OTES12 – Tópicos Avançados em Engenharia de Software

Universidade do Estado de Santa Catarina
Centro de Ciências Tecnológicas – DCC

Prof. Dr. William Alberto Cruz Castañeda

2020/2

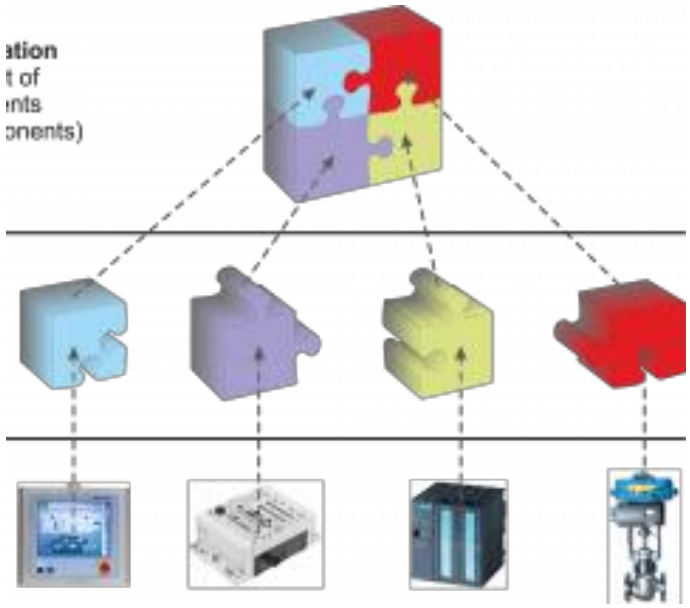
[Reusable Software Assets]

O que é um ativo de software?

Ativos reutilizáveis são compostos por uma coleção de produtos de software relacionados que podem ser reutilizados de um aplicativo para outro.

Componente → é usada com um significado específico:

- Ativo executável que pode ser integrado em um aplicativo;



Granularidade

- Função ou um procedimento;
- Classe;
- Grupo de classes;
- Subsistema, framework ou serviço;
- Aplicativo ou produto;

Estratégias técnicas

- Componentes de software podem ter formas diferentes, determinando a maneira como é implementado, usado e reutilizado.

Type of asset	Description	Type of usage
Executable component	It may execute separately. Applications reuse it by invoking it or calling its services at run-time.	Invoke
Library	A set of classes or procedures that may be integrated to and called from several applications.	Integrate
Pattern	They consist in identified modelling solutions that may be applied to several problems when modelling different systems.	Imitate
Frameworks	Applications are built customizing and adapting the framework in accordance to its architectural style and rules.	Customize

Domain models	They constitute the baseline on which any application (inside this domain) model is built.	Build from
Software packages	The simplest way to reuse is to integrate existing design and code work products.	Integrate
Application generator	It makes it possible to build a new application, or a part of it, from its specifications.	Generate
Requirement models	The requirement model, on which a product has been built, may be kept and compared to new customers' requirements in order to evaluate if the product can fulfil their need.	Compare
Product baseline	It consists of all stable software work-products that are contained in different versions of a product. Any new product is built from all or some of these software pieces.	Extract

Quais são as características dos componentes reutilizáveis?

Obviamente, um componente não é muito valioso se não se destina a ser reutilizado. Mas nem todo software é reutilizável.

1. **Critérios Gerais;**
2. **Critérios Funcionais;**
3. **Critérios Técnicos;**

Critérios Gerais: Qualidade e Reutilização

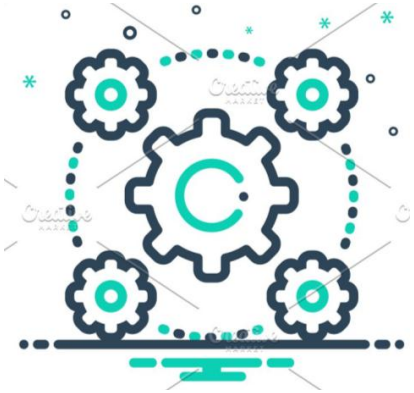
- Padrões, diretrizes e regras relativas ao design, codificação e documentação;
- Processo de engenharia e melhores praticas;
- Cumprir com artefatos e informações fornecidas em conformidade com um modelo;
- Simplicidade e compreensibilidade;
- Modularidade;

Critérios Funcionais

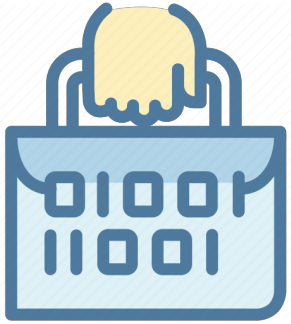
- Automatizar ou simular, total ou parcialmente, um processo de negócios;
- Fornece serviços e é capaz de priorizar solicitações, agendar e registrar sua execução;
- Permanecer disponível para seus clientes e deve processar atividades paralelas;
- Delegar serviços para componentes aninhados ou para outros ativos;
- Obedecer padrões em um domínio especifico;
- Compromisso entre ser muito específico (e menos reutilizável) e ser muito genérico (e menos valioso) → personalizável;

Critérios Técnicos

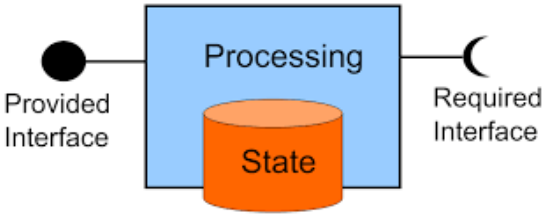
- **Interoperabilidade**: capacidade de se comunicar facilmente com outros componentes e, assim, serem integrados a um aplicativo.

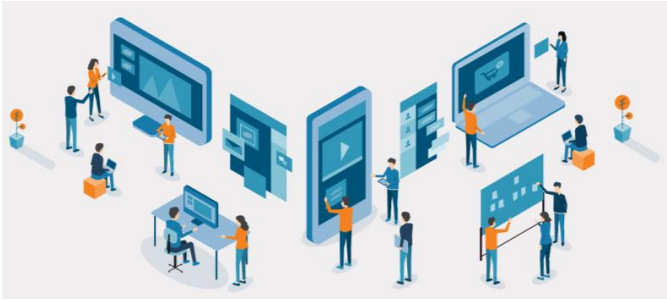


- **Portabilidade**: capacidade de executar em diferentes plataformas.

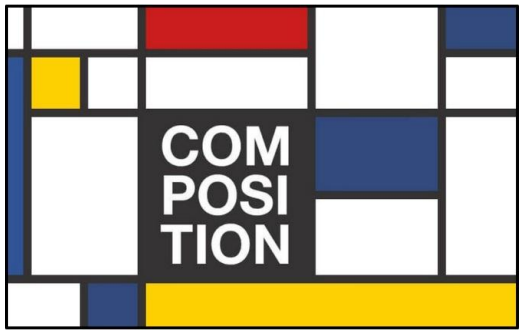


- **Diferenciação entre interface e implementação**: como um componente é usado (ou reutilizado) e como ele é implementado são duas coisas diferentes que não devem ser misturadas, pois podem evoluir separadamente.





- **Composição**: os componentes podem ser agrupados para formar ativos de nível superior e podem ser decompostos em ativos refinados.



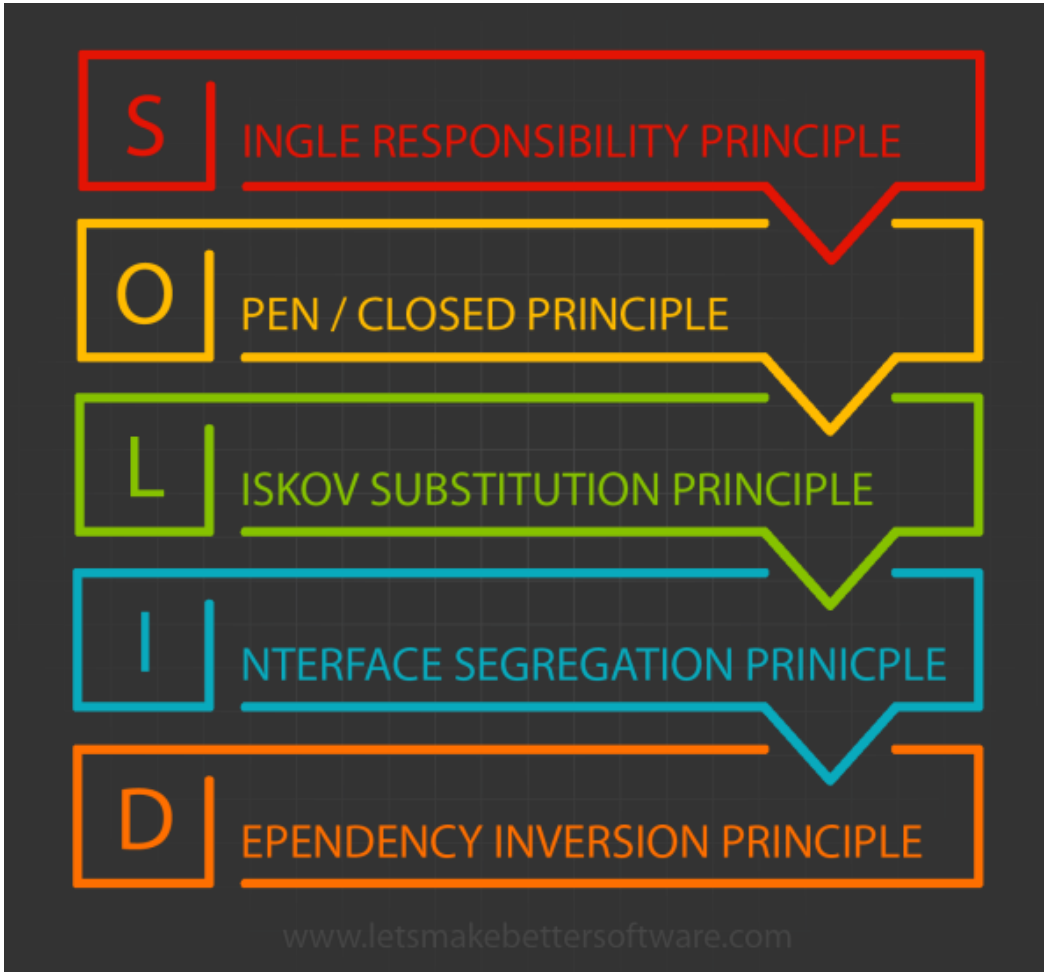
- **Auto-descrição**: um componente executável, manipulado como uma caixa preta, deve descrever sua própria interface, fornecer seu protocolo de uso. Assim o usuário poderá entender sua interface e propriedades.

- **Transparência de localização**: os componentes executáveis podem ser executados em máquinas diferentes, sem afetar o restante do sistema.

- **Segurança**: como o componente contém código executável e é reutilizado como uma caixa preta entre uma comunidade de desenvolvedores de software, quem reutilizar deve poder controlar a origem e o acesso do recurso a recursos privados.

- ***Plug and play***: a portabilidade do componente é necessária não no tempo de construção, mas no tempo de execução.

[Princípios Básicos]


S
INGLE RESPONSIBILITY PRINCIPLE
O
PEN / CLOSED PRINCIPLE
L
ISKOV SUBSTITUTION PRINCIPLE
I
NTERFACE SEGREGATION PRINICPLE
D
EPENDENCY INVERSION PRINCIPLE
www.letsmakebettersoftware.com

Single Responsibility Principle

Perspectiva diferente sobre um dos princípios mais fundamentais do design de software orientado a objetos: **coesão**.

Uma classe deve ter apenas um motivo para mudar

Uma classe com mais de uma razão para mudar tem mais de uma responsabilidade, ou seja, não é coesa. Isso introduz vários problemas:

- Difícil de entender e, portanto, difícil de manter;
- Não pode ser facilmente reutilizado;
- Dificulta mudança de responsabilidades (rigidez e fragilidade);
- Classe com número excessivo de dependências e sujeita a mudanças em outras classes;

Open-Close Principle (OCP)

Um componente deve ser aberto para extensão mas fechado para modificação.

Representa uma caracteristica de um bom nivel de design de componentes

The Liskov Substitution Principle (LSP)

Sugere que um componente que usa uma classe base continue a funcionar corretamente se uma classe derivada da classe base for passada para o componente.

- Extensão do open/close principle;
- Garante que novas classes derivadas estendam as classes base sem alterar seu comportamento;

Interface Segregation Principle

- Aborda a coesão de interfaces;
- Não forçar a confiar em métodos que não usam
- Para ter uma classe coesa e reutilizável, devemos dar uma única responsabilidade.
- Responsabilidade pode ser dividida em responsabilidades menores.

```
public interface IScrumTeamMember
{
    void PrioritizeBacklog();
    void ShieldTeam();
    void DevelopFeatures();
}
```

```
public class Developer : IScrumTeamMember
{
    public void PrioritizeBacklog() {}
    public void ShieldTeam() {}
    public void DevelopFeatures()
    {
        Console.Writeline("Developing new features");
    }
}

public class ScrumMaster : IScrumTeamMember
{
    public void PrioritizeBacklog() {}
    public void ShieldTeam()
    {
        Console.Writeline("Shield Scrum Development Team");
    }
    public void DevelopFeatures() {}
}

public class ProductOwner : IScrumTeamMember
{
    public void PrioritizeBacklog()
    {
        Console.Writeline("Prioritize backlog items");
    }
    public void ShieldTeam() {}
    public void DevelopFeatures() {}
}
```

```
public interface IScrumMasterFunction
{
    void ShieldTeam();
}

public class ScrumMaster : IScrumMasterFunction
{
    public void ShieldTeam()
    {
        Console.Writeline("Shield the Scrum Development Team");
    }
}
```

Dependency Inversion Principle (DIP)

Dependa de abstrações. Não dependa de detalhes

```
public class Button
{
    private Lamp _lamp;
    public void TurnOn()
    {
        if (condition)
            _lamp.Light();
    }
}
```

Pilar para uma boa arquitetura de software

```
public class Button
{
    private IDevice _device;
    public void TurnOn()
    {
        if (condition)
            _device.TurnOn();
    }
}

public interface IDevice
{
    void TurnOn();
    void TurnOff();
}

public class Lamp : IDevice
{
    public void TurnOn()
    {
    }
    public void TurnOff()
    {
    }
}
```

Não considere os componentes como partes de software enraizadas que não devem evoluir.

Acompanhar os históricos de uso dos componentes:

- Forma de facilitar medições;
- Dar confiança a potenciais reutilizadores;
- Notificar os usuários de um componente quando uma nova versão é lançada;

Organização da aquisição de componentes

Uma empresa ou projeto deve considerar várias opções se for necessário a utilização ode um componente que ainda não exista.

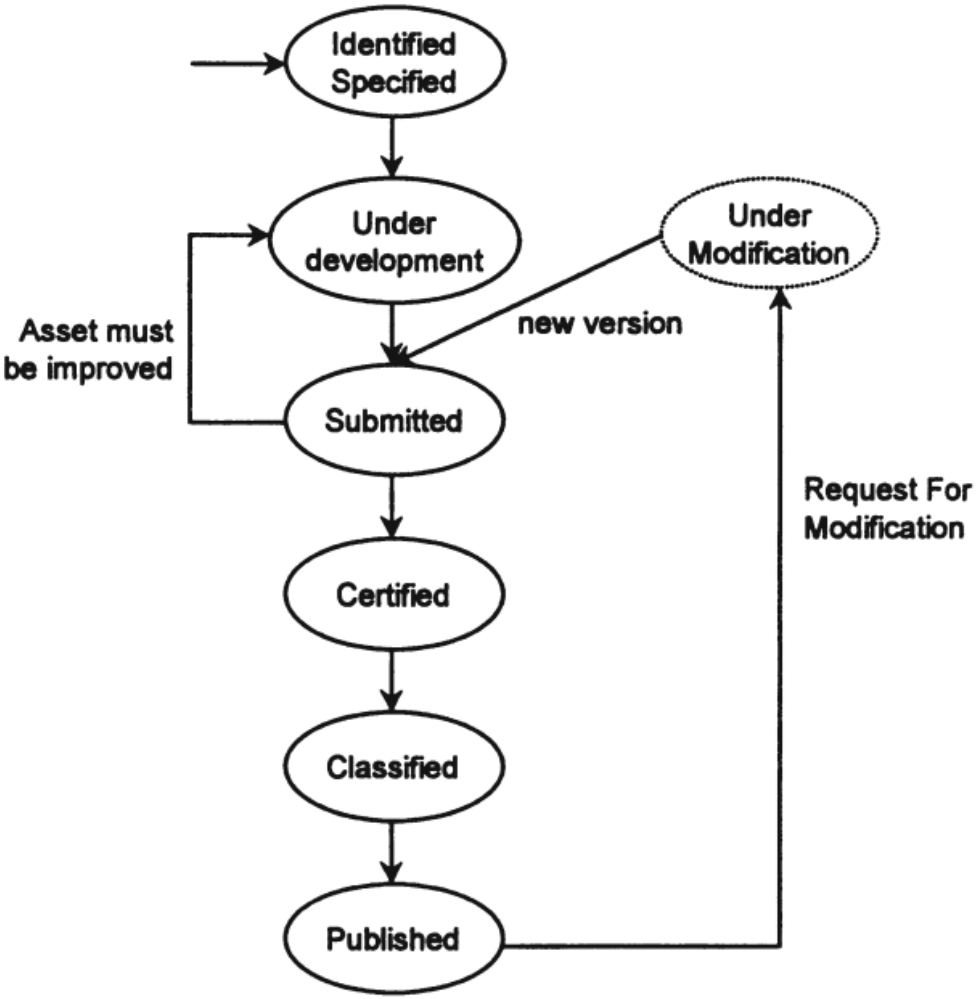
- Comprar, se existir, um produto comercial pronto para uso (COTS);
- Componente gratuito na internet ou em arquivos públicos;
- Desenvolver;
- Terceirizar o desenvolvimento;

[Gerenciando componentes de software]

Ciclo de vida do componente

Dois eixos principais de variabilidade:

- Ao longo do tempo: componente mantido, corrigido e aprimorado para corresponder aos requisitos em evolução;
- Entre aplicativos: o componente pode ser adaptado de acordo com os diferentes aplicativos que o utilizam.



[Repositório de Reuso]

O repositório é o local em que componentes de software reutilizáveis são armazenados, juntamente com o catálogo de componentes.

Vantagens:

- Definição e reconhecimento comum de um local para componentes;
- Documentar, pesquisar e contabilizar componentes;
- Gerenciar alterações e aprimoramentos de componentes;

OTES12 – Tópicos Avançados em Engenharia de Software

Universidade do Estado de Santa Catarina
Centro de Ciências Tecnológicas – DCC

Prof. Dr. William Alberto Cruz Castañeda

2020/2