Acoplamento E Coesão

Acoplamento e Coesão talvez sejam as características mais importantes de qualquer sistema. Muitos sistemas são como um Castelo de Cartas. Num Castelo de Cartas, ao tirar uma carta da estrutura, a probabilidade de estragos no castelo é alta. Se a carta estiver na base (parte inferior) do castelo, é quase certo que o castelo irá desmoronar.

E nos sistemas, não é muito diferente. Todo profissional da área já passou por algum sistema onde corrigi-se um bug e aparecem outros vinte; exclui-se uma funcionalidade e outras várias param de funcionar ou apresentam defeitos etc.

Basicamente é isso mesmo. Quando falamos de acoplar uma coisa em outra coisa, estamos falando de conectar estas duas coisas. Uma Roda está acoplada a um Carro. Uma Bateria está acoplada a um Smartphone. Uma Cabeça está acoplada a um Corpo.

Na estrutura de um sistema temos acoplamento em todo lugar, em qualquer fase do projeto, em qualquer fragmento de escopo do software. Quando falamos, por exemplo, de relacionamento entre Classes, Tabelas, Domínios, Sub-Sistemas, Casos de Uso etc. estamos falando de acoplamento.

Podemos afirmar que no contexto de um software qualquer relacionamento gera acoplamento.

Entendemos então que o acoplamento é algo determinístico ao processo de produção de software, ou seja, fatalmente estará presente neste tipo de atividade e em suas entregas. E tem que estar mesmo, pois as partes do todo precisam estar conectadas para se formar o todo (que no nosso caso, é um sistema).

Acoplamento Forte e Acoplamento Fraco

Sempre haverá acoplamento na estrutura de um software, é fato.

Mas uma coisa é haver a necessidade natural do acoplamento, outra é o “nível dessa medida”.

Devemos “medir” o nível de acoplamento sempre que lidamos com produção de software.

Mas medir isso é num nível de detalhe muito baixo é algo muito difícil, talvez impraticável para a maioria das empresas e profissionais da área.

Para viabilizar essa mensuração utilizamos duas formas de classificar o nível de acoplamento: Forte Acoplamento e Fraco Acoplamento.

Quando um sistema possui entre seus componentes uma relação de interdependência fraca, significa que a dependência entre seus componentes é baixa, ou seja, estão acoplados, mas fracamente acoplados. Isso chamamos de Fraco Acoplamento.

/* No contexto acima, entenda Componentes como qualquer unidade produzida no sistema, de qualquer tamanho. Seja Módulo, Funcionalidade, Classe, Função, Tabela etc. */

Quando um sistema possui entre seus componentes uma relação de interdependência forte, significa que a dependência entre seus componentes é alta, ou seja, estão acoplados, mas fortemente acoplados. Isso chamamos de Forte Acoplamento.

/* Dependendo da literatura consultada a terminologia pode sofrer alguma variação. Ao invés de Forte Acoplamento pode-se dizer Alto Acoplamento e ao invés de Fraco Acoplamento pode-se dizer Baixo Acoplamento. */

Mas como sabemos, software é algo que demanda manutenção constante, pois é um “organismo vivo”, só “morre” quando é descontinuado. E dificilmente sistemas são descontinuados.

Pensemos no castelo de cartas que citamos no início do post. Dar manutenção nele, tirar uma carta e colocar outra, ou ajustar o posicionamento de uma carta, é tarefa simples? Não, sem dúvida.

Mas qual a causa dessa dificuldade?

O castelo de cartas possui uma estrutura com Forte Acoplamento, onde suas partes possuem uma dependência muito alta umas com as outras, e isso torna sua estrutura complicadíssima de ser alterada.

O mesmo raciocínio se aplica a software. Um software com Forte Acoplamento torna-se um problema na hora de mantê-lo, devido ao alto grau de interdependência entre seus componentes.

Para evitar isso, devemos pensar software sempre com Fraco Acoplamento, focando num nível baixo de interdependência entre seus componentes, para que mantê-lo seja algo menos problemático de fazê-lo.

Coesão, do ponto de vista de significado da palavra, é um termo mais abstrato do que Acoplamento. Mais complexo para compreendermos.

Segundo o Google, Coeso é:

"(…) que apresenta harmonia; ajustado, concorde. (…)"

No contexto de um software podemos entender que Acoplamento é uma medida "inter" componentes, e Coesão é uma medida "intra" componentes.

A primeira refere-se ao mundo externo do componente, como ele se inter-relaciona com os outros componentes do sistema.

A segunda refere-se ao mundo interno do componente, como ele se intra-relaciona consigo mesmo.

Sempre ouvimos falar das duas juntas: Acoplamento e Coesão.

Cada parte de um todo possui seu contexto único, seu escopo, suas responsabilidades. Na estrutura de um software, cada um de seus componentes possuirá coesão, é algo determinístico também. São partes de um todo, e cada parte possui um conteúdo.

Uma "parte vazia" é um contrassenso, pois não teria razão de existir. É necessário haver um "conteúdo" para uma parte, e havendo conteúdo, há coesão.

Alta Coesão e Baixa Coesão

No castelo de cartas, cada carta possui um conteúdo, que é utilizado no contexto do baralho.

Um "As de Copas" é uma carta que, como as outras, possui um naipe (copas) e um número ou letra (A).

E essa carta possui apenas uma função: ser um "As de Copas". Seu conteúdo é este. Ela faz tudo que um "As de Copas" tem que fazer, e nada mais.

Seria viável jogar baralho (qualquer modalidade que seja) com cartas com mais de um naipe, ou mais de um número ou letra? Seria uma confusão total.

/* O conceito da Coesão está intimamente ligado ao Princípio da Responsabilidade Única (SRP) do SOLID. Entretanto os profissionais da nossa área costumam achar que o SRP aplica-se apenas a Classes e Objetos, mas é aderente e necessário há qualquer artefato contido no modelo do software. Vamos falar mais disso ao fim deste post. */

Uma carta de um baralho não pode possuir dois naipes, nem ter dois números, pois assim tem mais de uma responsabilidade, faria mais de uma coisa.

Um componente de software sempre possui alguma Coesão, como falamos. Mas esta Coesão pode ser Alta ou Baixa (ou Fraca ou Forte, dependendo da literatura consultada). E qual a diferença entre ambas?

Um componente com Alta Coesão é um componente que possui apenas uma única responsabilidade, que possui em seu conteúdo/suas funções, apenas aquilo que realmente deve fazer.

Ele não assume responsabilidades de outros componentes e não as mistura com as suas, nem delega suas responsabilidades a outros componentes, e não depende dos outros componentes para realizar suas funções conforme suas responsabilidades.

Já um componente com Baixa Coesão é o inverso disso. Possui responsabilidades além das suas, depende de outros componentes para realizar suas funções etc.

E o efeito da causa “Baixa Coesão” nos componentes de um software é semelhante ao efeito gerado pela causa “Forte Acoplamento”, onde podemos destacar:

Dificuldades para dar manutenção no software – o grau desta dificuldade será proporcional à medida de Acoplamento e Coesão.

Dificuldade de reuso de componentes – reuso não combina com responsabilidades misturadas, com falta de unidade.

Dificuldades de interpretação clara dos modelos utilizados no projeto – gerando baixa performance na equipe devido ao custo adicional para entender o sistema.

Quanto mais baixa for a coesão nos componentes de um software, mais problemático será mantê-lo.

Acoplamento e Coesão aplicam-se apenas à Modelagem de Classes?

Categoricamente: não!

A Modelagem de Classes tem como objetivo definir a estrutura de um sistema (considerando um sistema que será produzido utilizando alguma linguagem de programação Orientada a Objetos).

Mas o Modelo de Classes não é o único modelo produzido no projeto de um software. Temos Modelo de Dados, Modelo de Requisitos, Modelo de Casos de Uso, Modelos de Interfaces Gráficas, e alguns outros.

Não necessariamente todo projeto vai produzir todos os modelos, isso dependerá, naturalmente, do processo de desenvolvimento que a empresa adotará no ciclo de vida de seus projetos.

Coesão

Se refere ao relacionamento que os membros de um módulo possuem, não importa o que módulo significa. Indica se os membros tem uma relação mais direta e importante. Códigos coesos são aqueles de relação forte, onde seus membros estão intimamente ligados e estão ali por um objetivo comum. Membros que não são absolutamente necessários para aquele módulo não devem estar presentes em códigos coesos.

Módulos coesos são aqueles que possuem poucas responsabilidades. Desta forma a manutenção é mais simples e evita efeitos colaterais. Fica mais fácil alterar uma parte da aplicação sem afetar outras partes. Por isto é confundido com o princípio da responsabilidade única. E assim alguns paradigmas que isolam coisa podem ser mais coesos do que outros que incentivam por tudo junto, ao contrário da crença popular.

É importante tirar funcionalidades supérfluas de um módulo e transferir para outro modulo.

Isso lembra um pouco a normalização do banco de dados que, em última instância, determina que cada tabela possua apenas uma coluna de dado e uma ligação com as demais.

Há ainda uma maior facilidade para lembrar do que os módulos fazem quando eles fazem pouca coisa. O que não é coeso, é confuso, incoerente.

Exemplos óbvios de baixa coesão são módulos que cuidam ao mesmo tempo das regras de negócio, persistência de dados e interação com usuário. Mas existem casos menos óbvios quando você estabelece que um cliente pode decidir quando ele é inadimplente ou quais os vendedores que o atendem.

É comum achar que isto pertence ao cliente mas os clientes existem independentes disto, e que clientes são papéis de pessoas em uma organização.

É extremamente comum vermos designs com baixa coesão, algumas em exagero, outras na mediada certa. Evidentemente que tentar a alta coesão a todo custo também se tornará um problema de design. Tudo tem uma desvantagem. Há casos de alta coesão em exagero. Raramente faz sentido ter um módulo com um único membro.

Formas de coesão (do pior para o melhor):

Por coincidência - Acontece sem planejamento, pode estar certo ou não. Em geral se considera como a pior coesão.

Lógica - Membros da mesma categoria lógica, mesma atividade, estão juntos.

Temporal - Se relacionam pelo momento que são executados.

Procedural - Formam uma sequência para realizar uma tarefa maior.

Comunicacional - Quando essa sequência de execuções ocorrem no mesmo dado.

Sequencial - Quando a saída de um membro serve de entrada para outro membro.

Funcional - Agrupamento ocorre só porque elas realmente precisam estar juntas para contribuir com algo muito bem definido.

Acoplamento

Se refere ao relacionamento entre os módulos. Indica quanto um módulo depende de outro para funcionar. Códigos desacoplados são aqueles de relação fraca, e não dependem de outros para fazer sua funcionalidade básica. É sempre desejável um baixo nível de acoplamento.

Quando há baixo acoplamento, a aplicação fica mais flexível, reusável e mais organizada. É possível intercambiar partes que se mostraram problemáticas, ultrapassadas ou que exigem novas funcionalidades.

Quando se busca coesão é comum fazer com que módulos novos sejam criados com alto acoplamento e uma nova avaliação se faz necessária para reduzir esta dificuldade.

Em geral o alto acoplamento mantêm o problema da baixa coesão, e de uma forma até mais complicada. Quando há uma mudança em um módulo altamente acoplado, gera a necessidade de mudanças em outros módulos, o que pode ser mais complicado de fazer e gera mais riscos do que um módulo com baixa coesão.

Um exemplo de acoplamento ruim é uma nota fiscal que pega dados de um imposto para fazer o cálculo. O ideal é que a estratégia de cálculo do imposto seja completamente independente e desconhecida da nota fiscal, assim pode ser trocado facilmente.

São raras as aplicações que usam o baixo acoplamento de forma adequada. E quando o fazem, em geral é seguindo um padrão de projeto consagrado. Mesmo assim de forma irregular. O que não seria algo de todo ruim se fosse sempre feito com consciência. O baixo acoplamento acaba tornando, por vezes, as aplicações mais complexas.

Formas de acoplamento (do pior para o melhor):

Conteúdo - Quando um módulo pode acessar, modificar ou se referenciar diretamente a conteúdos de outro módulo.

Global - Quando os módulos podem ler e escrever dados globais de um módulo.

Controle - Quando um modulo irá decidir como outro funcionará.

Estrutura de dados - Quando os dados de um módulo são passados em uma estrutura de dados passada como parâmetro.

Dados - Quando apenas os dados realmente necessários para outro módulo são passados como parâmetros

Não é possível eliminar o acoplamento por completo, apenas podemos mantê-lo o mais baixo possível.

Fazer com que módulos (classes) dependam especificamente um outro módulo, que exijam uma implementação específica, que não possa mudar o comportamento específico, ou mesmo que exija um outro módulo quando ele pode resolver de forma coesa, são exemplos de acoplamento excessivo.

Em OOP para reduzir o acoplamento devemos programar para a interface e não para a implementação.

Conceitos e paradigmas

Estes conceitos existem pelo menos desde a década de 60. E ao contrário do que muitos acreditam, não é um conceito intrínseco da orientação a objeto. Infelizmente algumas pessoas acham que só existe esse paradigma e acabam escrevendo coisas erradas na internet atribuindo informações incorretas ao que elas conhecem e gostam.

Já estava bem documentando em livros dos anos 70 e foram sendo disseminados nos livros mais atuais que privilegiam um paradigma sobre outros.

Tão pouco estão relacionados à linguagens. Na verdade o conceito transcende a computação.

As ferramentas de engenharia de software são bem antigas e podem ser aplicadas, de uma forma geral, a todos os paradigmas de programação. Quando isto é feito, alguns dos paradigmas que parecem ultrapassados se tornam relevantes.

É verdade que paradigmas como OOP podem ajudar a organizar o código melhor, mas quando a pessoa não sabe o que fazer, ela fará errado em todos os paradigmas. E de fato OOP ajuda em alguns pontos, mas complica em outros. É raro encontrar alguém que consiga definir classes bem coesas e desacopladas. Então estes conceitos se tornaram importantes neste paradigma. Em paradigmas mais simples e livres, quando usados da maneira correta, é mais fácil manter a coesão e o baixo acoplamento. Apesar de trazer algumas outras dificuldades de manutenção.

OOP incentiva a subclasse (herança) que é uma das piores formas de acoplamento existentes. OOP incentiva agrupar todos os comportamentos em uma classe, o que frequentemente é abusado. Para resolver isto uma série de padrões de projeto são criados, o que não seria necessário em outro paradigma.

Não estou dizendo que outros paradigmas são melhores e livres de defeito. Digo apenas que se a engenharia de software for aplicada corretamente, o paradigma não é tão importante.

O desenvolvimento de software de qualidade, dentro do prazo e com o custo estabelecido é essencial para a sobrevivência de qualquer organização desenvolvedora de software. A facilidade com que se dá a manutenção e o potencial de reuso de um software desempenham papel de destaque nesse contexto, e os conceitos de coesão e acoplamento podem auxiliar muito neste sentido.

Coesão e acoplamento em sistemas orientados a objetos:

Os conceitos de coesão e acoplamento, surgidos no contexto da análise e projeto estruturados, embora tenham um grande impacto na qualidade de sistemas, são geralmente desconhecidos ou negligenciados por desenvolvedores iniciantes de sistemas orientados a objetos. O desenvolvimento de softwares com alta coesão e fraco acoplamento facilita, entre outras coisas, a manutenção e o reuso. Assim sendo, o estudo desses conceitos e seus desdobramentos torna-se importante para melhorar a qualidade de sistemas.

Muitos desenvolvedores de software percorreram caminhos que passaram pelo desenvolvimento procedural antes de chegarem à orientação a objetos.

Desenvolver um programa procedural significa entender e conceber o programa como um conjunto de procedimentos (também conhecidos como rotinas, sub-rotinas ou funções) que manipulam dados. Neste modelo de desenvolvimento, os procedimentos são geralmente as unidades de subdivisão ou modularidade de sistemas.

Diferentemente do paradigma orientado a objetos, no qual a subdivisão de sistemas é baseada no mapeamento de objetos do domínio do problema para o domínio da solução, o desenvolvimento procedural não possui uma semântica forte que oriente a subdivisão de sistemas. Por isso, os procedimentos ou conjuntos relacionados de procedimentos de sistemas procedurais muitas vezes tratam de aspectos distintos do sistema e, consequentemente, apresentam certas características, como um alto grau de interdependência, que dificultam a manutenção e o reuso.

Neste contexto, muitos conceitos e métricas foram definidos para avaliar e auxiliar a subdivisão de sistemas. Dois destes conceitos são especialmente importantes por terem grande influência na qualidade dos sistemas desenvolvidos: coesão e acoplamento.

Os conceitos de coesão e acoplamento surgiram em meados da década de 1960, a partir de um estudo conduzido por Larry Constantine sobre o motivo pelo qual certos tipos de módulos de sistemas eram definidos e da análise dos pontos positivos e negativos relativos a estes tipos. Estes conceitos foram bastante estudados e utilizados no contexto da análise e projeto estruturado de sistemas.

A falta de conhecimento ou o negligenciamento destes conceitos e métricas contribui para que muitos desenvolvedores de sistemas orientados a objetos, principalmente iniciantes, desenvolvam sistemas com características indesejáveis.

O objetivo desse artigo é apresentar uma introdução aos conceitos de coesão e acoplamento, buscando demonstrar o efeito de desprezá-los na modelagem de sistemas orientados a objetos através de alguns exemplos simples. Os exemplos são apresentados na forma de diagramas de classes da UML e código Java.

Estendendo a definição clássica de coesão para o paradigma orientado a objetos, pode-se definir coesão de um sistema de software como a relação existente entre as responsabilidades de uma determinada classe e de cada um de seus métodos. Uma classe altamente coesa tem responsabilidades e propósitos claros e bem definidos, enquanto uma classe com baixa coesão tem muitas responsabilidades diferentes e pouco relacionadas.

Para se ter uma ideia de classes com níveis diferentes de coesão, vamos analisar dois exemplos de classes retiradas de um sistema bancário e de um sistema de locadora, representadas nas Figuras 1 e 2, respectivamente.

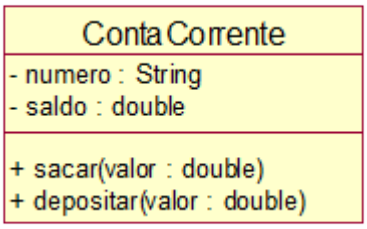

Figura 1. Exemplo de classe com um alto grau de coesão (sistema bancário).

No exemplo do sistema bancário, a classe ContaCorrente é responsável por operações relacionadas apenas a contas correntes: sacar() e depositar(). Nenhum outro “assunto” não relacionado a contas correntes é tratado por esta classe. Assim sendo, pode-se classificar esta classe como altamente coesa.