

Engenharia de Software

07 - Design e Implementação

Prof. Msc. Francisco de Assis de Souza Rodrigues

UniFAP
CENTRO UNIVERSITÁRIO PARAÍSO

"Instrui o sábio, e ele se tornará mais sábio; ensina o justo, e ele aumentará o seu conhecimento." **Provérbios 9:9**

Estudar é um processo contínuo. Quem aprende sempre cresce e se aprimora.

🎯 Objetivos

Gerais

- Compreender o papel do design e da implementação no processo de engenharia de software.
- Entender a importância da orientação a objetos e da UML no design de sistemas.
- Aplicar padrões de projeto (design patterns) no desenvolvimento de software real.

🎯 Objetivos

Específicos

- Descrever como o design orientado a objetos é representado com UML.
- Reconhecer os principais padrões de design e quando aplicá-los.
- Aplicar padrões de design (Factory, MVC, Singleton, etc.) no projeto prático da turma.
- Discutir implicações do uso de código aberto (open-source) na implementação.

UniFAP

Introdução ao Design e Implementação

- O papel do design na transformação dos requisitos em uma arquitetura de software.
- Diferença entre design de arquitetura e design detalhado.
- O impacto das decisões de design na manutenibilidade e qualidade do software.
- Relação entre design e implementação iterativa/incremental.

UniFAP

Introdução ao Design e Implementação

O papel do design na transformação dos requisitos em uma arquitetura de software

O design de software é a etapa que estabelece a ponte entre a especificação de requisitos e o código executável.

Enquanto os requisitos descrevem o que o sistema deve fazer, o design define como ele fará isso.

De acordo com Sommerville, o design é o processo de converter uma especificação de requisitos em uma descrição do sistema que possa ser implementada de forma eficiente.

Essa descrição define a estrutura geral, os componentes principais, suas interfaces e interações.

UniFAP

Introdução ao Design e Implementação

O papel do design na transformação dos requisitos em uma arquitetura de software

O design serve, portanto, como **plano técnico** do software — um modelo abstrato que guiará os programadores durante a implementação.

É nessa fase que se tomam as decisões fundamentais sobre **organização modular**, **representação de dados** e **fluxo de controle**.

O design é também a base para:

- **Dividir o sistema** em partes gerenciáveis.
- **Reduzir complexidade** através de abstrações.
- **Facilitar comunicação** entre membros da equipe de desenvolvimento.
- **Permitir reuso** e **evolução** futura do software.

Introdução ao Design e Implementação

Diferença entre design de arquitetura e design detalhado

Sommerville distingue duas camadas principais no processo de design:

a) Design de arquitetura

- Foca na **estrutura geral do sistema** e na **organização de seus componentes principais**.
- Define **como o sistema é decomposto** em módulos ou subsistemas, e **como esses módulos interagem**.
- Envolve decisões de alto nível, como o uso de um padrão arquitetural (por exemplo, cliente-servidor, camadas, pipeline, microserviços, etc.).
- Representa uma **visão macro** do software, útil para comunicação entre stakeholders e para garantir atributos de qualidade (desempenho, segurança, escalabilidade).

Exemplo (adaptado de Sommerville):
Em um sistema bancário, o design de arquitetura define que haverá módulos de **interface com o usuário**, **serviços de transação**, e **banco de dados**, especificando as conexões entre eles.

UniFAP

Introdução ao Design e Implementação

Diferença entre design de arquitetura e design detalhado

b) Design detalhado (ou de componentes)

- Especifica **como cada componente individual será implementado internamente**.
- Define as **classes, métodos, estruturas de dados, algoritmos** e **interfaces internas**.
- Representa uma **visão micro**, próxima da programação.
- Pode ser representado por **diagramas UML de classes e de sequência**, entre outros.

Assim, o **design arquitetural** trata da **estrutura global** do sistema, enquanto o **design detalhado** se concentra em **como os elementos internos** serão construídos.

Sommerville enfatiza que ambos estão interligados: decisões arquiteturais influenciam o design detalhado e vice-versa.

UniFAP

Introdução ao Design e Implementação

O impacto das decisões de design na manutenibilidade e qualidade do software

Sommerville destaca que **as decisões de design têm impacto direto na qualidade do sistema**, especialmente em atributos não funcionais como:

- **Manutenibilidade:** quanto mais bem estruturado e modular for o design, mais fácil será modificar, corrigir ou evoluir o sistema.
- **Reusabilidade:** componentes bem definidos e independentes podem ser reaproveitados em outros projetos.
- **Confiabilidade:** um design que separa responsabilidades reduz erros e facilita testes.
- **Desempenho:** certas decisões (como tipo de comunicação entre módulos) podem otimizar ou degradar o desempenho.
- **Segurança:** uma arquitetura mal definida pode introduzir vulnerabilidades estruturais.

UniFAP

Introdução ao Design e Implementação

O impacto das decisões de design na manutenibilidade e qualidade do software

Sommerville enfatiza que a **qualidade do design** depende de boas práticas, como:

- **Baixo acoplamento** (os módulos dependem pouco uns dos outros);
- **Alta coesão** (cada módulo tem uma responsabilidade clara e única);
- **Uso adequado de padrões de design** (para evitar reinvenção de soluções).

Essas propriedades tornam o sistema **mais sustentável** e **mais fácil de evoluir** ao longo do tempo.

UniFAP

Introdução ao Design e Implementação

Relação entre design e implementação iterativa/incremental

Segundo Sommerville, nas abordagens modernas (como o desenvolvimento ágil), **design e implementação não são etapas separadas**, mas **atividades que evoluem juntas**.

Em vez de projetar o sistema inteiro antes de programar (modelo em cascata), as equipes fazem **design incremental**:

- Em cada **iteração**, define-se o design apenas para as funcionalidades que serão implementadas naquele ciclo.
- Durante a **implementação**, o design pode ser **ajustado ou refinado**, conforme surgem novas descobertas ou restrições técnicas.
- O resultado é um **design evolutivo**, continuamente adaptado ao longo do projeto.

UniFAP

Introdução ao Design e Implementação

Relação entre design e implementação iterativa/incremental

Sommerville ressalta que esse processo requer **boas práticas de codificação** e **padrões de design consistentes**, para manter a coerência do sistema mesmo quando o design é modificado iterativamente.

> “O design e a implementação são interligados e, em muitos casos, são realizados de forma conjunta, especialmente em processos ágeis, onde o código é continuamente refinado e o design é constantemente revisto.”
> — *Sommerville, 10ª edição, Cap. 7, Seção 7.1*

UniFAP

Introdução ao Design e Implementação

Resumo

Aspecto	Descrição (conforme Sommerville)
Propósito do design	Traduzir requisitos em uma estrutura implementável
Design de arquitetura	Define estrutura e relações entre componentes
Design detalhado	Define lógica interna e estrutura de cada componente
Impacto das decisões	Influenciam manutenibilidade, desempenho e qualidade
Iteratividade	Design e implementação evoluem juntos nas práticas ágeis

UniFAP

Design Orientado a Objetos com UML
Introdução ao design orientado a objetos

Sommerville define o **design orientado a objetos (OO)** como uma abordagem de projeto que organiza um sistema como um **conjunto de objetos interativos**, onde cada objeto representa uma **entidade do mundo real ou um conceito do domínio**.

Cada objeto combina **dados (atributos)** e **operações (métodos)** que atuam sobre esses dados. Essa combinação reforça o princípio do **encapsulamento**, ou seja, os detalhes internos de um objeto são ocultados de outros objetos, reduzindo dependências e aumentando a modularidade.

> "O design orientado a objetos é um processo iterativo, no qual se desenvolve um modelo inicial do sistema e este é refinado progressivamente, adicionando-se detalhes até que possa ser implementado."
> — *Sommerville, 10ª ed., Cap. 7, p. 191 (edição em português)*

UniFAP

Design Orientado a Objetos com UML

Etapas principais do design orientado a objetos

O processo descrito por Sommerville segue uma **sequência iterativa de refinamento**, composta por quatro atividades principais:

1. Identificação de objetos

- O primeiro passo é identificar as **entidades do domínio do problema** que serão representadas como objetos no sistema.

- Essa identificação pode ser feita **a partir dos substantivos** nos requisitos ou nos casos de uso.

- Exemplo (sistema de biblioteca): *Livro*, *Usuário*, *Empréstimo*, *Bibliotecário*.

UniFAP

Design Orientado a Objetos com UML

Etapas principais do design orientado a objetos

2. Definição de classes

- Cada objeto é instanciado a partir de uma **classe**, que descreve os atributos e operações comuns.
- As classes representam **modelos genéricos** e os objetos são suas **instâncias específicas**.
- A definição das classes é acompanhada da **identificação de relacionamentos**, como:
  - **Associação** (ex.: um *Usuário* pode ter vários *Empréstimos*);
  - **Herança** (ex.: *Bibliotecário* é uma subclasse de *Usuário*);
  - **Agregação/Composição** (ex.: *Biblioteca* contém *Livros*).

UniFAP

Design Orientado a Objetos com UML

Etapas principais do design orientado a objetos

3. Especificação de interações

- Define **como os objetos colaboram** para realizar as funcionalidades do sistema.

- Essas interações são frequentemente representadas por **diagramas de sequência UML**, mostrando a **ordem de mensagens trocadas** entre objetos.

Design Orientado a Objetos com UML

Etapas principais do design orientado a objetos

4. Refinamento e design de subsistemas

- À medida que o projeto evolui, as classes e interações são refinadas, e o sistema é organizado em **subsistemas** ou **pacotes** lógicos.

- Essa organização permite **design modular** e **reuso de componentes**.

Design Orientado a Objetos com UML

Modelagem com UML

A **UML (Unified Modeling Language)** é a principal ferramenta para representar o design orientado a objetos de forma padronizada.

Sommerville destaca que a UML **não é uma metodologia**, mas sim uma **notação gráfica** que pode ser usada com qualquer processo de desenvolvimento.

UniFAP

Design Orientado a Objetos com UML
Modelagem com UML

As **principais visões e diagramas UML** utilizadas no design são:

a) Diagrama de Classes

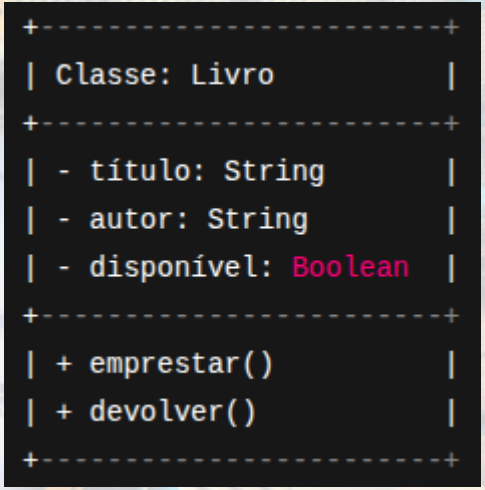
- Representa a **estrutura estática** do sistema.
- Mostra as **classes, atributos, operações** e os **relacionamentos** entre elas (associação, herança, dependência).
- É o diagrama central do design OO, pois define o “esqueleto” do sistema.

UniFAP

Design Orientado a Objetos com UML

Modelagem com UML

Elementos típicos (segundo Sommerville):



Exemplo de relacionamento:



UniFAP

Design Orientado a Objetos com UML

Modelagem com UML

b) Diagrama de Sequência

- Representa a **dinâmica do sistema**: a forma como os objetos interagem ao longo do tempo.
- Mostra **mensagens enviadas** entre objetos e **ordem temporal** dessas interações.
- É usado para refinar os casos de uso e entender o comportamento do sistema.

Exemplo simplificado (sistema de biblioteca):

```
Usuário → Sistema: solicitarEmpréstimo()
Sistema → Livro: verificarDisponibilidade()
Livro → Sistema: status = disponível
Sistema → Livro: emprestar()
```

UniFAP

Design Orientado a Objetos com UML

Modelagem com UML

c) Diagrama de Estados (ou de Máquina de Estados)

- Mostra **como um objeto muda de estado** em resposta a eventos.

- Útil para classes que têm **ciclo de vida definido** (ex.: *Pedido*, *Sessão*, *Empréstimo*).

Exemplo:

```
[Disponível] → (emprestar) → [Emprestado]
[Emprestado] → (devolver) → [Disponível]
```

UniFAP

Design Orientado a Objetos com UML

Benefícios do design orientado a objetos

Sommerville destaca que o design OO traz diversos benefícios em relação aos métodos estruturados tradicionais:

Benefício	Explicação
Reusabilidade	Classes e objetos podem ser reutilizados em outros sistemas.
Facilidade de manutenção	Mudanças em uma classe tendem a afetar apenas suas instâncias.
Naturalidade	O modelo OO reflete conceitos do mundo real, facilitando o entendimento.
Modularidade	O encapsulamento permite desenvolvimento paralelo por diferentes equipes.
Extensibilidade	Novas classes podem ser adicionadas por herança ou composição.

UniFAP

Design Orientado a Objetos com UML

Benefícios do design orientado a objetos

Exemplo prático

Sommerville apresenta um **exemplo de design orientado a objetos** usando um **sistema de reserva de viagens (ou biblioteca, conforme edição)**.

Esse exemplo ilustra como a UML fornece uma **visão clara e comunicável do design**, que serve de base direta para a implementação.

Design Orientado a Objetos com UML

Benefícios do design orientado a objetos

O processo segue estas etapas:

1. **Identificação de classes:**
   - *Cliente*, *Reserva*, *Voo*, *Pagamento*.
2. **Relacionamentos:**
   - *Cliente* faz *Reserva*, *Reserva* está associada a *Voo*.
3. **Operações principais:**
   - fazerReserva(), cancelarReserva(), emitirBilhete().
4. **Representação UML:**
   - Diagramas de classes mostram associações, multiplicidade e operações.
5. **Refinamento:**
   - O diagrama é detalhado com atributos e métodos específicos.

UniFAP

Design Orientado a Objetos com UML

Práticas recomendadas no design OO

- **Usar abstração:** representar apenas o que é relevante para o problema.
- **Encapsular dados:** não expor detalhes internos das classes.
- **Favorecer composição sobre herança:** quando possível, construir objetos a partir de outros, em vez de herdar comportamento.
- **Minimizar acoplamento:** reduzir dependências diretas entre classes.
- **Maximizar coesão:** garantir que cada classe tenha uma responsabilidade única.
- **Iterar o design:** revisar continuamente os diagramas à medida que o código evolui.

Design Orientado a Objetos com UML

Resumo

Conceito	Descrição conforme Sommerville
Design OO	Organiza o sistema como um conjunto de objetos interativos
Classe	Modelo genérico que define estrutura e comportamento
Objeto	Instância concreta de uma classe
Principais diagramas UML	Classes, Sequência, Estados
Benefícios	Reuso, manutenção, modularidade, naturalidade
Princípios	Abstração, encapsulamento, coesão e baixo acoplamento

UniFAP

Padrões de Design

O que são padrões de design

Sommerville define **padrões de design (design patterns)** como **soluções reutilizáveis para problemas recorrentes de design de software**.
Eles representam **boas práticas**, desenvolvidas a partir da **experiência acumulada de projetistas**, e podem ser aplicados em diferentes contextos para **melhorar a estrutura e a qualidade** do sistema.

> “Um padrão de design é uma descrição abstrata de uma solução para um problema recorrente de design. Ele fornece um modelo genérico que pode ser adaptado a situações específicas.”
> — *Sommerville, 10ª ed., Cap. 7, Seção 7.4*

Os padrões **não são fragmentos de código prontos**, mas sim **modelos conceituais** que orientam a tomada de decisão durante o design detalhado.

UniFAP

Padrões de Design

Estrutura de um padrão de design

De acordo com Sommerville, cada padrão é normalmente descrito por uma estrutura composta por quatro elementos principais:

Elemento	Descrição
Nome	Identifica o padrão e facilita sua comunicação entre desenvolvedores.
Problema	Descreve a situação recorrente que o padrão resolve.
Solução	Apresenta a estrutura geral e as relações entre classes e objetos para resolver o problema.
Consequências	Explica as vantagens e desvantagens de aplicar o padrão.

Essa estrutura padronizada permite que os engenheiros comuniquem **ideias complexas de design** de forma **simples e padronizada**, reduzindo ambiguidades.

UniFAP

Padrões de Design

Estrutura de um padrão de design

Essa estrutura padronizada permite que os engenheiros comuniquem **ideias complexas de design** de forma **simples e padronizada**, reduzindo ambiguidades.

Padrões de Design

Classificação dos padrões de design

Sommerville adota a **classificação proposta pelo grupo “Gang of Four (GoF)”**, que divide os padrões em três categorias principais:

Categoria	Foco	Exemplo
Padrões de criação (Creational)	Tratam da forma como objetos são criados, abstraindo o processo de instanciação.	*Factory Method*, *Singleton*, *Abstract Factory*
Padrões estruturais (Structural)	Definem como classes e objetos são compostos para formar estruturas maiores.	*Adapter*, *Decorator*, *Facade*
Padrões comportamentais (Behavioral)	Descrevem como objetos interagem e se comunicam.	*Observer*, *Strategy*, *Command*

UniFAP

Padrões de Design

O uso adequado desses padrões **aumenta a flexibilidade**, **reduz o acoplamento** e **facilita a manutenção** do software.

Padrões de Design

Padrões de design apresentados por Sommerville

O autor destaca alguns dos padrões mais conhecidos e aplicáveis no contexto de sistemas orientados a objetos.

A seguir estão os principais, conforme discutidos no livro.

Padrões de Design

🏭 1. Factory Method (Padrão de Criação)

Problema:

Em muitos sistemas, é necessário criar objetos sem acoplar o código a classes concretas específicas.

Solução:

O padrão *Factory Method* define uma **operação (método de fábrica)** que cria objetos, mas permite que subclasses decidam **qual classe concreta instanciar**.

Isso **isola o código cliente** dos detalhes de criação.

UniFAP

Padrões de Design

🏭 1. Factory Method (Padrão de Criação)

Exemplo:

Em um sistema de interface gráfica, uma classe abstrata `Janela` pode ter um método `criarBotao()`.

Cada subclasse (`JanelaWindows`, `JanelaLinux`) implementa o método criando um tipo específico de botão.

Assim, o sistema é independente da plataforma.

Benefícios:

- Facilita a **extensão** (novos produtos podem ser adicionados sem alterar código existente).
- Reduz o **acoplamento** entre código cliente e classes concretas.

UniFAP

Padrões de Design

Model-View-Controller (MVC) (Padrão Arquitetural / Estrutural)

Problema:

Em sistemas interativos, é necessário **separar a lógica de negócio da interface com o usuário**, para facilitar manutenção e reuso.

Solução:

O padrão **Model-View-Controller (MVC)** divide o sistema em três componentes principais:

Componente	Responsabilidade
Model	Representa os dados e a lógica de negócio.
View	Exibe as informações ao usuário.
Controller	Interpreta as ações do usuário e atualiza o modelo ou a visão.

UniFAP

Padrões de Design

Model-View-Controller (MVC) (Padrão Arquitetural / Estrutural)

Fluxo típico:

1. O usuário interage com a **View**.
2. A **Controller** interpreta o evento e faz alterações no **Model**.
3. O **Model** notifica a **View**, que atualiza a interface.

Benefícios:

- Facilita a **manutenção e teste**, pois cada camada é independente.
- Permite **múltiplas visualizações** para o mesmo modelo.
- Amplamente usado em frameworks modernos (ex.: Django, Laravel, React com arquitetura Flux).

UniFAP

Padrões de Design

👁 3. Observer (Padrão Comportamental)

Problema:

É necessário manter **múltiplos objetos sincronizados** com as mudanças de outro objeto, sem criar dependências rígidas entre eles.

Solução:

O padrão *Observer* define uma relação “um-para-muitos” entre objetos, onde **quando o estado de um objeto muda**, todos os seus **observadores** são **notificados automaticamente**.

UniFAP

Padrões de Design

👁️ 3. Observer (Padrão Comportamental)

Exemplo:

Em um sistema de monitoramento, o objeto Sensor (Subject) notifica automaticamente todos os Painéis de exibição (Observers) sempre que o valor medido muda.

Benefícios:

Favorece baixo acoplamento entre os componentes.

Facilita extensão (novos observadores podem ser adicionados sem alterar o sujeito).

UniFAP

Padrões de Design

🔒 4. Singleton (Padrão de Criação)

Problema:

Em certos contextos, é necessário garantir que apenas uma instância de uma classe exista durante a execução (ex.: gerenciador de conexão, logger, cache global).

Solução:

O padrão Singleton restringe a criação de objetos, garantindo que somente uma instância exista e fornecendo um ponto global de acesso a ela.

UniFAP

Padrões de Design

Benefícios:

- Controle centralizado de recursos compartilhados.
- Reduz inconsistências.

Cuidados:

Sommerville observa que o Singleton **deve ser usado com moderação**, pois pode introduzir **dependências globais** que dificultam testes e manutenção.

UniFAP

Padrões de Design

Vantagens do uso de padrões de design

Sommerville destaca que os padrões de design:

- **Promovem reuso de soluções testadas e comprovadas.**
- **Reduzem erros de design**, pois encapsulam boas práticas.
- **Facilitam a comunicação** entre desenvolvedores por meio de uma linguagem comum.
- **Aceleram o desenvolvimento**, já que evitam “reinventar a roda”.
- **Aumentam a manutenibilidade**, pois promovem consistência estrutural entre diferentes partes do sistema.

> “Os padrões de design fornecem um vocabulário comum para os engenheiros de software e permitem que os problemas de design sejam discutidos em um nível mais alto de abstração.”
> *— Sommerville, 10ª ed., Cap. 7*

UniFAP

Padrões de Design

Padrões e frameworks open-source

O autor também observa que frameworks de código aberto aplicam extensivamente padrões de design:

- Frameworks web como Spring (Java) e Django (Python) implementam MVC.
- Frameworks gráficos e de interface usam Factory e Observer para criação e atualização de componentes.
- O uso de padrões aumenta a previsibilidade e facilita o entendimento de grandes bases de código open-source.

Assim, estudar padrões de design ajuda o aluno a compreender e contribuir melhor com projetos reais que seguem essas convenções.

UniFAP

Padrões de Design
Resumo

Conceito	Definição conforme Sommerville
Padrão de design	Solução reutilizável para um problema recorrente de design
Elementos	Nome, problema, solução, consequências
Categorias	Criação, Estrutura, Comportamento
Exemplos	Factory, MVC, Observer, Singleton
Benefícios	Reuso, manutenção, comunicação, consistência

UniFAP

Atividade

1. Identificar um problema recorrente de design (ex.: criação de objetos, atualização de interface, controle de dependências).

2. Escolher um padrão entre os apresentados (Factory, MVC, Observer ou Singleton).

3. Representar a solução com **diagrama UML de classes** e **trecho de código**.

4. Explicar como a aplicação do padrão melhorou a **organização, reuso ou manutenção** do projeto.

Questões de Implementação

Padrões e frameworks open-source

O autor também observa que frameworks de código aberto aplicam extensivamente padrões de design:

- Frameworks web como Spring (Java) e Django (Python) implementam MVC.
- Frameworks gráficos e de interface usam Factory e Observer para criação e atualização de componentes.
- O uso de padrões aumenta a previsibilidade e facilita o entendimento de grandes bases de código open-source.

Assim, estudar padrões de design ajuda a compreender e contribuir melhor com projetos reais que seguem essas convenções.

UniFAP

Questões de Implementação
Introdução

Após definir o design do sistema, a **implementação** consiste em **traduzir esse design em código executável**.

Sommerville enfatiza que a implementação não é apenas “digitar código”: ela envolve decisões técnicas que impactam **qualidade, desempenho e manutenção** do software.

A seção aborda:

- Decisões de linguagem e estrutura de código.
- Boas práticas de programação.
- Modularidade e reutilização.
- Uso de software open-source.

UniFAP

Questões de Implementação
Decisões de implementação

Sommerville identifica **decisões críticas** que o desenvolvedor deve tomar ao implementar um sistema:

a) Escolha da linguagem de programação

- Deve refletir os **requisitos do projeto** e o **design definido**.
- Linguagens orientadas a objetos (Java, C++, Python) suportam diretamente **design OO e padrões de design**.
- Linguagens funcionais ou procedurais podem exigir adaptações.

Exemplo: Se o design utiliza MVC e Factory, uma linguagem OO como Java facilita a implementação direta de classes, interfaces e herança.

UniFAP

Questões de Implementação

Decisões de implementação

b) Estrutura do código

- Organização de **pacotes, módulos e diretórios** de acordo com o design.
- Convenção de nomes consistente e legível.
- Comentários claros para explicar decisões complexas.

UniFAP

Questões de Implementação
Decisões de implementação

c) Reuso de componentes

- Sempre que possível, utilizar **bibliotecas e frameworks existentes** para funcionalidades comuns (ex.: autenticação, logging, acesso a banco).

- Evitar “reinventar a roda” para acelerar o desenvolvimento e reduzir bugs.

Questões de Implementação

Boas práticas de programação

Sommerville recomenda seguir **princípios que refletem o design** durante a codificação:

Princípio	Descrição
Encapsulamento	Cada módulo deve esconder detalhes internos e expor apenas interfaces claras.
Coesão	Cada módulo ou classe deve ter uma responsabilidade única.
Baixo acoplamento	Minimizar dependências diretas entre módulos; mudanças em um módulo não devem afetar os outros.
Consistência	Seguir padrões de codificação, nomes e estruturas em todo o projeto.
Documentação mínima necessária	Código autoexplicativo + comentários em trechos complexos.

UniFAP

Questões de Implementação
Modularidade e reutilização

A modularidade é **uma consequência direta do design orientado a objetos e da aplicação de padrões**.

- Um sistema modular é dividido em **módulos independentes** que podem ser **desenvolvidos, testados e substituídos separadamente**.
- Módulos bem projetados promovem **reuso**, pois podem ser utilizados em outros projetos sem grandes alterações.
- A modularidade também facilita **teste unitário**, já que cada módulo pode ser avaliado isoladamente.

Exemplo: Uma classe Logger implementada como Singleton pode ser usada em qualquer módulo do sistema sem necessidade de duplicação de código.

UniFAP

Questões de Implementação

Uso de software open-source

Sommerville destaca que **o software open-source oferece recursos valiosos** durante a implementação:

- **Bibliotecas e frameworks**: aceleram o desenvolvimento e reduzem erros (ex.: React, Django, Spring).
- **Exemplos de código real**: ajudam na aprendizagem de padrões e boas práticas.
- **Colaboração e comunidade**: contribuições externas podem corrigir falhas ou adicionar funcionalidades.

Questões de Implementação

Uso de software open-source

Cuidados ao usar open-source:

- Verificar **licença** e conformidade legal.
- Garantir **compatibilidade com o design do sistema**.
- Avaliar **qualidade e manutenção** da biblioteca ou framework.

Questões de Implementação
Iteratividade entre design e implementação

Sommerville reforça que **design e implementação não são atividades isoladas**:

- Durante a implementação, problemas podem surgir que exigem **refinar o design**.
- Padrões de design e modularidade facilitam **alterações incrementais**, mantendo consistência.
- Em metodologias ágeis, cada iteração implementa parte do design, permitindo **feedback rápido** e ajustes contínuos.

> “Implementação e design devem evoluir juntos. A implementação fornece feedback sobre decisões de design, e o design orienta a implementação para manter a qualidade.”
> *— Sommerville, 10ª ed., Cap. 7, Seção 7.5*

UniFAP

Questões de Implementação

Resumo

Tópico	Observações principais (Sommerville)
Escolha da linguagem	Deve suportar design OO e padrões de design
Estrutura do código	Pacotes, diretórios, consistência de nomes e comentários
Boas práticas	Encapsulamento, coesão, baixo acoplamento, consistência
Modularidade	Facilita reuso, manutenção e testes unitários
Open-source	Permite acelerar desenvolvimento, mas exige atenção a licença e qualidade
Design x Implementação	Evoluem de forma iterativa; feedback contínuo melhora qualidade

UniFAP

Aplicando Padrões de Design no Projeto - Atividade

Objetivo: incorporar um ou mais padrões de design ao seu projeto.

Etapas:

Análise do projeto existente: identificar partes do sistema que repetem código, acoplamento alto ou dependências diretas.

Escolha de um padrão:

- *Factory Method* → para criação de objetos sem expor lógica de instância.
- *MVC* → para organizar camadas de aplicação web.
- *Singleton* → para gerenciar recursos únicos (banco de dados, logger).

UniFAP

Aplicando Padrões de Design no Projeto - Atividade

Implementação:

- Criar/ajustar as classes conforme o padrão escolhido.
- Justificar a escolha em um comentário no código ou README.

Diagrama UML: desenhar (no papel, Figma ou draw.io) o diagrama de classes antes e depois da aplicação do padrão.

Apresentação rápida: cada grupo explica em 3 minutos o padrão aplicado e o ganho obtido.

UniFAP

Aplicando Padrões de Design no Projeto - Atividade

- Ler as seções 7.3 e 7.4. do livro *Engenharia de Software* (Sommerville, 10ª edição) — Capítulo 7.

- Escolher um padrão **não aplicado na aula** (por exemplo, *Observer* ou *Decorator*) e sugerir onde ele poderia ser usado seu projeto.

- Entregar um pequeno resumo (meia página) no AVA.

UniFAP

Recursos

- **Livro:** *Engenharia de Software* (Sommerville, 10ª edição) — Capítulo 7.
- Slides de apoio (baseados nas figuras e tabelas do capítulo).
- **Ferramentas sugeridas:** Visual Paradigm, StarUML, Figma, Draw.io.
- **Editor de código:** VSCode, outros.

UniFAP

Sugestões

Site de ensino de padrão: https://refactoring.guru/



UniFAP

Bibliografia

SOMMERVILLE, Ian. Engenharia de software. 10. ed. São Paulo, SP: Pearson, 2018. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 15 ago 2025.

https://plataforma.bvirtual.com.br/Acervo/Publicacao/168127



Referências

CHATGPT. Expansão conceitual sobre integração entre COBIT, ITIL e PMBOK. OpenAI, 2025. Disponível em: https://chat.openai.com/. Acesso em: 11 ago. 2025.


UniFAP


OBRIGADO
THANK YOU
Merci
Asante
Grazie
Kiitos
Spasibo
Arigato
Obrigado
Mochchakkeram
Terma Kasih
Chokrane
Dankon
Maake
Multumesc
Raibh Maith Agat
Ua Tsaug Rau Koj
Kia Ora
Mahalo
Welalin