Resumo da parte 5.5 – **Características de um bom projeto**

Quando projetamos um sistema, queremos garantir a inclusão de alguns atributos de qualidade. Os projetos de boa qualidade deveriam ter características que levassem a produtos de qualidade: facilidade de entendimento; facilidade de implementação; facilidade de realização de testes; facilidade de modificação e de tradução correta das especificações de requisitos. A capacidade de se fazer modificações é especialmente importante, pois as mudanças nos requisitos ou as mudanças necessárias para a correção de defeitos, algumas vezes, resultam em uma modificação no projeto.

**Independência dos componentes**

Em um bom projeto de software se torna essencial a independência dos componentes, pois:

* Torna-se mais fácil modificar um componente independente;
* Torna mais fácil a compreensão dos programas, principalmente os maiores e mais complexos;
* Facilidade de testar e encontrar erros;
* Facilidade de manutenção;

Para reconhecer e medir o grau de independência dos componentes de um projeto utilizamos os conceitos: Acoplamento e coesão.

**Acoplamento**

Dizemos que dois componentes têm um alto grau de acoplamento quando existe uma grande dependência entre eles. São pouco associados quando têm alguma dependência, mas suas interconexões são pequenas. E sem acoplamento quando não possuem nenhuma interconexão entre si, sendo completamente independentes.

Os graus de acoplamento são divididos em cinco níveis/subgrupos: *Acoplamento por conteúdo, acoplamento comum, acoplamento por controle, acoplamento por identificação e acoplamento por dados.*

Dos cinco citados, o bom projeto almeja o acoplamento de dados, pois é o mais simples e permite menos possibilidade de erros. Uma vez que é mais fácil rastrear apenas os dados e fazer as modificações.

**Coesão**

Quanto mais coeso o componente, mais relacionadas estão as partes internas do componente entre si e ao seu propósito geral. Em outras palavras, um componente é coeso se todos os seus elementos estiverem direcionados à realização da mesma tarefa e forem essenciais a ela.

Assim como ocorre em acoplamentos, os tipos de coesão também estão divididos, porem em sete tipos/subgrupos: coesão funcional, sequencial, de comunicação, procedural, temporal, logica e coincidente.

Dos sete tipos citados, nosso ideal é a coesão funcional, em que todo elemento processado é essencial para o desempenho de uma única função , e todos elementos essenciais estão contidos em um componente. Um componente funcionalmente coeso realiza somente a função para qual é designado, e realiza apenas essa função, e nada mais.

**Identificação e tratamento da exceção**

Assim como acontece no trânsito, quando dirigimos na defensiva, também devemos realizar um projeto baseando-se “na defensiva”, ou seja, tentando prever as situações que podem levar a problemas no sistema. Criar um projeto assumindo uma postura defensiva não é fácil. A especificação de requisitos nos diz o que o sistema deve fazer, mas ela geralmente não informa o que o sistema não deve fazer.

Então incluímos o tratamento de exceções, de modo que o sistema aborde cada exceção de maneira satisfatória, sem afetar as funções do sistema.

As exceções típicas incluem:

* Falha em fornecer um serviço;
* Fornecer o serviço ou dados errados;
* Corromper os dados

Podemos tratar cada exceção que identificamos utilizando um dos três modos:

1. Tentar novamente: restabelecemos o sistema ao seu estado anterior e tentamos novamente realizar o serviço, utilizando uma estratégia diferente.
2. Corrigir: restabelecemos o sistema ao seu estado anterior, corrigimos alguns aspectos do sistema e tentamos novamente realizar o serviço, utilizando a mesma estratégia.
3. Fazer um relatório: restabelecemos o sistema ao seu estado anterior, relatamos o problema para um componente de tratamento de erros e não realizamos o serviço.

Há diversas técnicas que podem ser utilizadas no projeto, a fim de identificar as exceções à medida que o código está sendo executado:

* Soma verificadora e verificação de dígitos: verifica duas vezes a precisão dos dados e dos cálculos.
* Links redundantes: incluindo ponteiros pra frente e pra trás.
* Registradores de tempo.

**Prevenção de defeitos e tolerância a defeitos**

Os projetos devem tentar antecipar os defeitos e resolve-los, de modo a minimizar as possíveis interrupções e maximizar a segurança. Mas também é importante nos prevenirmos contra defeitos existentes em cada componente, assim como contra defeitos introduzidos por outros componentes, sistemas e interfaces.

Os defeitos são gerados por erros humanos em algum produto do software. Esses defeitos de projeto podem ser propagados como um código incorreto, instruções incorretas no manual do usuário, ou como uma bateria de testes incorretos. Desta maneira um simples erro pode geral um ou mais defeitos, em um ou mais produtos.

* **Falha vs. Defeito**

Distinguimos os defeitos de falhas. Uma falha se refere a quando um sistema se desvia do seu comportamento requerido. Falhas podem ser descobertas antes ou depois da entrega do sistema, pois elas podem ocorrer tanto nos testes como durante a operação. De certa maneira, as falhas e os defeitos oferecem visões internas e externas do sistema. Em outras palavras, os defeitos representam problemas que os desenvolvedores podem verificar, enquanto as falhas são problemas que os usuários ou clientes observam.

**Detecção passiva de defeitos**: É quando projetamos um sistema pra que ele espere até que uma falha ocorra durante a execução.

**Detecção ativa de defeitos**: É quando, periodicamente, procurarmos identificar ‘sintomas’ de defeitos, ou se tentarmos prever quando as falhas ocorrerão.

**Correção de defeitos**: é a compensação do sistema para a presença do defeito.

Nossos projetos devem incluir uma estratégia para lidar com defeitos, à medida que eles são encontrados. Frequentemente interrompemos o sistema quando o defeito o afeta de alguma maneira ou, simplesmente, registramos a existência da falha, observamos o estado do sistema no momento em que a falha ocorreu, e retornamos para corrigir o erro posteriormente.

**Tolerância a defeitos**

Muitas vezes, a correção de um defeito é muito dispendiosa, arriscada ou inconveniente. Em vez disso nosso projeto minimiza os danos causados pelo defeito, resultando em uma pequena interrupção para os usuários. A tolerância de defeitos é o isolamento do dano causado por um defeito, e ela é conveniente, ou até mesmo desejável, em algumas circunstâncias.

Algumas estratégias de tolerância a defeitos dependem da habilidade de prever a localização de defeito se do controle de tempo das falhas. Alguns defeitos são fáceis de prever, mas os sistemas mais complexos são mais difíceis a serem analisados. Ao mesmo tempo, nos sistemas mais complexos, é maior a probabilidade de haver defeitos significativos.

Assim, existem estratégias de tolerância a defeitos que isolam as áreas com prováveis defeitos, em vez de provar o defeito real em si.