de usar programação em pares.

• Selecione estruturas de dados que atendam às necessidades do projeto.

• Domine a arquitetura de software e crie interfaces coerentes com ela.

• Mantenha a lógica condicional tão simples quanto possível.

• Crie “loops” agrupados de tal forma que testes sejam facilmente aplicáveis.

• Escolha denominações de variáveis significativas e obedeça a outros padrões de codificação locais.

• Escreva código que se documente automaticamente.

• Crie uma disposição (layout) visual (por exemplo, recuos e linhas em branco) que auxilie a compreensão.

Após completar a primeira etapa de codificação, certifique-se de:

• Aplicar uma revisão de código quando for apropriado.

• Realizar testes de unidades e corrigir erros ainda não identificados.

• Refabricar o código.

2.4.2 Princípios de Testes

Princípio 1. Todos os testes devem estar alinhados com os requisitos do cliente. O objetivo do teste de software é descobrir erros. Constata-se que os efeitos mais críticos do ponto de vista do cliente são aqueles que conduzem a falhas no programa quanto a seus requisitos.

Princípio 2. Os testes devem ser planejados muito antes de serem iniciados. O planejamento dos testes pode começar assim que o modelo de requisitos estiver concluído. A definição detalhada dos casos de teste pode começar assim que o modelo de projeto tenha sido solidificado. Portanto, todos os testes podem ser planejados e projetados antes que qualquer codificação tenha sido gerada.

Princípio 3. O princípio de Pareto se aplica a testes de software. Neste contexto, o princípio de Pareto indica que 80% de todos os erros revelados durante testes provavelmente estarão em aproximadamente 20% de todos os componentes do programa. O problema, evidentemente, consiste em isolar os componentes suspeitos e testá-los por completo.

Princípio 4. Os testes devem começar “no particular” e progredir para o teste “no geral”. Os primeiros testes planejados e executados geralmente se concentram nos componentes individuais. À medida que os testes progridem, o enfoque muda para tentar encontrar erros em grupos de componentes integrados e, posteriormente, no sistema inteiro.

Princípio 5. Testes exaustivos são impossíveis. A quantidade de trocas de direção, mesmo para um programa de tamanho moderado, é excepcionalmente grande. Por

essa razão, é impossível executar todas as rotas durante os testes. O que é possível, no entanto, é cobrir adequadamente a lógica do programa e garantir que todas as condições referentes ao projeto no nível de componentes sejam exercidas.

Princípio 6. Aplique a cada módulo do sistema um teste equivalente à sua densidade de defeitos esperada. Frequentemente, esses são os módulos mais recentes ou os que são menos compreendidos pelos desenvolvedores.

Princípio 7. Técnicas de testes estáticos podem gerar resultados importantes. Mais de 85% dos defeitos de software são originados na sua documentação (requisitos, especificações, ensaios de código e manuais de usuário). Testar a documentação do sistema pode ser vantajoso.

Princípio 8. Rastreie defeitos e procure padrões em falhas descobertas pelos testes. O total dos defeitos descobertos é um bom indicador da qualidade do software. Os tipos de defeitos descobertos podem ser uma boa medida da estabilidade do software. Padrões de defeitos encontrados com o passar do tempo podem projetar os números de falhas esperadas.

Princípio 9. Inclua casos de teste que demonstrem que o software está se comportando corretamente. À medida que os componentes do software vão sendo mantidos ou adaptados, interações inesperadas causam efeitos colaterais involuntários em outros componentes. É importante ter um conjunto de casos de teste de regressão pronto para verificar o comportamento do sistema depois que alterações forem aplicadas a um produto de software.

2.5 Princípios da Disponibilização

Princípio 1. As expectativas do cliente para o software devem ser gerenciadas. Muitas vezes, o cliente espera mais do que a equipe havia prometido entregar e ocorre a decepção. Isso resulta em feedback não produtivo e arruína o moral da equipe. Em seu livro sobre gerenciamento de expectativas, Naomi Karten afirma: “O ponto de partida para administrar expectativas consiste em se tornar mais consciente sobre como e o que vai comunicar”. Ela sugere que um engenheiro de software deve ser cauteloso em relação ao envio de mensagens conflituosas ao cliente (por exemplo, prometer mais do que pode entregar racionalmente no prazo estabelecido ou entregar

mais do que o prometido para determinado incremento e, em seguida, menos do que prometera para o próximo).

Princípio 2. Um pacote de entrega completo deve ser montado e testado. Todo software executável, arquivo de dados de suporte, documento de suporte e outras informações relevantes deve ser completamente montado e testado com usuários reais em uma versão beta. Todos os roteiros de instalação e outros itens operacionais devem ser aplicados inteiramente no maior número possível de configurações computacionais (isto é, hardware, sistemas operacionais, dispositivos periféricos, disposições de rede).

Princípio 3. É preciso estabelecer uma estrutura de suporte antes da entrega do software. Um usuário conta com o recebimento de informações precisas e com responsabilidade caso surja um problema. Se o suporte for local, ou, pior ainda, inexistente, imediatamente o cliente ficará insatisfeito. O suporte deve ser planejado, seus materiais devem estar preparados, e mecanismos para manutenção de registros apropriados devem estar determinados para que a equipe de software possa oferecer uma avaliação de qualidade das formas de suporte solicitadas.

Princípio 4. Material instrucional adequado deve ser fornecido aos usuários. A equipe de software deve entregar mais do que o software em si. Auxílio em treinamento de forma adequada (se solicitado) deve ser desenvolvido; orientações quanto a problemas inesperados devem ser oferecidas; quando necessário, é importante publicar uma descrição sobre as “diferenças existentes no incremento de software”.

Princípio 5. Software com bugs deve ser primeiramente corrigido e, depois, entregue. Sob a pressão do prazo, muitas empresas de software entregam incrementos de baixa qualidade, notificando o cliente de que os bugs “serão corrigidos na próxima versão”. Isso é um erro. Há um ditado no mercado de software: “Os clientes esquecerão a entrega de um produto de alta qualidade em poucos dias, mas jamais esquecerão os problemas causados por um produto de baixa qualidade. O software os lembra disso todos os dias”.