

INTEGRAÇÃO DE SISTEMAS LEGADOS

CONVERSA INICIAL

Nesta aula, serão explicados os conceitos de reengenharia de *software*. No segundo, descritos os modelos de reengenharia de *software* propostos. O terceiro tema trata da reengenharia de *software* para sistemas que seguem a arquitetura cliente servidor. O quarto tema descreve a reengenharia de *software* para o paradigma de orientação a objetos, enquanto o quinto dos padrões de projeto de *software*.

CONTEXTUALIZANDO

Nesta aula, serão vistos conceitos sobre reengenharia de *software*. Imagine um cenário em que um sistema legado deverá ainda ser mantido por vários anos, e a qualidade do seu código já não suporta uma grande quantidade de mudanças de forma adequada. Em uma situação como essa, a reengenharia de *software* pode ser uma alternativa mais barata e segura para a evolução do sistema, em vez de desenvolver um novo sistema por completo para substituir o sistema legado.

Como os sistemas legados costumam ser muito antigos, muitos deles podem ser baseados em *mainframe*, tecnologia popular das décadas de 1970 e 1980. Logo, a reengenharia deverá ter a preocupação de migrar o sistema para arquiteturas mais modernas, como a cliente servidor ou o paradigma de orientação a objetos. Padrões de projeto são extremamente úteis para utilização no processo de reengenharia, os quais serão tratados no fim desta aula.

A reengenharia busca modernizar sistemas legados a fim de estender seu ciclo de vida. A arquitetura cliente servidor, tratada nesta aula, serve para suportar clientes diversos; entre eles, dispositivos móveis.

TEMA 1 – CONCEITOS DE REENGENHARIA DE SOFTWARE

O processo de evolução de sistemas envolve o entendimento do *software*, que deve ser modificado para que se possa, posteriormente, modificá-lo. Todavia, especialmente em sistemas legados, essa tarefa é difícil de ser realizada. Os programas a serem modificados podem ter sido otimizados para ter um desempenho melhor ou uma melhor utilização de memória, em detrimento de uma fácil compreensão do seu funcionamento. Além disso, sua estrutura, ao longo do tempo, pode ter sido degradada por causa de uma série de mudanças.

Considere um programa com “módulos” de 2 000 linhas, com poucas linhas de comentário e que deve ser modificado para acomodar mudanças de requisitos. As seguintes opções podem ser consideradas:

1. modificar o código para fazer as alterações necessárias, lidando com a grande dificuldade de entendimento do programa e riscos de instabilidade das alterações em andamento;
2. entender melhor o funcionamento geral do programa, para tentar fazer com que as alterações sejam mais efetivas;
3. refazer a arquitetura, o código e os testes das partes do sistema relacionadas às modificações, aplicando conceitos de engenharia de *software*;
4. refazer a arquitetura, o código e os testes de todo o sistema, utilizando ferramentas automatizadas para entender sua arquitetura atual.

Não existe uma opção que seja a melhor. As circunstâncias podem implicar na escolha da primeira opção, mesmo que se tenha preferência pelas demais. Caso estejam previstas grandes modificações ou melhorias em pouco tempo, as opções 2, 3 ou 4 podem ser consideradas. Em um primeiro momento, a opção de refazer um grande sistema quando uma versão funcional já existe pode parecer extravagante. Antes de decidir por isso, deve-se considerar o seguinte:

- O custo de manutenção de uma linha de código pode ser de 20 a 40 vezes maior que seu desenvolvimento inicial.
- A reconstrução da arquitetura do *software*, utilizando conceitos modernos de arquitetura, pode facilitar muito a manutenção futura.
- Pelo fato de uma versão funcional do sistema já existir, a produtividade de desenvolvimento deve ser muito maior que a média.
- O usuário agora tem experiência com o sistema. Logo, novos requisitos e o direcionamento das mudanças podem ser feitos de forma mais fácil.
- Ferramentas automatizadas podem facilitar algumas partes do trabalho.
- Uma configuração completa do sistema (documentação, programas e dados) existirá depois do fim desse processo.

O processo de reengenharia aplica princípios de engenharia de *software* para recriar o sistema. Em muitos casos, o processo de reengenharia não cria apenas uma versão moderna equivalente ao sistema mais antigo. Novos requisitos de usuários e de tecnologia são integrados à nova versão do sistema. O novo sistema estende a capacidade da versão anterior.

Existem dois benefícios importantes da reengenharia, em comparação com a substituição de um sistema:

1. **redução de riscos** – existe um alto risco no desenvolvimento de um sistema crítico para o negócio de uma organização. Erros podem ser cometidos na especificação do sistema, ou podem ocorrer problemas de desenvolvimento. Atrasos em introduzir o novo *software* podem significar perdas para o negócio, além de custos adicionais.
2. **redução de custos** – o custo de atividades de reengenharia pode ser significativamente menor que o do desenvolvimento de um novo sistema.

Depois da finalização do processo de reengenharia, para fazer com que o sistema tenha interoperabilidade com novos módulos de *software*, a implementação de serviços de adaptação pode ser necessária. Esses serviços encapsulam as interfaces originais do sistema e apresentam interfaces novas e melhor estruturadas, que podem ser utilizadas por outros componentes. Esse processo de encapsulamento de sistemas legados é uma importante técnica para o desenvolvimento de serviços a serem utilizados em larga escala.

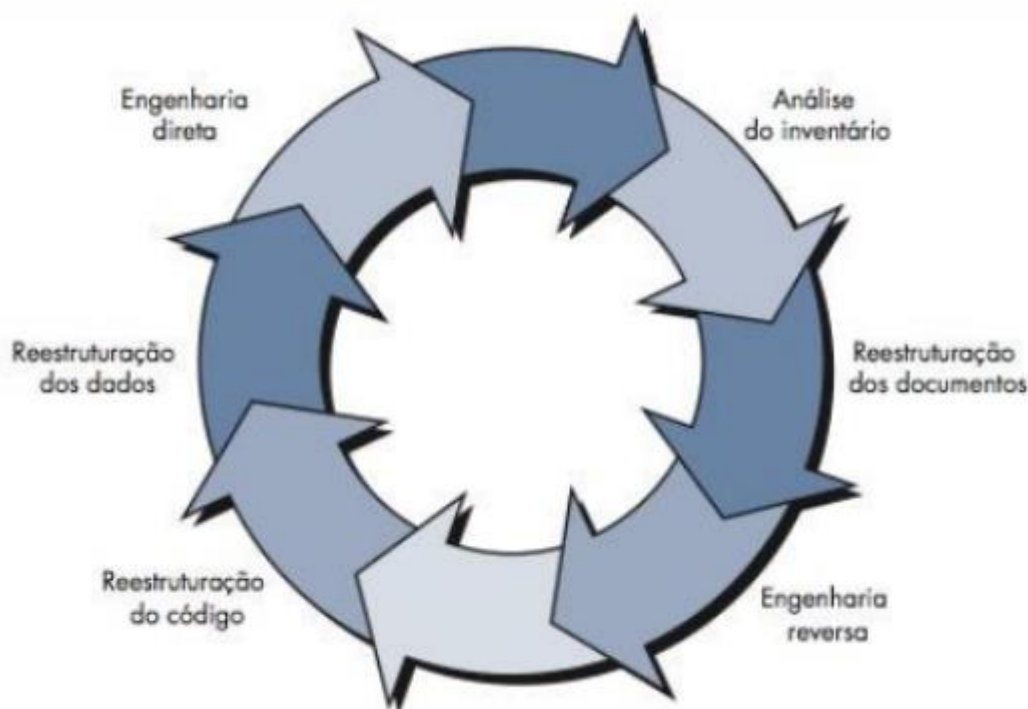
A desvantagem da reengenharia de *software* é que existem limites do quanto se pode melhorar um sistema. Dependendo da qualidade estrutural do sistema a ser convertido, muitas mudanças de arquitetura seriam requeridas, tornando todo o processo muito caro. Além disso, o sistema refeito não seria tão fácil de manter quanto um que foi desenvolvido com métodos de engenharia de *software* mais modernos.

TEMA 2 – MODELOS DE REENGENHARIA DE SOFTWARE

2.1. Modelo de reengenharia de *software* proposto por Roger Pressman

A figura 1 ilustra um modelo geral cíclico do processo de reengenharia proposto por Roger Pressman. Segundo esse modelo, as atividades de um processo de reengenharia são: análise de inventário, reestruturação da documentação, engenharia reversa, reestruturação de código, reestruturação de dados e engenharia direta (ou avante).

Figura 1 – Modelo de reengenharia de *software*



Fonte: Pressman, 2011, p. 668.

2.1.1 Análise de inventário

Qualquer organização deve ter um inventário de todas as suas aplicações. O inventário pode ser apenas uma planilha com informações detalhadas (tamanho, idade, criticidade etc.) de cada aplicação ativa. Classificando essas informações de acordo com a importância para os negócios, longevidade, manutenibilidade, entre outros critérios, obtém-se uma lista de aplicações candidatas a um processo de reengenharia. Esse processo de análise de inventário deve ser feito de forma cíclica com o passar do tempo.

2.1.2 Reestruturação de documentação

Documentação desatualizada é uma característica comum em sistemas legados. A seguir, encontram-se alternativas de como lidar com esse problema. A documentação de um sistema, quando for criada/recriada, deve ser a mínima possível para conter apenas o necessário para o entendimento do sistema.

- **Não atualizar a documentação:** documentação é um processo que envolve um grande esforço. Se o sistema funciona, você pode escolher

deixar a documentação como está, pois é praticamente impossível recriar a documentação de vários programas. Se um programa não sofre muitas modificações e está próximo de ser substituído, é melhor não lidar com sua documentação.

- **Atualizar a documentação parcialmente:** a documentação deve ser atualizada, mas a organização tem recursos limitados. Nesse caso, é melhor atualizar a documentação de acordo com as mudanças em andamento no sistema. Ao longo do tempo, o sistema terá uma documentação relevante sobre sua operação.
- **Atualizar a documentação totalmente:** o sistema é crítico para o negócio, e toda a sua documentação deve ser refeita. Mesmo nesse caso, a documentação deve ser feita para que tenha apenas o mínimo de informações importantes.

2.1.3 Engenharia reversa

Essa expressão tem origem na área de *hardware*, em que uma companhia desmonta um equipamento de um competidor para entender seu funcionamento. O funcionamento do equipamento poderia ser melhor compreendido se sua especificação tivesse sido obtida, mas a especificação não está disponível.

A engenharia reversa na área de *software* é parecida. Na maioria dos casos, todavia, o programa, para efetuar o processo de engenharia reversa, costuma ser da própria organização, e não de um competidor. Os “segredos” a serem compreendidos são obscuros porque nenhuma especificação foi desenvolvida. Logo, a engenharia reversa é o processo de análise de um programa, em um esforço para criar uma representação deste em um nível mais alto de abstração do que o código-fonte. A engenharia reversa pode ser interpretada como um processo de recuperação da especificação do sistema. Podem-se utilizar ferramentas para extrair dados que auxiliem no processo de entendimento.

2.1.4 Reestruturação de código (refatoração)

O tipo mais comum de reengenharia é a reestruturação de código, também chamada de refatoração de código. Alguns sistemas legados têm uma arquitetura relativamente sólida, mas seus módulos costumam ser feitos de forma que fica

difícil de entender, testar e manter. Nesses casos, o código desses módulos pode ser reestruturado.

Para ajudar a realizar essa atividade, o código fonte pode ser analisado com ferramentas de análise de código. Com base nos resultados da análise, pode ser feita a reestruturação do código. A documentação interna do código, caso exista, pode ser atualizada.

2.1.5 Reestruturação de dados

Um programa com uma arquitetura de dados deficiente será difícil de adaptar e melhorar. De fato, para muitas aplicações, a arquitetura de informação do sistema está mais relacionada à sua duração de longo prazo que a seu próprio código-fonte.

Ao contrário da reestruturação de código, que acontece em um nível mais baixo de abstração, a reestruturação de dados é uma atividade de reengenharia de larga escala. Em muitos casos, a engenharia reversa começa a partir da reestruturação de dados. A arquitetura de dados é inspecionada, e modelos de dados necessários são definidos. Objetos de dados e atributos são identificados, e estruturas de dados existentes são revisadas para melhoria de sua qualidade.

Quando a estrutura de dados é deficitária, os dados passam por um processo de reengenharia. Um exemplo de dados deficitários seria o armazenamento dos dados em arquivos texto, sem regras de normalização de dados que poderiam ser aplicadas, resultando em informações duplicadas, perda de desempenho no acesso a dados etc.

Pelo fato de que a arquitetura de dados tem uma grande influência na arquitetura do sistema e nos algoritmos que populam esses dados, mudanças nos dados podem vir a resultar em mudanças de arquitetura e de código no sistema.

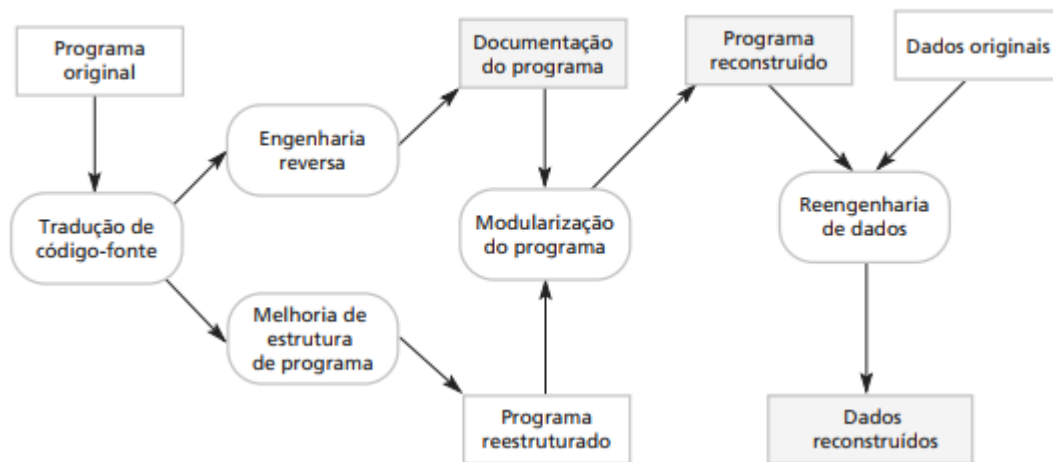
2.1.6 Engenharia direta/avante

Essa expressão vem do inglês *forward engineering*. É o processo contrário ao da engenharia reversa, em que as especificações de mais alto nível obtidas a partir do passo da engenharia reversa são agora traduzidas em código. Parte do processo de engenharia direta consiste nas reestruturações de código e de dados, mas sua etapa final consiste na revisão da arquitetura geral do novo sistema, podendo ser interpretado como um *refactoring* de arquitetura. Padrões de projeto de *software* podem ser utilizados como base para essa atividade.

2.2 Modelo de reengenharia de *software* de Ian Sommerville

Os passos de engenharia reversa, reestruturação de código, reestruturação de dados e engenharia direta podem ser melhor compreendidos na figura 2, que ilustra um modelo de reengenharia de dados proposto por Ian Sommerville.

Figura 2 – Processo de reengenharia



Fonte: Sommerville, 2011, p. 175.

Observe, a seguir, uma descrição dos processos que compõem esse modelo de reengenharia de sistemas.

- **Tradução de código-fonte:** com uma ferramenta de tradução, o programa é convertido de uma linguagem de programação antiga para uma versão mais moderna da mesma linguagem, ou para outra diferente, caso uma ferramenta como esta esteja disponível.
- **Engenharia reversa:** o código-fonte do programa é analisado para obter informações sobre sua estrutura. Com base nessas informações, uma nova documentação do sistema pode ser gerada. Isso pode, por exemplo, ser feito com ferramentas de geração de diagramas de UML, que podem gerar diagramas de classes, de sequência etc.
- **Melhoria da estrutura do programa:** a estrutura de controle do programa é analisada e modificada para fazer com que seja mais fácil de ler e entender. Esse processo pode ser automatizado parcialmente.
- **Modularização:** partes comuns do sistema são agrupadas e, quando apropriado, a redundância de código é removida. Em alguns casos, esse passo pode envolver a refatoração da arquitetura do sistema.

- **Reengenharia de dados:** esse passo pode compreender a redefinição de esquemas de banco de dados, além da conversão dos bancos de dados para a nova estrutura. Deve-se, em geral, efetuar uma revisão, ou “limpeza”, dos dados, corrigindo erros, removendo registros duplicados etc. Existem ferramentas para suportar esse tipo de atividade.

O processo de reengenharia não precisa, necessariamente, abranger todas as atividades descritas na figura 2. Por exemplo, não é necessário traduzir o código-fonte do sistema se a linguagem de programação em que este foi feito continuar sendo utilizada.

Com relação ao modelo cíclico de Pressman apresentado anteriormente, o passo de aperfeiçoamento do programa equivale ao de reestruturação de código, enquanto o passo de modularização pode ser interpretado como os de reestruturação de código e de engenharia direta.

Na figura 2, apesar do processo de reengenharia de dados considerar o código do sistema refeito como uma de suas entradas, a reengenharia de dados pode causar mudanças no código do sistema, inclusive de nível arquitetural. Isso acontece porque a arquitetura dos dados do sistema, muitas vezes, está relacionada à arquitetura do código do sistema, como descrito na seção 2.1.5.

TEMA 3 – REENGENHARIA DE SISTEMAS LEGADOS PARA ARQUITETURA CLIENTE SERVIDOR

Nas últimas décadas, muitas aplicações de *mainframe* foram refeitas para suportar arquiteturas do tipo cliente servidor, incluindo aplicações *web* e de dispositivos móveis. Em essência, recursos centralizados de computação, incluindo *softwares*, são distribuídos entre muitas plataformas de clientes. Mesmo que uma grande variedade de ambientes distribuídos possa ser especificada, a aplicação típica de *mainframe* que passa por um processo de reengenharia para suportar uma arquitetura cliente servidor tem as seguintes características:

- funcionalidade de aplicações migra para cada cliente;
- novas interfaces de usuário são implementadas nos clientes;
- funções de banco de dados são alocadas no servidor;
- funcionalidade especializada (por exemplo, análise computacional intensiva) pode permanecer no servidor;

- novos requisitos de comunicações, segurança, arquivamento e controle devem ser estabelecidos tanto no lado do servidor quanto no lado dos clientes. (Pressman, 2011)

É importante ressaltar que a migração do *mainframe* para uma arquitetura cliente servidor requer a reengenharia de processos de *software*. Adicionalmente, uma infraestrutura de comunicação organizacional deve ser estabelecida.

A reengenharia para aplicações cliente servidor começa com uma extensa análise do ambiente de negócio que compreende o sistema atual. Três camadas de abstração podem ser identificadas. O banco de dados se encontra na base de uma arquitetura cliente servidor, gerenciando transações e consultas de aplicações situadas em um servidor de aplicação. Adicionalmente, essas transações e consultas devem ser controladas em um contexto de regras de negócio, definidas por um processo de negócio existente. Aplicações cliente disponibilizam funcionalidades para a comunidade de usuários.

As funções do sistema de gerenciamento e a arquitetura de dados do banco de dados existente devem passar por um processo de engenharia reversa para ser possível efetuar uma nova especificação da camada de banco de dados. Em alguns casos, um novo modelo de dados é criado. Em cada caso, um banco de dados cliente servidor é refeito para assegurar que as transações sejam executadas de maneira consistente, que todas as modificações sejam feitas por usuários autorizados, que regras de negócio sejam reforçadas (por exemplo, não permitir que um dado seja excluído sem que antes sejam excluídos dados relacionados), que pesquisas sejam executadas de forma eficiente e que as funcionalidades de arquivamento tenham sido estabelecidas.

A camada de negócios representa um *software* que se encontra tanto no lado do servidor quanto no lado do cliente. Esse *software* efetua o controle e a coordenação de tarefas para assegurar que as transações e as consultas entre a aplicação cliente e o banco de dados estejam de acordo com os processos de negócio existente.

TEMA 4 – REENGENHARIA DE SISTEMAS LEGADOS PARA ARQUITETURA ORIENTADA A OBJETOS

Engenharia de *software* orientada a objetos se tornou o principal paradigma de desenvolvimento entre muitas organizações de desenvolvimento de *software*. Mas e quanto a aplicações existentes que foram desenvolvidas utilizando-se

métodos mais antigos de desenvolvimento? Em alguns casos, a resposta é deixar essas aplicações como estão. Em outros, essas aplicações devem ser refeitas para que possam ser integradas a grandes sistemas orientados a objetos.

O processo de reengenharia de *software* para gerar, como saída, um sistema orientado a objetos, deve acontecer da seguinte forma: Em primeiro lugar, o *software* existente deve passar por um processo de engenharia reversa para que seus modelos de dados, funcionais e de comportamento possam ser gerados. Se o sistema a ser feito terá novas funcionalidades, diagramas de caso de uso (UML) deverão ser criados para especificar os novos requisitos. Os modelos de dados criados durante o processo de engenharia reversa são utilizados com modelos de dados adicionais, referentes a novos requisitos, para estabelecer a base das classes de definição do sistema. Hierarquias de classes, modelos de relacionamento entre objetos e subsistemas são definidos e, assim, começa a arquitetura orientada a objetos do sistema.

Assim que as atividades migram da fase de análise para a de definição da arquitetura, um modelo de componentes deve ser feito. Se a aplicação existente se situa em um domínio composto de várias aplicações orientadas a objetos, é provável que uma biblioteca de componentes já exista e que possa ser utilizada durante o processo de reengenharia.

Caso não existam bibliotecas com classes orientadas a objetos a serem reaproveitadas, deve ser possível reutilizar algoritmos e estruturas de dados de aplicações não necessariamente orientadas a objetos.

TEMA 5 – PADRÕES DE PROJETO

Em engenharia de *software*, um padrão de projeto é uma solução generalizada e reutilizável para um problema recorrente em projetos de *software*. Um padrão de projeto não é um padrão finalizado, que pode ser aplicado diretamente em uma codificação. É uma descrição, ou um modelo, de como resolver um problema, que pode ser utilizado em diferentes situações.

Padrões de projeto podem acelerar o desenvolvimento de *software*, disponibilizando paradigmas de desenvolvimento testados e comprovados. Projetos de *software* efetivos consideram problemas que podem não estar aparentes até o final do desenvolvimento. A reutilização de padrões de projeto auxilia a prevenir problemas súbitos, que podem causar outros graves problemas,

e melhoram o entendimento do código para desenvolvedores habituados com esses padrões.

Frequentemente, as pessoas apenas entendem como aplicar certas técnicas de desenvolvimento de *software* para resolver um conjunto limitado de problemas. Essas técnicas são difíceis de aplicar na resolução de um conjunto maior de problemas. Os padrões de projeto disponibilizam soluções genéricas, documentadas em um formato que não requer particularidades relacionadas a um determinado problema.

O mais importante sobre os padrões é que eles são soluções aprovadas. Cada catálogo inclui apenas padrões que foram considerados úteis por diversos desenvolvedores em vários projetos. Os padrões catalogados também são bem definidos; os autores descrevem cada padrão com muito cuidado e em seu próprio contexto, portanto, será fácil aplicar o padrão em suas próprias circunstâncias. Eles também formam um vocabulário comum entre os desenvolvedores. (Ferreira, 2017).

5.1. Antipadrões

Os padrões são um mapa, não uma estratégia. Os catálogos geralmente apresentarão algum código-fonte como uma estratégia de exemplo, portanto não devem ser considerados como definitivos. Os padrões não ajudarão a determinar qual aplicação você deve estar escrevendo apenas como implementar melhor a aplicação [...]. Os padrões ajudam com o que e como, mas não com por que ou quando.

O conceito de utilizar os padrões de forma errônea é conhecido como antipadrões (*anti patterns*). [...] se um padrão representa a “melhor prática”, então um antipadrão representa uma “lição aprendida”.

Existem duas noções de antipadrões:

1. Aqueles que descrevem uma solução ruim para um problema que resultou em uma situação ruim;
2. Aqueles que descrevem como se livrar de uma situação ruim e como proceder dessa situação para uma situação boa.

Em suma, um antipadrão constitui o uso indevido dos padrões de projeto, ou o seu uso exagerado, o que pode ser constatado pela utilização de padrões impróprios para um determinado contexto, ou uso inadequado. A utilização dos padrões proporciona um aumento na flexibilidade do sistema, entretanto, pode deixá-lo mais complexo ou degradar a performance. Algumas perdas são toleráveis, mas subestimar os efeitos colaterais da adoção dos *patterns* é um erro comum, principalmente daqueles que tomam o uso como um diferencial e não pela real necessidade. (Ferreira, 2017).

5.2. Como selecionar um padrão de projeto

Escolher dentre os padrões existentes aquele que melhor soluciona um problema do projeto, sem cometer o erro de escolher de forma errônea e torná-lo inviável, é uma das tarefas mais difíceis. Em suma, a escolha de um padrão de projeto a ser utilizado pode ser baseada nos seguintes critérios:

1. Considerar como os padrões de projeto solucionam problemas de projeto.
2. Examinar qual a intenção do padrão, ou seja, o que faz de fato o padrão de projeto, quais seus princípios e que tópico ou problema particular de projeto ele trata (soluciona).
3. Estudar como os padrões se relacionam.

-
4. Estudar as semelhanças existentes entre os padrões.
 5. Examinar uma causa de reformulação de projeto.
 6. considerar o que deveria ser variável no seu projeto, ou seja, ao invés de considerar o que pode forçar uma mudança em um projeto, considerar o que você quer ser capaz de mudar sem reprojeta-lo. (Ferreira, 2017).

FINALIZANDO

Nesta aula, abordamos a reengenharia de *software* e os conceitos envolvidos nessa abordagem. Depois, estudamos modelos de reengenharia propostos por autores famosos. A reengenharia de *software* para arquiteturas cliente servidor e sistemas orientados a objetos também foi discutida. Por fim, vimos os conceitos de padrões de projeto, extremamente importantes para utilização como base no processo de reengenharia de sistemas legados.

REFERÊNCIAS

FERREIRA, A. **Conheça os padrões de projeto**. Disponível em: <<http://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>>. Acesso em: 9 out. 2017.

PRESSMAN, R. S. **Engenharia de software** – Uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. Tradução de Kalinka Oliveira e Ivan Bosnic. São Paulo: Pearson Prentice Hall, 2011.

SOURCEMAKING. **Design patterns**. Disponível em: <https://sourcemaking.com/design_patterns>. Acesso em: 9 out. 2017.