

Faculdade de Ciências e Tecnologia

Departamento de Matemática e Computação

Bacharelado em Ciência da Computação

unesp

UNIVERSIDADE ESTADUAL PAULISTA

"JÚLIO DE MESQUITA FILHO"

Engenharia de Software II

Aula 01 - Revisão

Prof. Dr. Rogério Eduardo Garcia

(rogerio.garcia@unesp.br)

1

unesp

Revisão

31/07/2025


Prof. Dr. Rogério Eduardo Garcia

Engenharia de Software I

Método Larman

2

Revisão

Engenharia de Software

31/07/2025

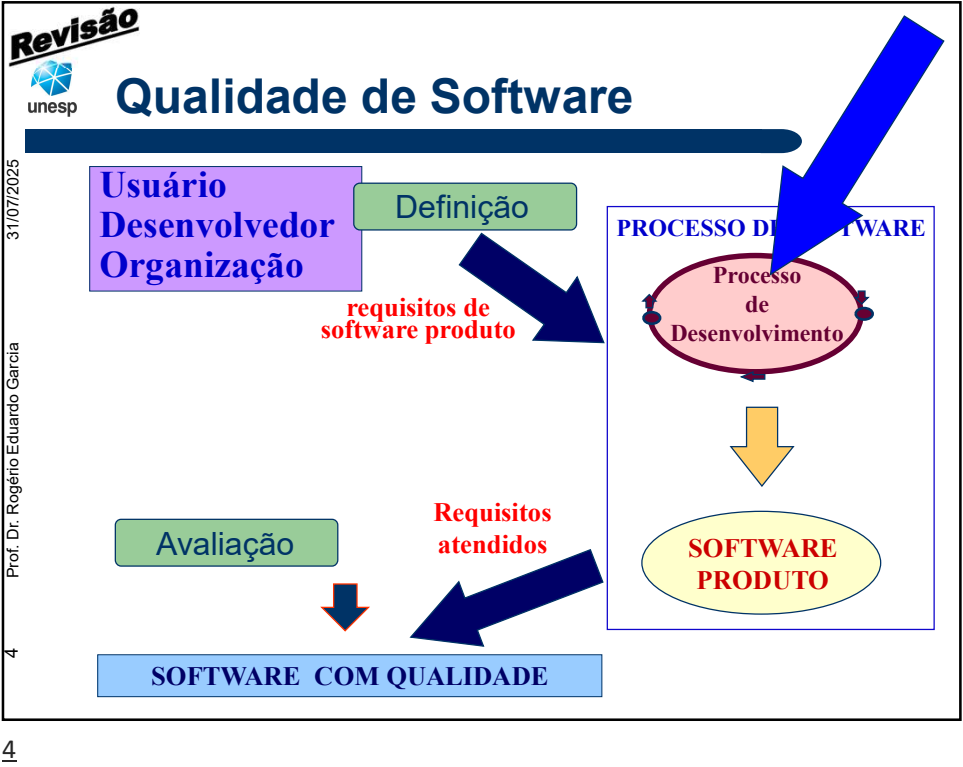
Prof. Dr. Rogério Eduardo Garcia

3

PROCESSO DE SOFTWARE

A aplicação de uma **abordagem** sistemática, disciplinada e possível de ser medida para o desenvolvimento, operação e manutenção do software (*IEEE*)

3



Revisão

unesp

Processo de Software

Abrange um conjunto de três elementos fundamentais: **Métodos**, **Ferramentas** e **Procedimentos** para projetar, construir e manter grandes sistemas de software de forma profissional

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
5

5

Revisão

unesp

O Processo de Software


- **MÉTODOS:** proporcionam os detalhes de como fazer para construir o software

- Planejamento e estimativa de projeto
- Análise de requisitos de software e de sistemas
- Projeto da estrutura de dados
- Algoritmo de processamento
- Codificação
- Teste
- Manutenção

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
6

6

Revisão

 **O Processo de Software**


31/07/2025
Prof. Dr. Rogério Eduardo Garcia

- **FERRAMENTAS:** dão suporte automatizado aos métodos.
 - Existem atualmente ferramentas para sustentar cada um dos métodos
 - Quando as ferramentas são integradas, é estabelecido um sistema de suporte ao desenvolvimento de software chamado *CASE - Computer Aided Software Engineering*

7

Z

Revisão

 **O Processo de Software**


31/07/2025
Prof. Dr. Rogério Eduardo Garcia

- **PROCEDIMENTOS:** constituem o elo de ligação entre os métodos e ferramentas
 - Sequência em que os métodos devem ser aplicados
 - Produtos que se exige que sejam entregues
 - Controles que ajudam assegurar a qualidade e coordenar as alterações
 - Marcos de referência que possibilitam administrar o progresso do software.

8

|∞

Revisão

 unesp


Fases Genéricas

- Independentemente da natureza do projeto e aplicação os modelos de processo de software possuem:
 - fase de definição
 - fase de desenvolvimento
 - fase de manutenção
 - atividades de apoio

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
9

10

Revisão

 unesp

Fase de Definição

focaliza "o que" será desenvolvido

- que informação vai ser processada e que processamento é esse
- que função e que desempenho são desejados
- que comportamento é esperado do sistema
- que interfaces devem ser estabelecidas
- que restrições de projeto existem
- que critérios de validação são exigidos para definir um sistema bem sucedido
- que tarefas devem ser realizadas

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
10

10

Revisão

unesp

Fase de Definição

focaliza "o *que*" será desenvolvido

- que informação vai ser processada
- que função e desempenho são desejados
- que comportamento pode ser esperado do sistema

três tarefas principais ocorrem de alguma forma:

engenharia de sistemas

planejamento do projeto de software

análise de requisitos

um

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

11

11

Revisão

unesp

Fase de Desenvolvimento

Focaliza "**como**" o software será desenvolvido

- como os dados vão ser estruturados
- como a função vai ser implementada como uma arquitetura de software
- como os detalhes procedimentais vão ser implementados
- como as interfaces vão ser caracterizadas
- como o projeto será traduzido em uma linguagem de programação
- como os testes serão efetuados

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

12

12

Revisão

unesp

Fase de Desenvolvimento

- Focaliza "como" o software será desenvolvido
- como os dados vão ser estruturados
- como a função vai ser implementada como uma arquitetura de software

três tarefas técnicas específicas deverão ocorrer sempre:

- projeto de software
- (geração de) código-fonte
- Inspeção e teste de software

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

13

13

Revisão

unesp

Fase de Manutenção

focaliza as "**mudanças**" que ocorrerão depois que o software for liberado para uso operacional

- A fase de manutenção reaplica os passos das fases de definição e desenvolvimento, mas faz isso no contexto de um software existente.


31/07/2025

Prof. Dr. Rogério Eduardo Garcia

14

14

Revisão

 **Fase de Manutenção**

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

15


- focaliza as "*mudanças*" que ocorrerão depois que o software for liberado para uso operacional
- A fase de manutenção reaplica os passos das fases de definição e desenvolvimento, mas faz isso no contexto

As mudanças estão associadas com

- *correção* de erros/defeitos
- *adaptações* exigidas conforme o ambiente do software evolui
- *mudanças* devido a *melhoramentos* ocorridos por alterações nos requisitos dos clientes

15

Revisão

 **Atividades de Apoio**

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

16

- As três fases genéricas do processo de software são complementadas por uma série de *atividades de apoio*.
- As atividades de apoio são aplicadas durante toda a engenharia do software

16

Revisão

unesp

Atividades de Apoio

- As três fases genéricas do processo de software são complementadas por uma série de atividades de apoio

Atividades típicas nessa categoria são:

- Controle e Acompanhamento do Projeto de Software
- Revisões Técnicas Formais
- Garantia de Qualidade de Software
- Gerenciamento de Configuração de Software
- Preparação e Produção de Documentos
- Gerenciamento de Reusabilidade
- Medidas

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
17

17

Revisão

unesp


Modelos de Processo de Software

- Existem vários modelos de processo de software (ou paradigmas de engenharia de software)
- Cada um representa uma tentativa de colocar ordem em uma atividade inerentemente caótica
- Pode-se citar os seguintes modelos de processo de software

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
18

18

Revisão




Modelos de Processo de Software

- O Modelo Sequencial Linear
 - também chamado Modelo Cascata
- O Modelo de Prototipação
- O Modelo RAD (Rapid Application Development)
- Modelos Evolutivos de Processo de Software
 - O Modelo Incremental
 - O Modelo Espiral
 - O Modelo de Montagem de Componentes
 - O Modelo de Desenvolvimento Concorrente
- Modelo de Métodos Formais
- Técnicas de Quarta Geração

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
19

19

Revisão

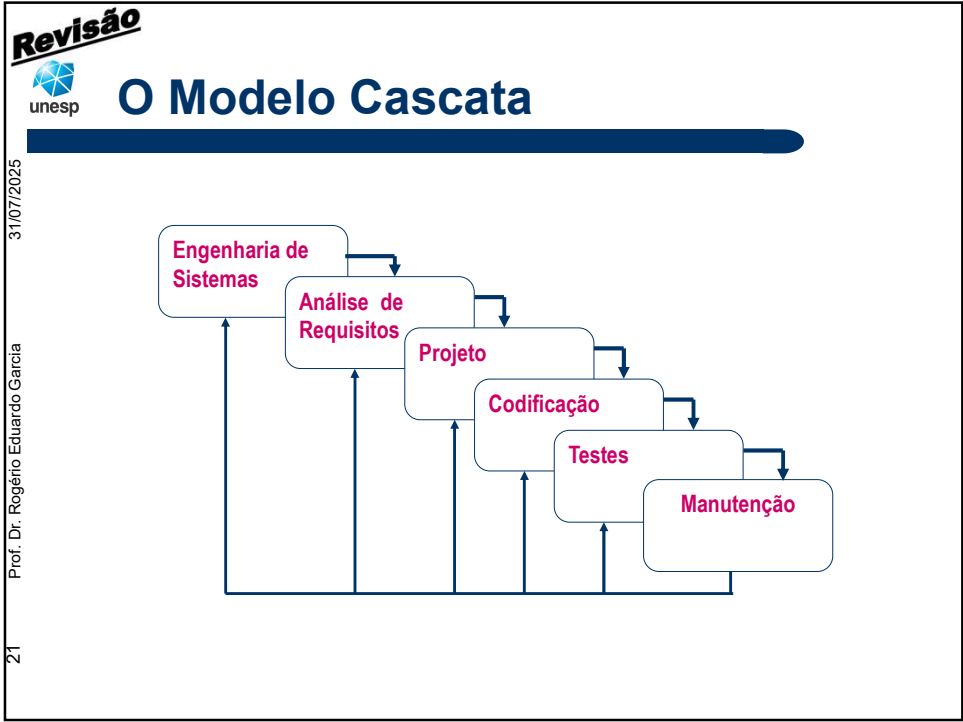


Modelos de Processo de Software

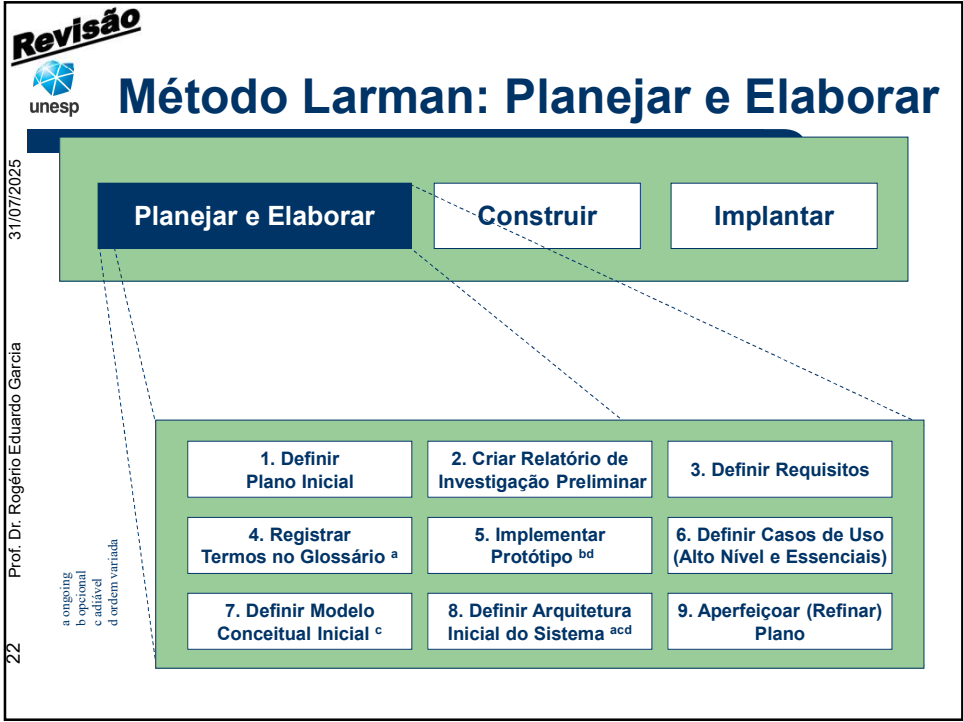
- O Modelo Sequencial Linear
 - também chamado **Modelo Cascata**
- O Modelo de Prototipação
- O Modelo RAD (Rapid Application Development)
- Modelos Evolutivos de Processo de Software
 - O Modelo Incremental
 - O Modelo Espiral
 - O Modelo de Montagem de Componentes
 - O Modelo de Desenvolvimento Concorrente
- Modelo de Métodos Formais
- Técnicas de Quarta Geração

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
20

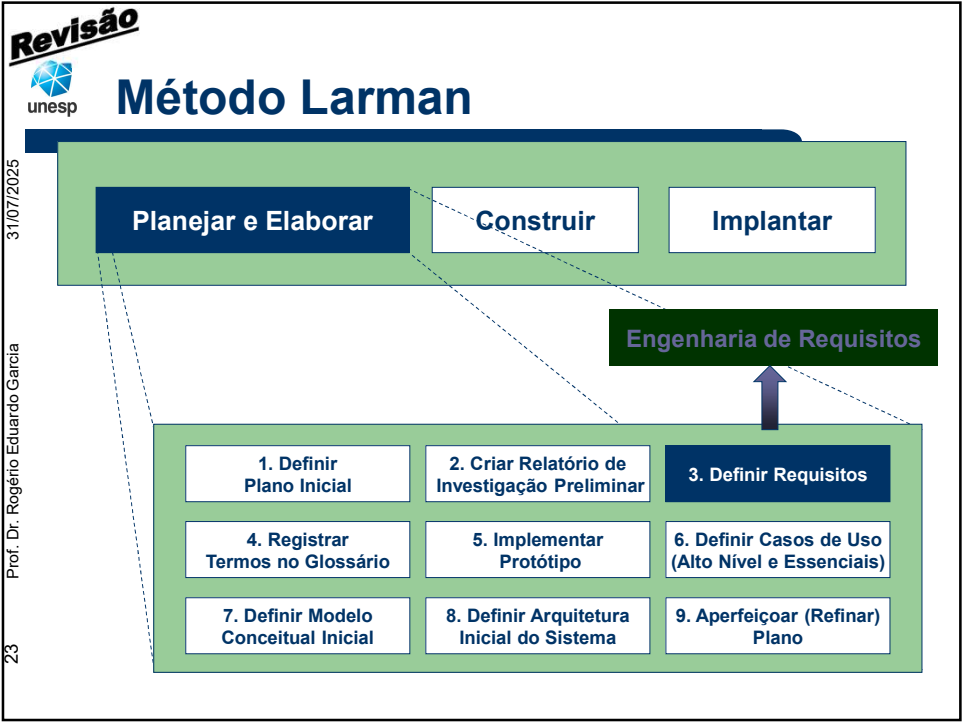
20



21



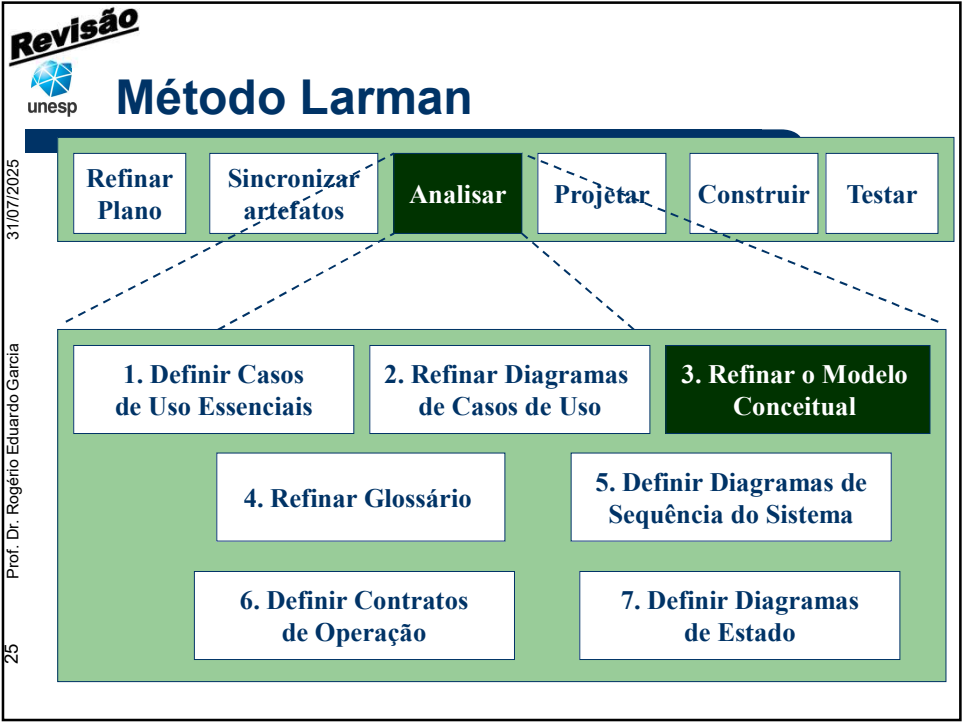
22



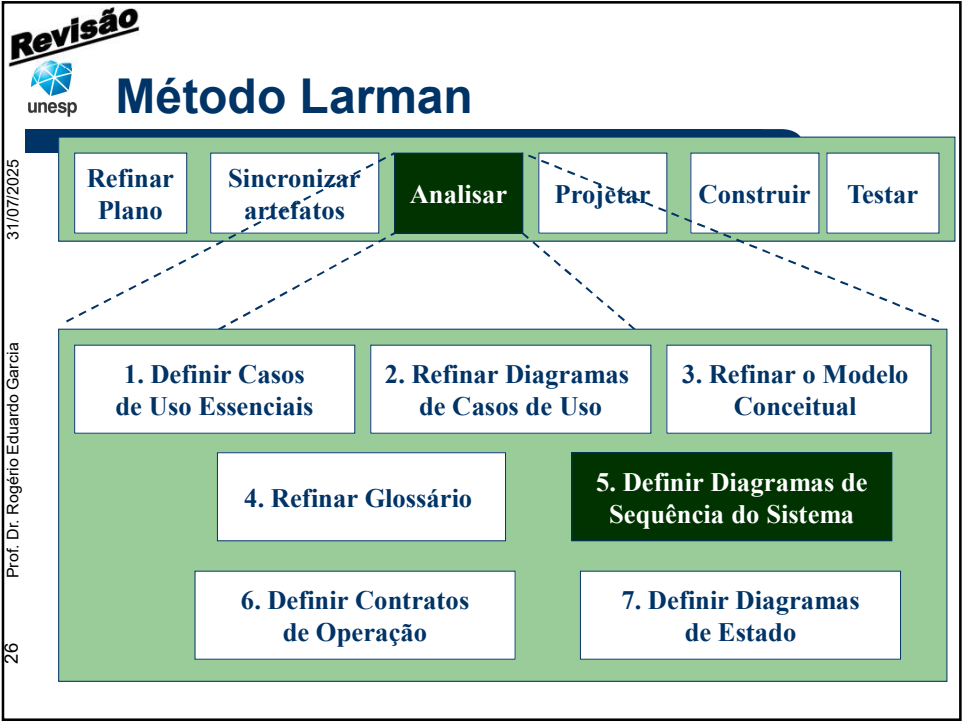
23



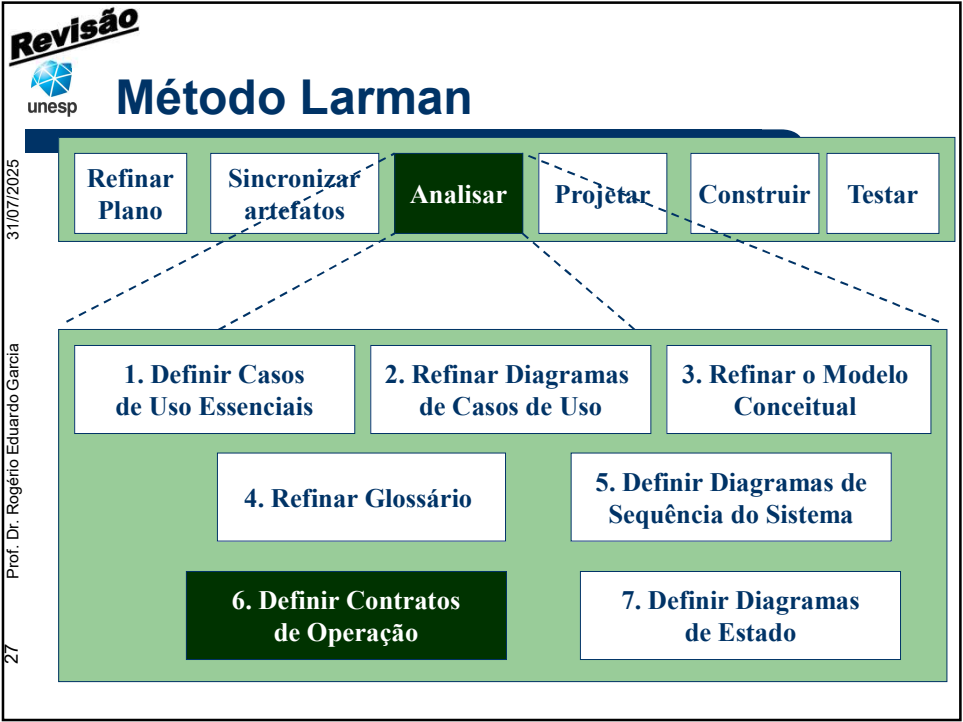
24



25



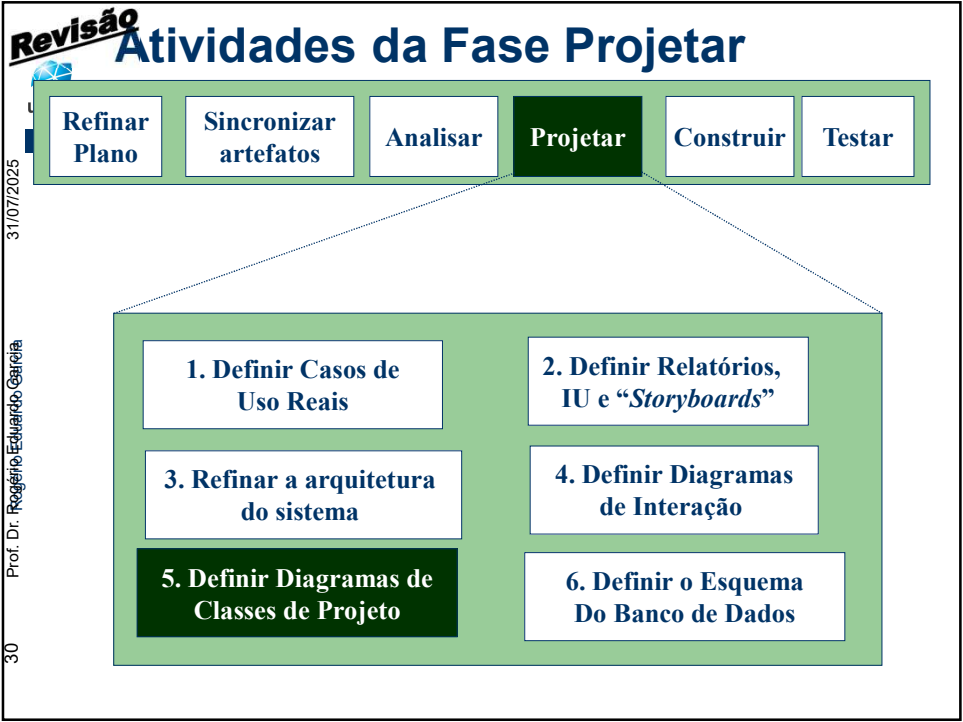
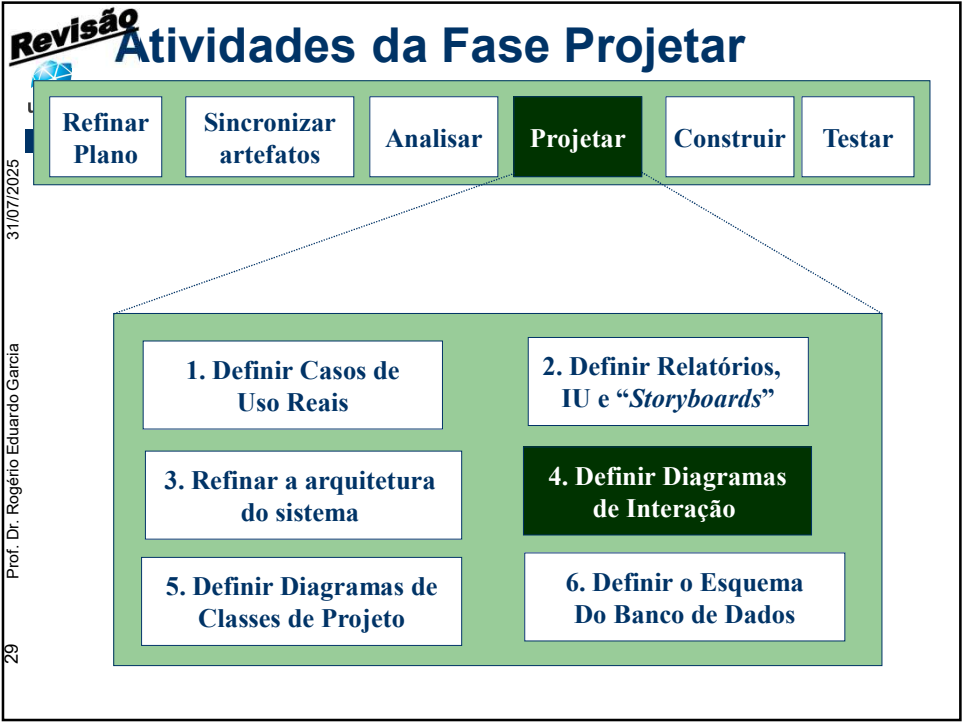
26




27



28



Revisão


unesp

31/07/2025

Prof. Dr. Rogério Eduardo Garcia


31

Padrões de Projeto (revisão)

Uma breve introdução e GRASP

31


Revisão


unesp

Padrões

- Desenvolvedores de software experientes criaram um repertório de princípios gerais e boas soluções para guiar a construção de softwares
- Essas soluções foram descritas em um formato padronizado (nome, problema, solução) e podem ser usadas em outros contextos


32

Revisão

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
33

Padrões

- Padrões usualmente não contêm novas idéias
 - organizam conhecimentos e princípios existentes, testados e consagrados
- **Padrão** é uma descrição nomeada de um problema e uma solução, que pode ser aplicado em novos contextos

33


Revisão

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
34

Padrões GRASP

- GRASP = *General Responsibility Assignment Software Patterns*
- Descrevem princípios fundamentais de atribuição de responsabilidade a objetos
- Alguns padrões GRASP principais:
 - Especialista (*Expert*)
 - Criador (*Creator*)
 - Coesão alta (*High Cohesion*)
 - Acoplamento fraco (*Low Coupling*)
 - Controlador (*Controller*)

34

Revisão



Controlador

31/07/2025


Prof. Dr. Rogério Eduardo Garcia

35

- Problema: Quem deve ser responsável por tratar um evento do sistema ?
- Solução: A responsabilidade de receber ou tratar as mensagens de eventos (operações) do sistema pode ser atribuída uma classe que:
 - represente todo o sistema, um dispositivo ou um subsistema – chamado de controlador fachada - OU
 - represente um cenário de um caso de uso dentro do qual ocorra o evento
 - TratadorDe<NomeDoCasoDeUso>, ControladorDe<NomeDoCasoDeUso>

35

Revisão



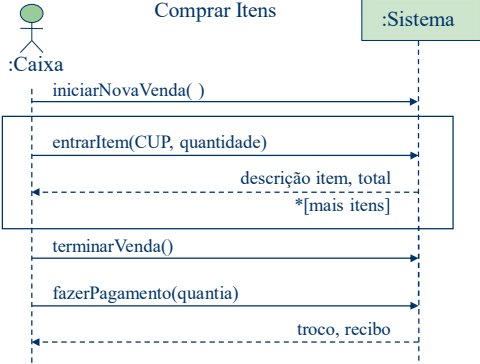
Controlador

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

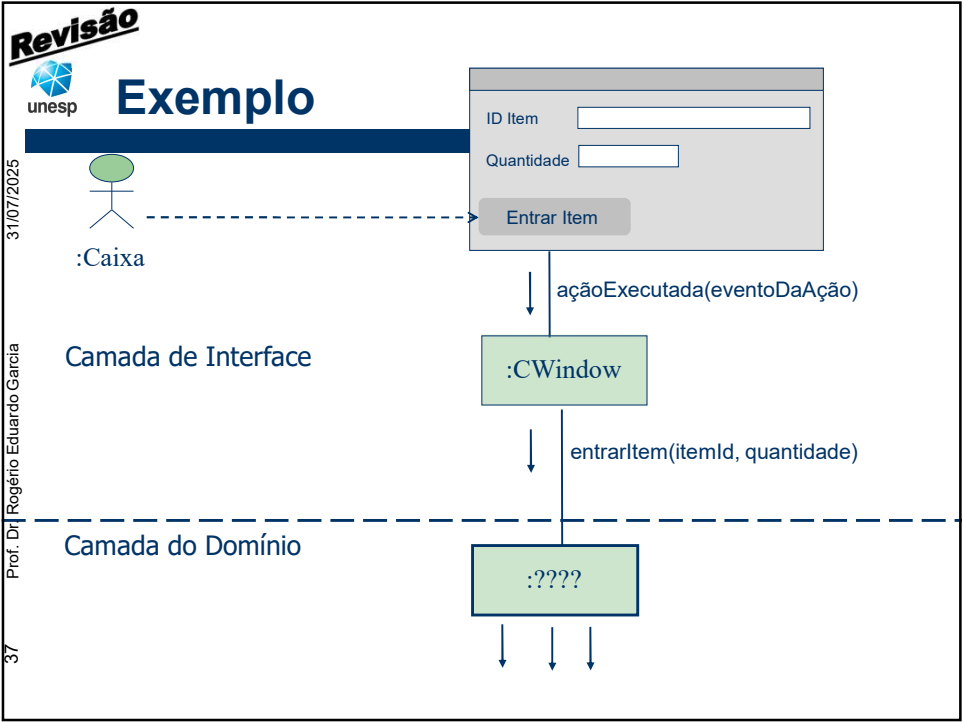
36

- Exemplo: quem vai tratar os eventos do sistema TPV?

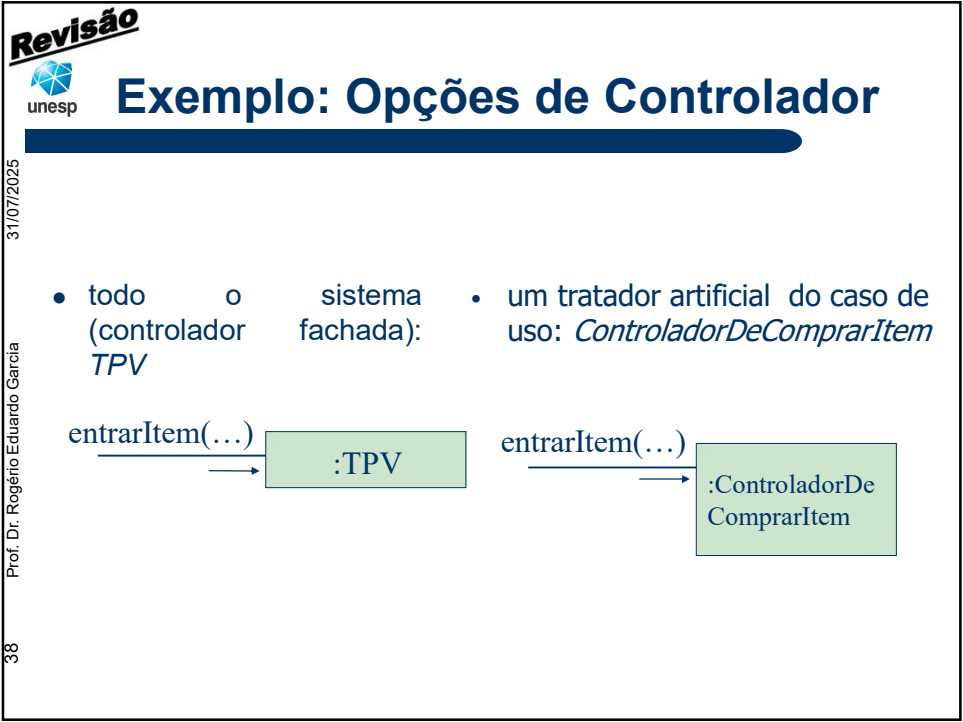


```
sequenceDiagram
    actor Caixa as :Caixa
    participant Sistema as :Sistema
    Caixa->>Sistema: iniciarNovaVenda()
    Caixa->>Sistema: entrarItem(CUP, quantidade)
    Sistema-->>Caixa: descrição item, total
    Note over Sistema: *[mais itens]
    Caixa->>Sistema: terminarVenda()
    Caixa->>Sistema: fazerPagamento(quantia)
    Sistema-->>Caixa: troco, recibo
```


36



37




38

Revisão

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
39

Discussão: Controladores Fachada

- Um controlador fachada deve ser um objeto (do domínio) que sugira uma cobertura sobre outras camadas e que seja o ponto principal para as chamadas provenientes da interface com o usuário ou de outros sistemas
 - pode ser uma abstração de uma entidade física – ex: TPV
 - pode ser um conceito que represente o sistema – ex: sistemaTPV
- São adequados quando não há uma quantidade muito grande de eventos de sistema
- Não é possível redirecionar mensagens do sistema para controladores alternativos (ex: outros subsistemas)


39

Revisão

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
40

Controladores de Casos de Uso

- Deve existir um controlador diferente para cada caso de uso
- Não é um objeto do domínio, e sim uma construção artificial para dar suporte ao sistema. Ex: *ControladorDeComprarItem*, *ControladorDeDevolução*
- Pode ser uma alternativa se a escolha de controladores fachada deixar a classe controladora com alto acoplamento e/ou baixa coesão (controlador inchado por excesso de responsabilidade)
- É uma boa alternativa quando existem muitos eventos envolvendo diferentes processos.


40

Revisão


Controladores inchados

- Classe controladora mal projetada - inchada
 - coesão baixa – falta de foco e tratamentos de muitas responsabilidades
- Sinais de inchaço:
 - uma única classe controladora tratando todos os eventos, que são muitos. Comum com controladores fachada
 - o próprio controlador executa as tarefas necessárias para atender o evento, sem delegar para outras classes (coesão alta, não especialista)
 - controlador tem muitos atributos e mantém informação significativa sobre o domínio, ou duplica informações existentes em outros lugares

41


Revisão


Controlador

- Curas para controladores inchados
 - acrescentar mais controladores – misturar controladores fachada e de casos de uso
 - delegar responsabilidades
- Corolário: objetos de interface (como objetos “janela”) e da camada de apresentação não devem ter a responsabilidade de tratar eventos do sistema

42

Revisão



Controlador

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

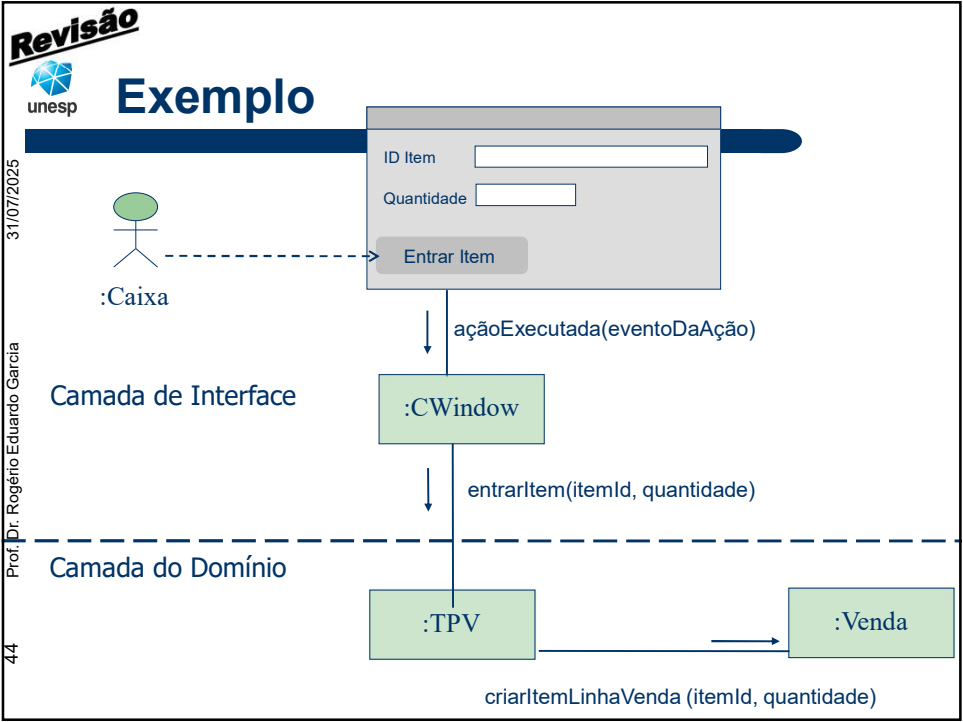
43

• Benefícios:

– aumento das possibilidades de reutilização de classes e do uso de interfaces “plugáveis”.


– conhecimento do estado do caso de uso – controlador pode armazenar estado do caso de uso, garantindo a sequência correta de execução de operações

43



44

Revisão




31/07/2025
Prof. Dr. Rogério Eduardo Garcia
45

Especialista

- Problema: qual é o princípio mais básico de atribuição de responsabilidades a objetos ?
- Solução: Atribuir responsabilidade ao especialista da informação.
- Exemplo: no sistema TPV, alguma classe precisa conhecer o total geral de uma venda

45

Revisão

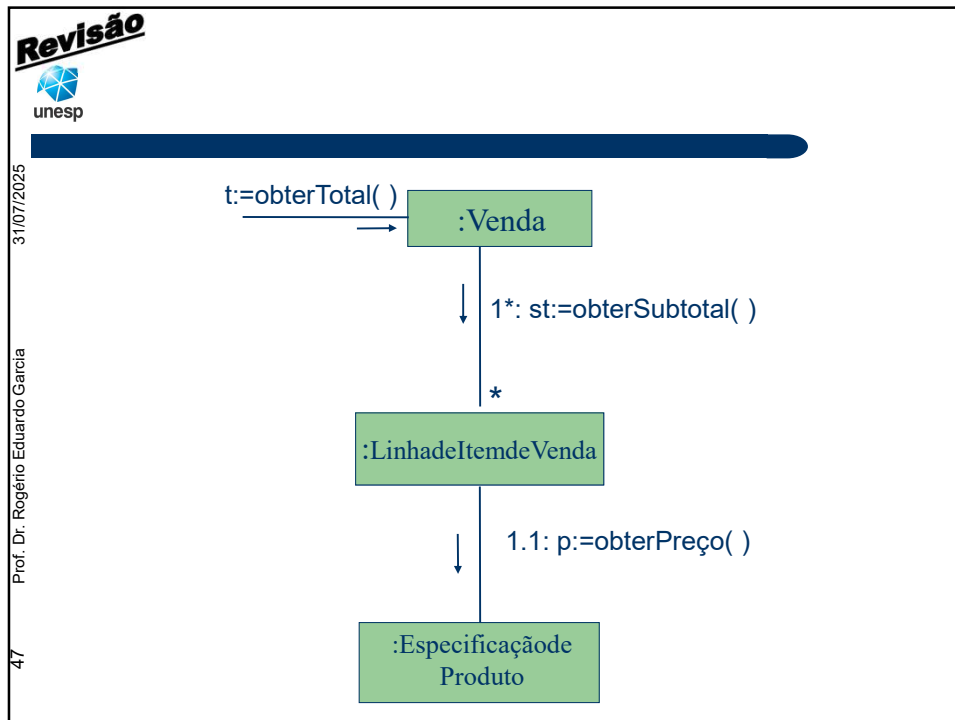


31/07/2025
Prof. Dr. Rogério Eduardo Garcia
46

Exemplo

- *Venda* → conhece o total da venda
- *ItemLinhaVenda* → conhece o subtotal da linha
- *EspecificaçãoProduto* → conhece o preço do produto.

46



47

Revisão

unesp

Especialista


31/07/2025

Prof. Dr. Rogério Eduardo Garcia

48

- Discussão
 - É o padrão mais utilizado
 - “Fazê-lo eu mesmo”
 - objetos fazem coisas relacionadas à informação que têm
 - Lembrar que existem especialistas parciais que colaboram numa tarefa
 - informação espalhada → comunicação via mensagens
 - Tem uma analogia no mundo real


48

Revisão


Especialista

- Benefícios:
 - Mantém encapsulamento → favorece o acoplamento fraco
 - O comportamento fica distribuído entre as classes que têm a informação necessária (classes “leves”) → favorece alta coesão
- Contraindicações
 - contra indicado quando aumenta acoplamento e reduz coesão
 - Ex: quem é responsável por salvar uma *Venda* no banco de dados?

49

Revisão


Criador

- Problema: Quem deveria ser responsável pela criação de uma nova instância de alguma classe ?
- Solução: atribua à classe B a responsabilidade de criar uma nova instância da classe A se uma das seguintes condições for verdadeira:
 - B agrega objetos de A
 - B contém objetos de A
 - B registra objetos de A
 - B usa objetos de A
 - B tem os valores iniciais que serão passados para objetos de A, quando de sua criação

50

Revisão

unesp

Criador

- Exemplo: No sistema TPV, quem é responsável pela criação de uma instância de *ItemLinhaVenda* ?

criarItemLinha (quantidade)

```

graph TD
    A[criarItemLinha (quantidade)] --> B[:Venda]
    B -- "1:criar(quantidade)" --> C[:ItemLinhaVenda]
  
```

➤ Venda contém vários itens de linha de venda

51

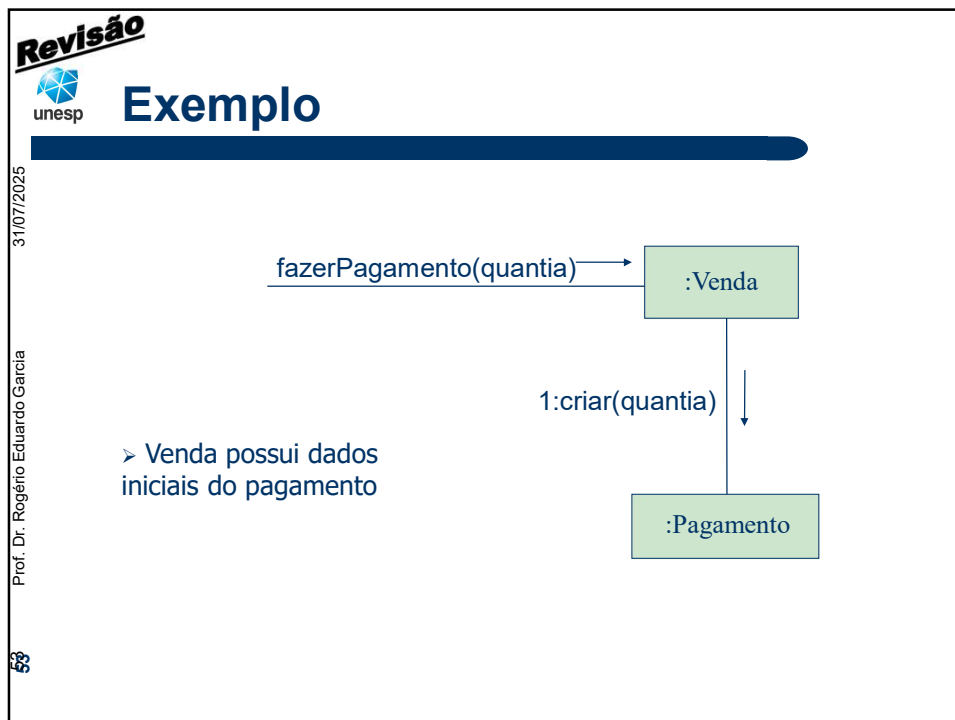
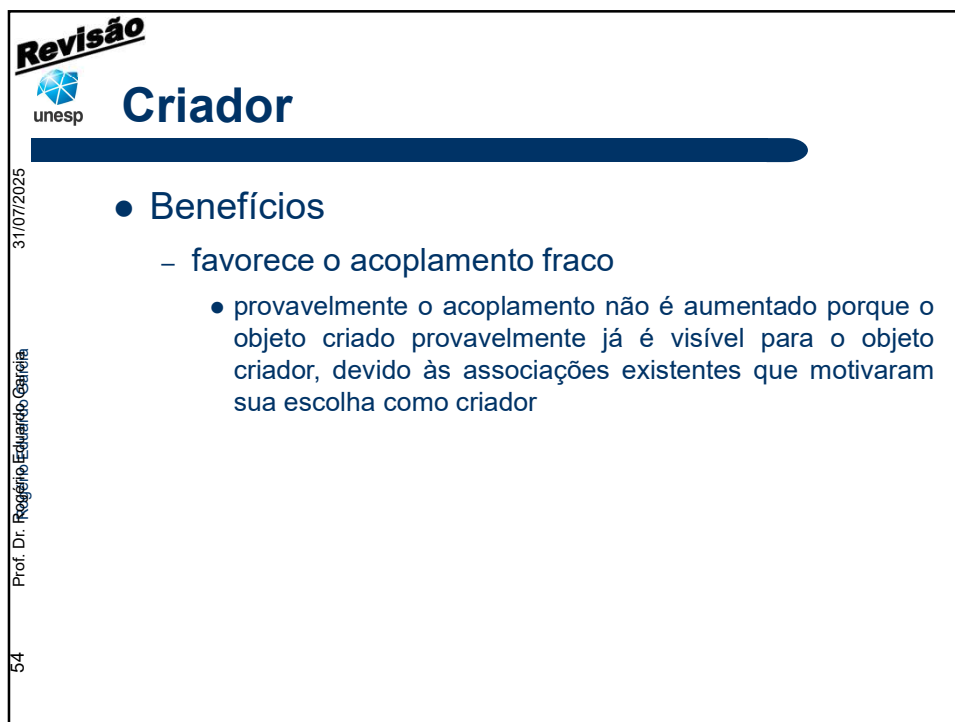
Revisão

unesp


Criador

- Discussão
 - objetivo do padrão: definir como criador o objeto que precise ser conectado ao objeto criado em algum evento
 - escolha adequada favorece acoplamento fraco
 - objetos agregados, contêineres e registradores são bons candidatos à responsabilidade de criar outros objetos
 - algumas vezes o candidato a criador é o objeto que conhece os dados iniciais do objeto a ser criado
 - Ex: *Venda* e *Pagamento*

52

5354

Revisão




Acoplamento

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
55

- Acoplamento: dependência entre elementos (classes, subsistemas,...), normalmente resultante de colaboração para atender a uma responsabilidade
- O acoplamento mede o quanto um objeto está conectado a, tem conhecimento de ou depende de outros objetos
 - acoplamento fraco (ou baixo) – um objeto não depende de muitos outros
 - acoplamento forte (ou alto) – um objeto depende de muitos outros

55

Revisão




Acoplamento

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
56

- Problemas do acoplamento alto:
 - mudanças em classes interdependentes forçam mudanças locais
 - dificulta a compreensão do objetivo de cada classe
 - dificulta reutilização

56

Revisão



31/07/2025

Prof. Dr. Rogério Eduardo Garcia


57

Acoplamento Fraco

- Problema: como favorecer a baixa dependência e aumentar a reutilização ?
- Solução: Atribuir responsabilidade de maneira que o acoplamento permaneça baixo.
- Exemplo: No sistema TPV, suponha que queremos criar uma instância de pagamento e associá-la à venda. Qual classe deve ser responsável por essa tarefa?

57

Revisão



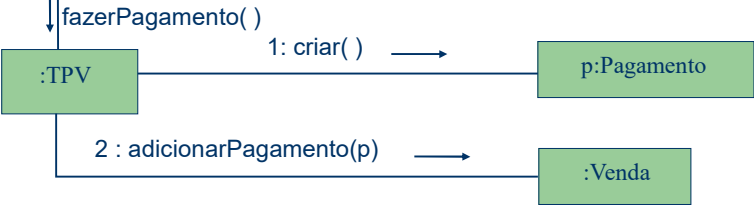
31/07/2025

Prof. Dr. Rogério Eduardo Garcia

58

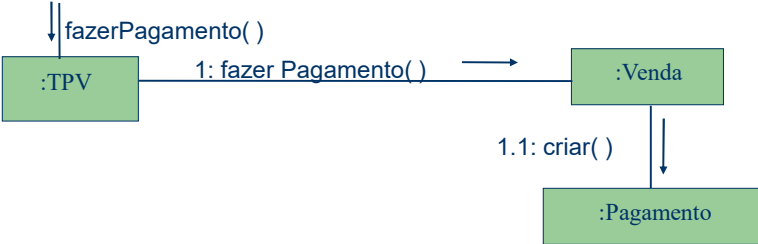
Qual?

Projeto 1: segundo padrão Criador, responsabilidade atribuída ao *TPV*



```
sequenceDiagram
    participant TPV as :TPV
    TPV->>TPV: fazerPagamento()
    TPV->>pPagamento as 1: criar()
    activate pPagamento
    deactivate pPagamento
    TPV->>Venda as 2: adicionarPagamento(p)
    activate Venda
    deactivate Venda
```

Projeto 2: alternativa – responsabilidade atribuída à *Venda*



```
sequenceDiagram
    participant TPV as :TPV
    TPV->>TPV: fazerPagamento()
    TPV->>Venda as 1: fazer Pagamento()
    activate Venda
    Venda->>Pagamento as 1.1: criar()
    activate Pagamento
    deactivate Pagamento
    deactivate Venda
```


58

Revisão

unesp

Qual projeto é melhor?

- Qual dos projetos anteriores favorece o acoplamento fraco?
 - em ambos os casos, *Venda* será acoplada a (terá conhecimento de) *Pagamento*
 - projeto 1 – acoplamento entre *Pagamento* e *TPV*
 - projeto 2 – não aumenta acoplamento

 **PREFERÍVEL**

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
59

59

Revisão


unesp

Formas de Acoplamento

- Um objeto tem um atributo que referencia um objeto de outra classe
- Um objeto tem um método que referencia um objeto de outra classe
 - parâmetro, variável local ou retorno
- Um objeto invoca os serviços de um objeto de outra classe
- Uma classe é subclasse de outra, direta ou indiretamente

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
60

60


Revisão


Acoplamento Fraco

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
61

- Discussão:
 - Acoplamento fraco → classes mais independentes
 - reduz impacto de mudanças
 - favorece reuso de classes
 - Considerado em conjunto com outros padrões
 - Extremo de acoplamento fraco não é desejável
 - fere princípios da tecnologia de objetos – comunicação por mensagens
 - projeto pobre: objetos inchados e complexos, responsáveis por muito trabalho → baixa coesão

61


Revisão


Acoplamento Fraco

31/07/2025
Prof. Dr. Rogério Eduardo Garcia
62

- Discussão:
 - dica: concentre-se em reduzir o acoplamento em pontos de evolução ou de alta instabilidade do sistema
 - ex: cálculo de impostos terceirizados no sistema TPV
- Benefícios:
 - classe são pouco afetadas por mudanças em outras partes
 - classes são simples de entender isoladamente
 - conveniente para reutilização


62

Revisão


Coesão

- **Coesão** mede o quanto as responsabilidades de um elemento (classe, objeto, subsistema,...) são fortemente focalizadas e relacionadas
- Objeto com Coesão Alta → objeto cujas responsabilidades são altamente relacionadas e que não executa um volume muito grande de trabalho
- Objeto com Coesão Baixa → objeto que faz muitas coisas não relacionadas ou executa muitas tarefas
 - difícil de compreender, reutilizar e manter
 - constantemente afetadas por mudanças

63


Revisão


Coesão Alta

- Problema: Como manter a complexidade sob controle?
- Solução: Atribuir responsabilidade de tal forma que a coesão permaneça alta.
- Exemplo (o mesmo para o acoplamento fraco): No sistema TPV, suponha que queremos criar uma instância de pagamento e associá-la à venda. Qual classe deve ser responsável por essa tarefa?

64

Revisão

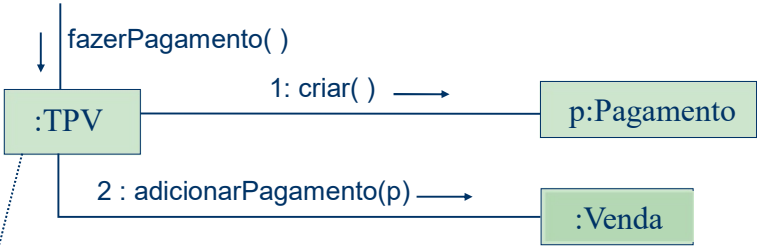
 **Exemplo**

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

65

Projeto 1: segundo padrão Criador, responsabilidade atribuída ao *TPV*




```
sequenceDiagram
    actor User
    User->>TPV: fazerPagamento()
    activate TPV
    TPV->>pPagamento: 1: criar()
    activate pPagamento
    deactivate pPagamento
    TPV->>Venda: 2: adicionarPagamento(p)
    activate Venda
    deactivate Venda
    deactivate TPV
```

O TPV toma parte na responsabilidade de fazer pagamento. Neste exemplo, isso seria aceitável, mas o que aconteceria se houvessem 50 mensagens recebidas por TPV?

65

Revisão

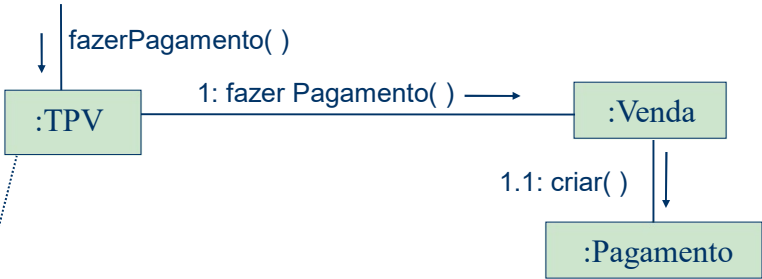
 **Exemplo**

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

66


Projeto 2: alternativa – responsabilidade atribuída à *Venda*



```
sequenceDiagram
    actor User
    User->>TPV: fazerPagamento()
    activate TPV
    TPV->>Venda: 1: fazer Pagamento()
    activate Venda
    Venda->>pPagamento: 1.1: criar()
    activate pPagamento
    deactivate pPagamento
    deactivate Venda
    deactivate TPV
```

Esta solução favorece uma coesão mais alta em TPV e também um acoplamento mais fraco. Portanto, projeto 2 é preferível.

66

Revisão


Coesão Alta


- Discussão:
 - Coesão alta, assim como Acoplamento Fraco, são princípios que devem ser considerados no projeto de objetos
 - má coesão traz acoplamento ruim e vice-versa
 - regra prática: classe com coesão alta tem um número relativamente pequeno de métodos, com funcionalidades relacionadas, e não executa muito trabalho
 - analogia com mundo real
 - ex: pessoas que assume muitas responsabilidades não associadas podem tornar-se (e normalmente tornam-se) ineficientes

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

67

67

Revisão


Coesão Alta

- Benefícios
 - mais clareza e facilidade de compreensão no projeto
 - simplificação de manutenção e acréscimo de funcionalidade/melhorias
 - favorecimento do acoplamento fraco
 - aumento no potencial de reutilização
 - classe altamente coesa pode ser usada para uma finalidade bastante específica

31/07/2025

Prof. Dr. Rogério Eduardo Garcia

68

68

