Projeto e Sistemas de Software 2002.2

Aula 01

Gustavo Robichez de Carvalho
Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Agenda

Processo de Desenvolvimento de Software
- Produto
- Definição

Introdução à UML

Introdução à Orientação a Objetos

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Processo de Desenvolvimento de Software

Gustavo Robichez de Carvalho
Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Objetivo

Objetivo da Engenharia de Software é desenvolver um produto.

Este produto poderá ser:

- Software
- Documentação
- Manuais

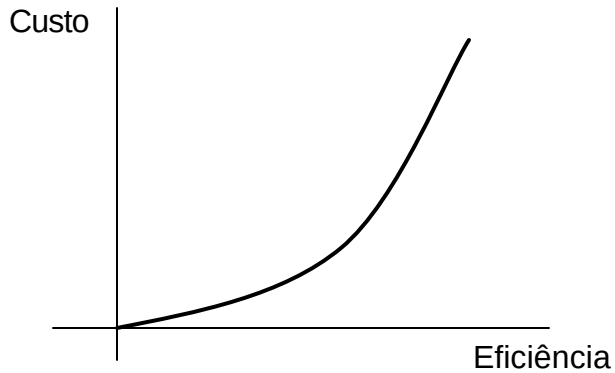
Características do produto

Características do produto	Descrição
Manutenibilidade	Facilidade para realizar modificações
Usabilidade	Facilidade de uso
Eficiência	Desempenho do produto
Confiabilidade	O quanto confiável é o produto desenvolvido
Acurácia	O produto gera resultados precisos?
Conformidade	Se o produto está de acordo com padrões, convenções ou regras estabelecidas

Algumas considerações...

O custo de modificação do produto geralmente é de 60% do custo total do projeto

Desenhar o produto de maneira flexível, facilitando futuras modificações é essencial.



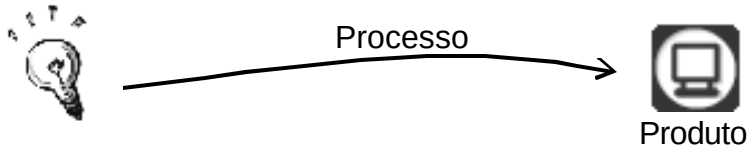
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Processo de desenvolvimento

O processo de software é um conjunto de atividades e resultados associados que produz um produto.

Ferramentas CASE (computer-aided software engineering) podem ser usadas para ajudar em algumas atividades do processo como modelagem e especificação do software.

- Ex.: Rose e Talisman



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Processo de desenvolvimento (continuação)

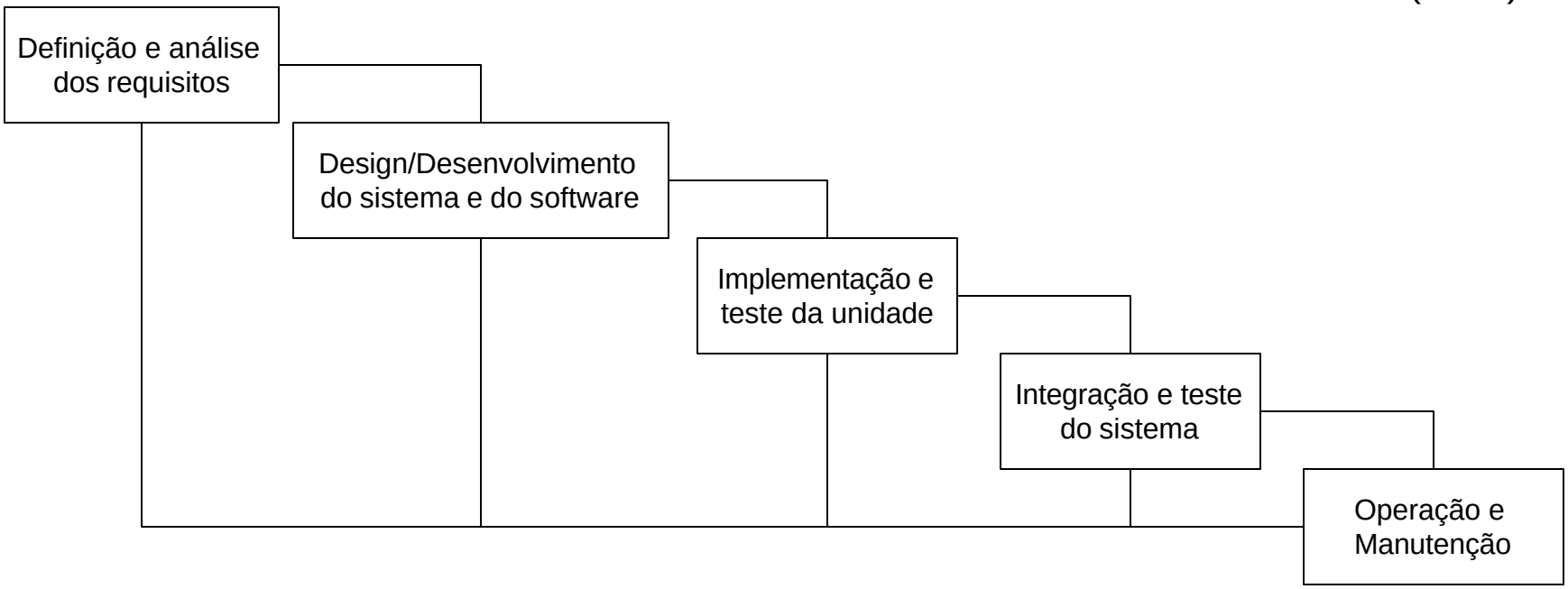
As atividades mais importantes do processo:

- Análise / Especificação do software - a funcionalidade do software e suas restrições precisam estar definidas;
- Design / Desenvolvimento do software - o software deve ser desenvolvido de acordo com as especificações;
- Teste / Validação do software - o software necessita ser validado para assegurar que ele faz o que o cliente deseja;
- Evolução do software - o software necessita evoluir para possibilitar as mudanças que o cliente necessita.

Modelo em Cascata

Ciclo de vida do software

(1970)



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

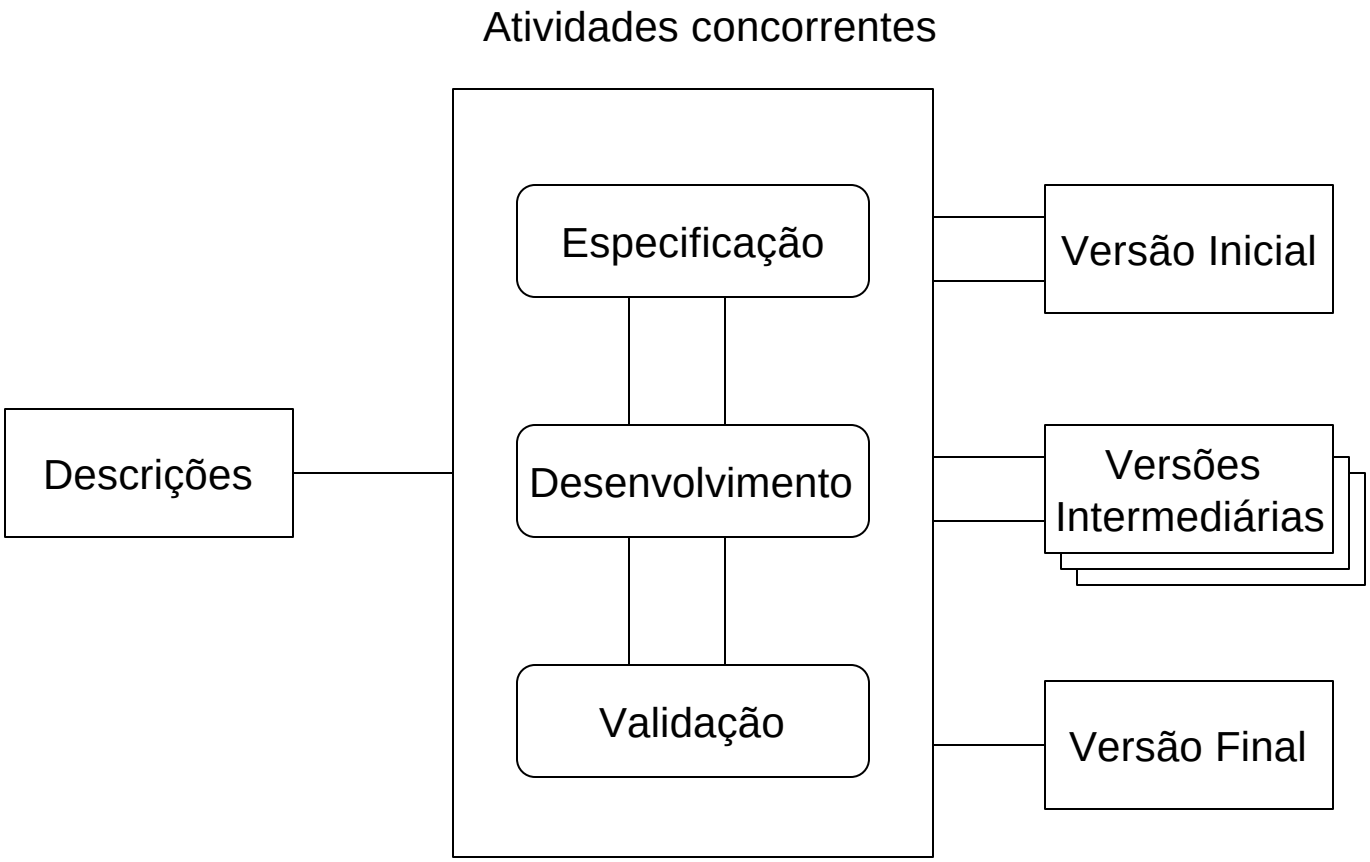
Desenvolvimento evolutivo

Dois tipos de desenvolvimento evolutivo:

- "Programação exploratória"
  - Explorar os requisitos junto com o cliente
  - Antes de implementar, entender muito bem os requisitos do software.
  - O sistema evolui a medida que novas características são adicionadas

- *"Throw-away prototyping"*
  - Construção de protótipos
    - Protótipos auxiliam no levantamento dos requisitos

Desenvolvimento evolutivo (continuação)

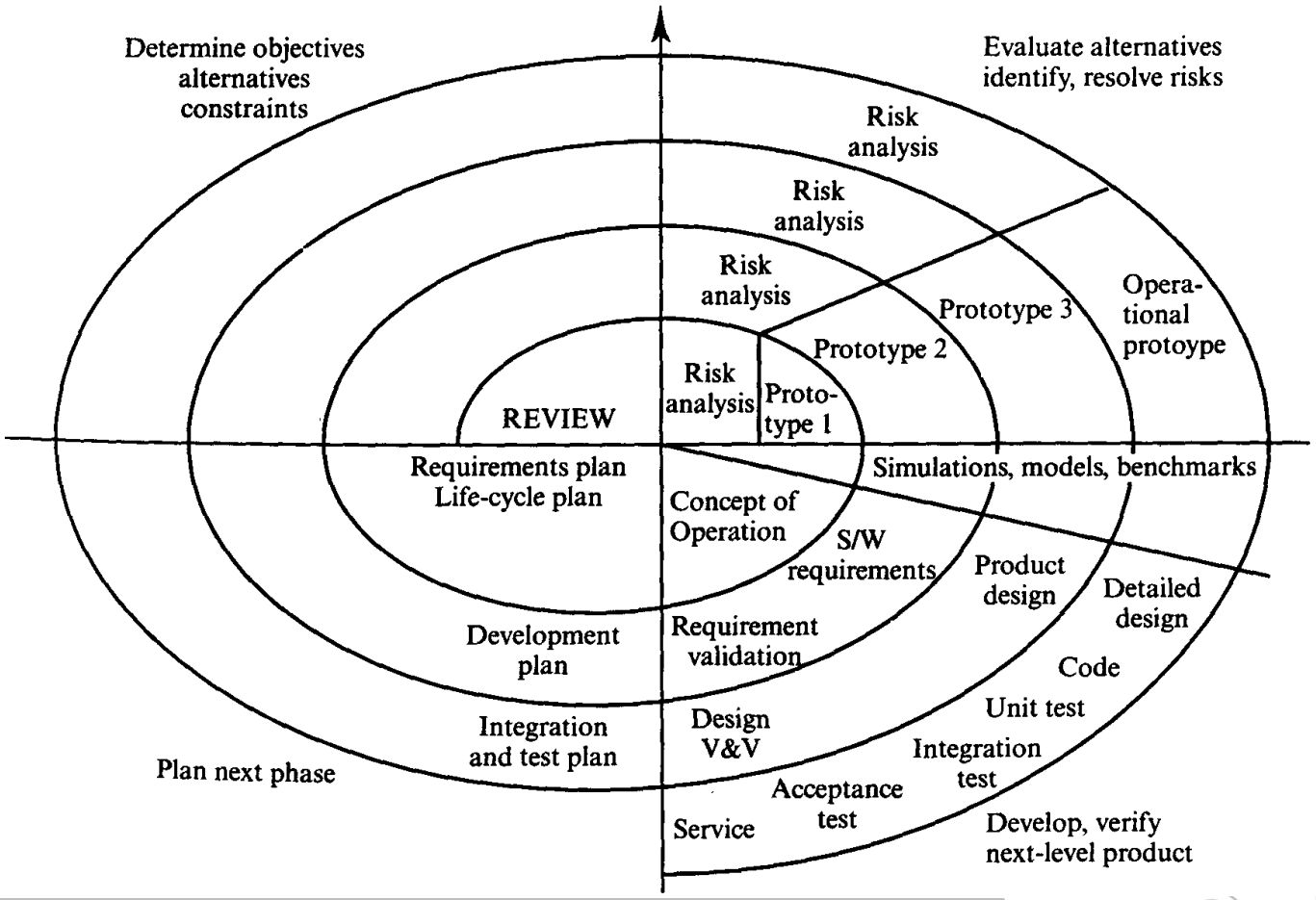


Desenvolvimento evolutivo (continuação)

O desenvolvimento evolutivo é mais apropriado para:

- Desenvolvimento de sistemas pequenos.
  - Re-implementação do sistema na sua íntegra sempre que mudanças significativas são necessárias;
- Desenvolvimento de sistemas com curto tempo de vida.
- Desenvolvimento de sistemas ou partes de um sistema maior onde é impossível detalhar muito as especificações. Exemplos deste tipo de sistemas são sistemas IA e interfaces de usuário.

Modelo em espiral de Boehm



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Visibilidade do processo

A maioria das organizações envolvidas no desenvolvimento de sistemas grandes usa um processo *"deliverable-oriented"* (orientado a entrega).

Cada atividade precisa terminar com a produção de alguma documentação. Esta documentação gera a visibilidade do processo.

Visibilidade do processo (continuação)

Atividade	**Documentação de saída**
Análise dos requisitos	Estudo dos possíveis requisitos Requisitos implícitos
Definição dos requisitos	Documento dos requisitos
Especificação do Sistema	Especificação funcional Aprovação do plano de teste Esboço do manual do usuário
Design da arquitetura	Especificação arquitetural Plano de teste do sistema
Design da Interface	Especificação da interface Plano do teste de integração
Codificação	Código do programa
Teste da unidade	Relatório do teste da unidade

Visibilidade do processo (continuação)

Atividade	Documentação de saída
Teste do módulo	Relatório do teste do módulo
Teste da integração	Relatório do teste da integração
	Versão final do manual do usuário
Teste do sistema	Relatório do teste do sistema
Teste	Sistema final mais sua documentação

Visibilidade do processo (continuação)

Comparação entre os modelos de processo

Modelo de Processo	Visibilidade do Processo
Modelo em cascata	Boa visibilidade. Cada atividade produz alguma informação.
Desenvolvimento evolutivo	Fraca visibilidade. Não é "econômico" produzir documentação durante interações rápidas.
Modelo em Espiral	Boa visibilidade. Cada segmento e cada parte da espiral pode produzir alguma documentação.

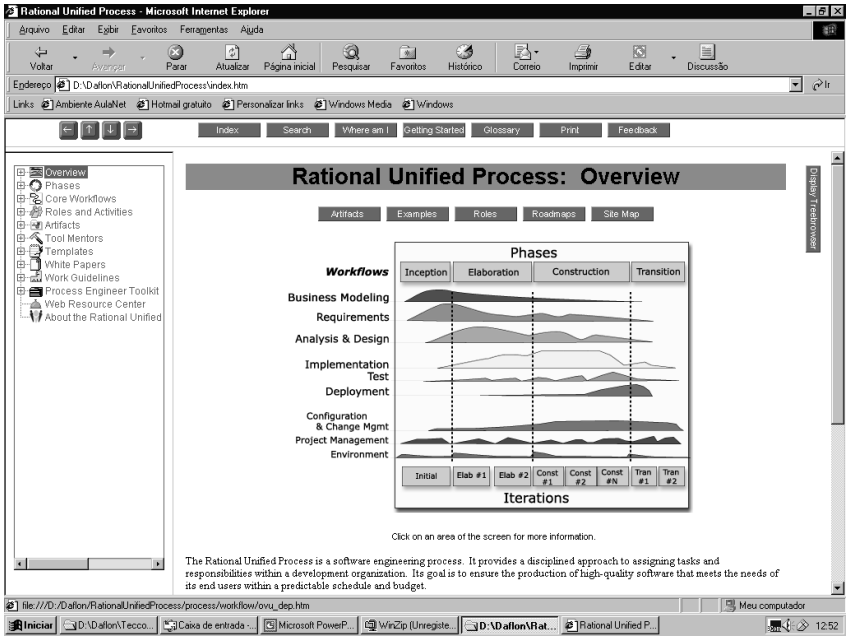
Rational Unified Process

O RUP é um processo de engenharia de software.

- Abordagem disciplinar de atribuir tarefas e responsabilidades em um processo de desenvolvimento de uma organização.

O RUP é um Framework.

- Pode ser instanciado gerando diversos processos.
  - Dependentes da organização.
  - Dependentes da aplicação.
    - Nenhum processo é adequado para todos as organizações/aplicações.



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Rational Unified Process

Propósito:

- Assegurar um excelente nível de qualidade na produção de artefatos de software em uma organização.
- Gerenciar cronogramas e orçamentos.
- Aumentar a produtividade da equipe.
  - Provê a cada membro acesso a uma base de conhecimento única.

Base de conhecimento:

- Guias,
- Modelos e
- Mentores de ferramentas.

Atividades criam e mantém modelos.

- Promove o uso efetivo de Unified Modeling Language (UML).

Utiliza ferramentas para:

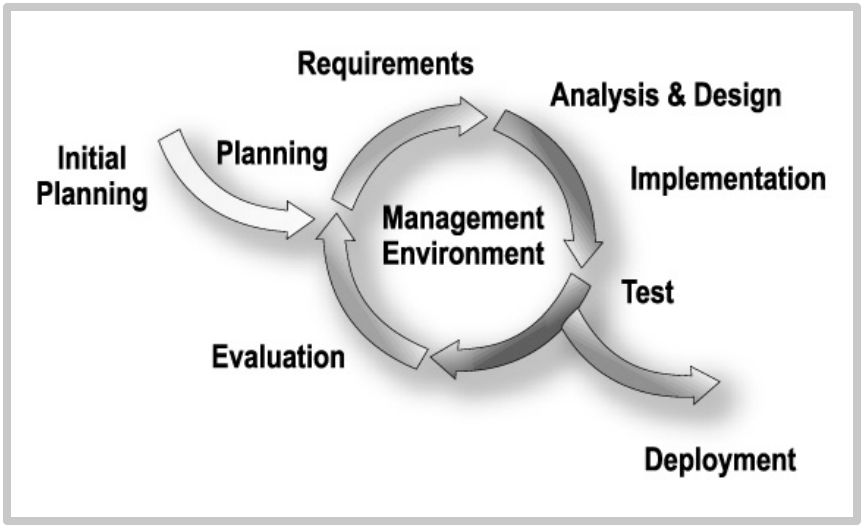
- Automatizar parte do processo,
- Criar e manter os diversos artefatos/modelos,
- Gerar modelagem visual,
- Programar,
- Testar, etc.

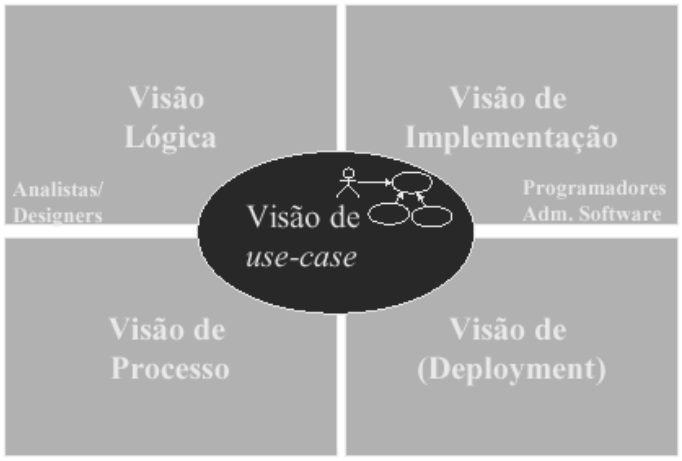
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Características

Pontos principais:

- Visão macro
  - Pode ser considerado um processo de desenvolvimento serial.
- Visão mais detalhada
  - Assume características de um processo iterativo e incremental.
- Dirigido a Use Cases.
- Arquitetura centralizada:
  - Serve como base para a reutilização.
  - Facilita:
    - O desenvolvimento voltado a componentes.
    - Administração da complexidade do projeto.
    - Manutenção da integridade do sistema.





Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Práticas Recomendadas

Desenvolvimento Iterativo

- Desenvolvimento de sistemas complexos geram dificuldades:
  - Na definição do problema
  - Na construção do software
  - Em se validar o produtos
    - Através de testes.
- Processo iterativo:
  - Entendimento incremental do problema.
  - Refinamentos sucessivos.
  - Desenvolvimento da solução após múltiplas iterações.
  - Avaliação de riscos em cada estágio do ciclo de vida.
  - Geração de versões executáveis a cada ciclo.

Gerenciamento de requisitos

- Descreve como fazer a elicitação, organização e documentação de restrições e funcionalidades.
- Utiliza Use cases e cenários como principais artefatos.
- Guia para as disciplinas de design, implementação e teste.
- Rastreabilidade.

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Práticas Recomendadas

Arquitetura baseada em componentes

Componentes:

- Módulos não triviais.
- Subsistemas que desempenham uma função bem definida.

Arquitetura:

- Flexível.
- Acomoda mudanças.
- Semântica.
- Promove reuso.

Modelagem Visual de Software

Representação gráfica:

- Capturar a estrutura e o comportamento da arquitetura e seus componentes.
- Utilizar de UML.
- Promover a compreensão do problema e de sua respectiva solução.
- Diminuir a ambigüidade na comunicação.

Práticas Recomendadas

Verificação da qualidade de software

Pontos críticos de uma solução

- performance da aplicação
- confiança
- conformidade com os requisitos

Dar suporte ao planejamento, desenvolvimento, execução e avaliação de casos de testes

Avaliação de qualidade

- Intrínseca ao processo
  - em suas atividades
  - envolvendo os seus participantes.
- Utiliza métricas e critérios.

Controle de mudanças no software

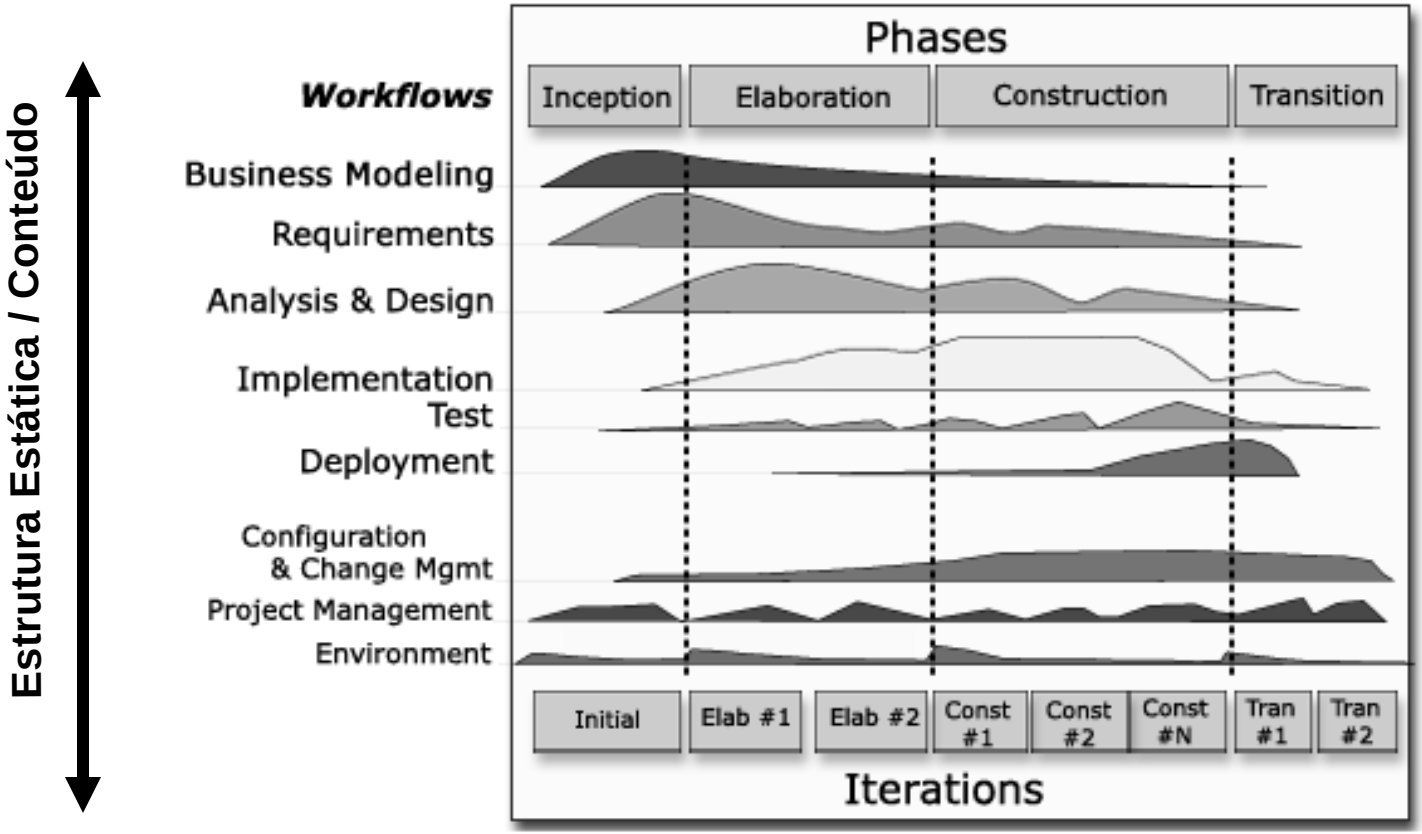
Evolução do software

- Processo iterativo
- Habilidade em gerenciar, aceitar e rastrear mudanças

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Visão geral

Estrutura Dinâmica / Tempo

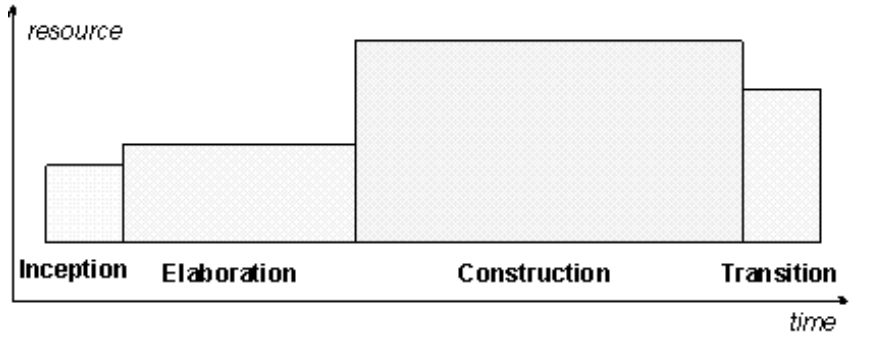


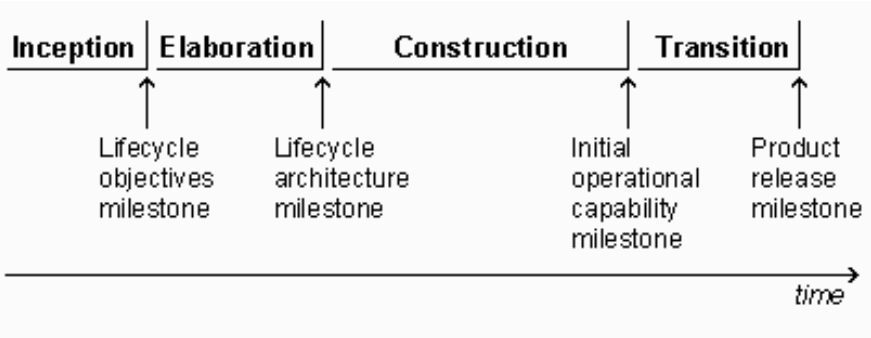
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Fases

Ponto de vista gerencial:

- Ciclo de vida do RUP é dividido em quatro fases seqüenciais.
  - Concepção
  - Elaboração
  - Construção
  - Transição
- Concluídas com pontos de controle.





Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Fase de Concepção

Durante esta fase é determinado o escopo do projeto:

- Identificar atores.
- Gerar plano de negócio:
  - Critérios de sucesso,
  - Avaliação de riscos,
  - Estimativas de recursos,
  - Cronograma com os principais pontos de controle.

Artefatos:

- Documento de visão.
  - Visão geral dos principais requisitos de negócios, palavras chaves e principais restrições.
- Modelo inicial de use case.
  - 10%-20% completo.
- Glossário inicial do projeto.
- Plano de negócios:
  - Contexto de negócio.
  - Critérios de sucesso.
  - Previsão financeira.
  - Avaliação inicial de riscos.
- Plano de projeto:
  - fases e interações
- Modelo de negócio.
- Protótipos.

Fase de Concepção

Ponto de controle: Objetivos do ciclo de vida

- Critérios de avaliação:
  - Concorrência de interesses na definição de estimativas de custos/cronogramas.
  - Entendimento de requisitos
  - Avaliação da credibilidade de estimativas de custos/cronogramas, prioridades, riscos e o processo de desenvolvimento.
  - Avaliação e detalhamento de protótipos que exemplifiquem a arquitetura.
  - Avaliação de Gastos Correntes contra Gastos Estimados

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Fase de Elaboração

Propósito:

- Analisar o domínio do problema.
  - Noção do todo, sem se ater a detalhes.
- Estabelecer fundamentos arquiteturais.
- Desenvolver Plano de Projeto.
- Eliminar principais fatores de risco.

Fase mais crítica.

Decisões arquiteturais:

- Escopo,
- Funcionalidades principais,
- Requisitos não funcionais,
- Requisitos performance.

Estabilidade:

- Arquitetura
- Requisitos
- Planos

Protótipo executável da arquitetura:

- Atender aos use cases críticos identificados na fase de concepção.
- Expor os principais riscos do projeto.
- Devem ser desenvolvidos diversos protótipos para amadurecer o conhecimento do problema.

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Fase de Elaboração

Artefatos:

- Modelo de use-cases
  - mínimo 80% completo
- Requisitos suplementares.
  - Correspondentes aos requisitos não funcionais
- Descrição da arquitetura de software.
- Protótipo executável da arquitetura.
- Riscos e plano de negócios revisado.
- Plano para desenvolvimento do projeto.
- Manual do usuário preliminar.

Ponto de controle : Ciclo de vida arquitetural

Examinar escopo e objetivos detalhados do sistema, a escolha de arquitetura e a resolução dos riscos principais.

Resposta a perguntas:

- A definição do produto está estável?
- A definição da arquitetura está estável?
- Riscos estão claros e devidamente tratados?
- Plano para a fase de construção está suficientemente detalhado e preciso? E suas estimativas?
- Todos os interessados concordam com a visão corrente?
- É aceitável alocação de recursos versus planejamento de gastos aceitáveis?

Fase de Construção

Propósito:

- Desenvolver componentes e funcionalidades.
- Fazer a integração de componentes.
- Realizar testes.
- Gerenciar recursos.
- Otimizar controle operacional:
  - Custos.
  - Tempo de projeto.
  - Qualidade.

Desenvolvimento em paralelo:

- Aumenta a complexidade de gerenciamento e sincronização do fluxo de trabalho.

Arquitetura bem definida na Fase de Elaboração:

- Facilidade na Fase de Construção.

Artefatos:

- Produto
  - Componentes integrados na plataforma adequada.
- Manual do usuário.
- Descrição da versão corrente.

Ponto de controle: Capacidade operacional inicial

- Verificar se o produto, o ambiente, os usuários estão preparados para trabalhar no estágio operacional sem expor o projeto a riscos elevados.
  - Versão beta.

Resposta a estas perguntas:

- O produto está estável e maduro para ser utilizado pela comunidade de usuários?
- Todos os interessados estão preparados para a transição?
- Os recursos gastos versus os recursos planejados ainda estão aceitáveis?

Fase de Transição

Propósito:

- Iniciada quando a *baseline* está estável para ser entregue ao usuário final.
  - Nível de qualidade aceitável.
  - Documentação disponível.
  - Correção de problemas críticos.
  - Testes beta para validar o produto.
- Entregar o produto a comunidade de usuários.
- Operações paralelas com o sistema legado que está sendo substituído.
- Conversão de banco de dados operacionais
- Treinamento dos usuários e pessoal de suporte

Ponto de controle: Versão do produto

Verificar se os objetivos foram atingidos:

- Respostas a estas perguntas:
  - É necessário iniciar um novo ciclo de desenvolvimento?
  - O usuário está satisfeito?
  - Os recursos gastos versus os recursos planejados ainda estão aceitáveis?

Iterações

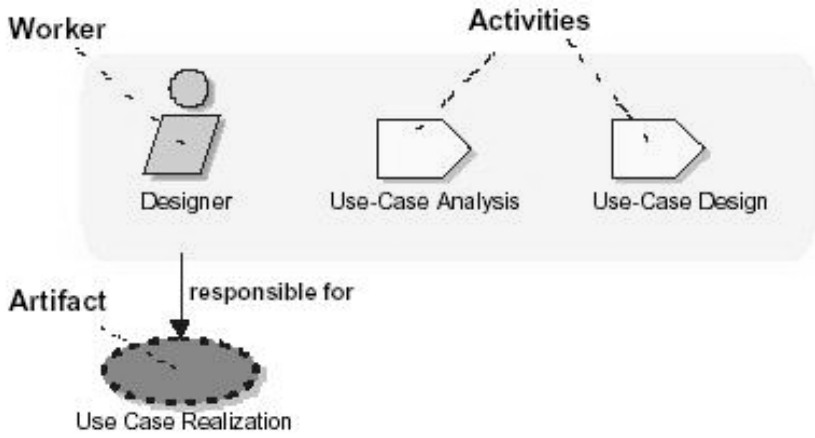
Um ciclo completo de desenvolvimento que resulta em um *release* (interno ou externo)

- Produto executável.
- Subconjunto do produto em desenvolvimento.
- Crescimento incremental.
- Cada fase no RUP pode ser quebrada em diversas iterações.
- Benefícios da abordagem iterativa:
  - Riscos são identificados no início.
  - Mudanças são mais fáceis de serem gerenciáveis.
    - Alto grau de reuso.
    - Aprendizado ao longo da iteração.
    - Melhoria na qualidade.

Estrutura estática do processo

Um processo descreve quem está fazendo o que, como e quando.

- Papéis
  - *Quem*
- Atividades
  - *Como*
- Artefatos
  - *O que*
- Disciplinas
  - *Quando*



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Papel

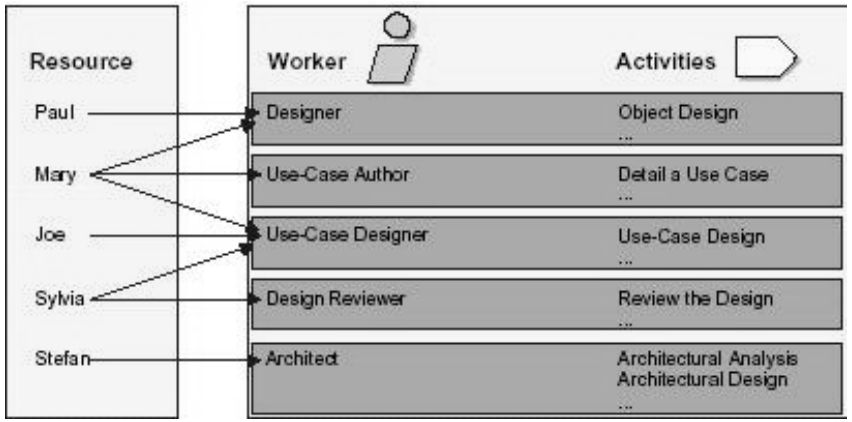
É uma definição abstrata para um conjunto de atividades realizadas e seus artefatos.

Um papel pode ser realizado por:

- uma pessoa ou
- uma equipe

Um papel corresponde a:

- Responsabilidades e
- Comportamentos.



Exemplos

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Atividades e Artefatos

Atividade

- Uma atividade é uma unidade de trabalho atribuída a um papel.
- A atividade tem um propósito claro.
  - Criação e atualização de artefatos.
- A atividade deve ser utilizada como elemento para planejamento e controle de progresso.

Exemplo de atividades:

- Planejar uma iteração.
  - Papel: Gerente de Projeto.
- Identificar atores e use cases.
  - Papel: Analista de Sistemas.
- Revisar o design.
  - Papel: Revisor
- Executar testes de performance.
  - Papel: Testador de Performance

Atividades e Artefatos:

- As atividades estão estritamente ligadas a artefatos.
- Os artefatos são as entradas ou saídas de uma atividade.
- Os artefatos servem como mecanismo de comunicação entre as atividades.

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

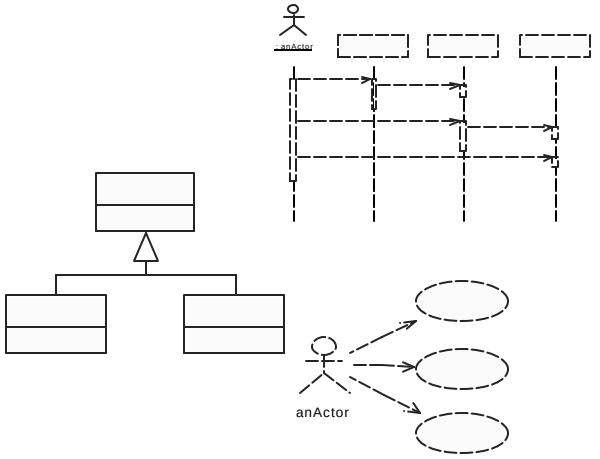
Artefatos

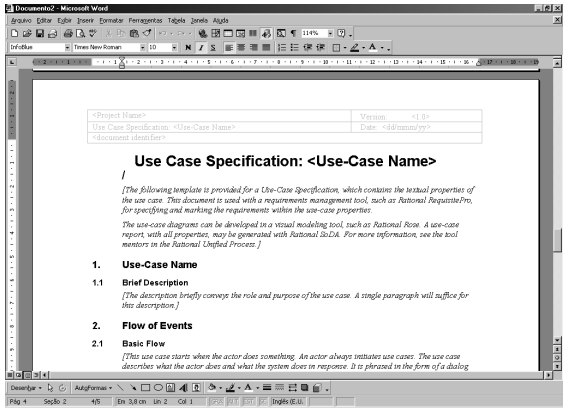
Artefato

- Um artefato é um pedaço de informação que é produzido, modificado ou utilizado pelo processo.
- Artefatos podem ser:
  - Modelos.
  - Elementos de Modelo.
  - Documentos.
  - Código Fontes.
  - Executáveis.

Artefatos podem ser expresso:

- Visualmente.
- Textualmente.





Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Disciplinas

Disciplinas são **seqüências de atividades** que produzem artefatos e esclarecem as interações entre papéis.

Para cada disciplina são abordados os seguintes tópicos:

Introdução

Conceitos

Workflow

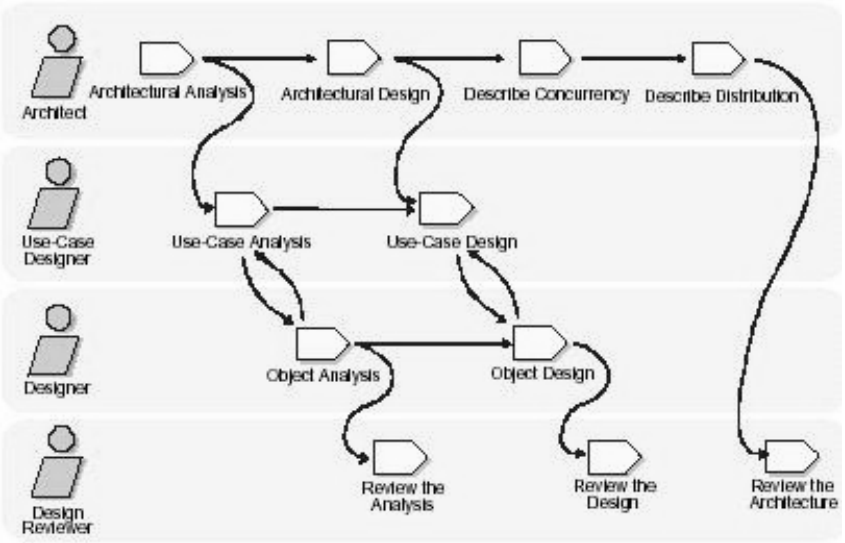
Atividades

Artefatos

Guias

Em UML, uma disciplina pode ser expressa como:

- Diagrama de seqüência, ou
- de colaboração, ou
- de atividade.

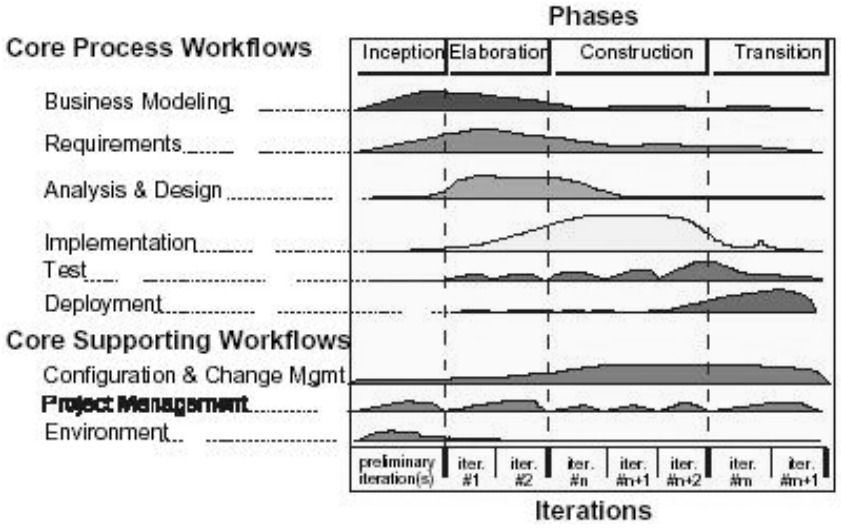


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Disciplinas

Existem 9 disciplinas:

- Modelagem de negócio
- Requisitos
- Análise e Design
- Implementação
- Teste
- Deployment
- Gerenciamento de mudanças e configuração
- Gerência de projetos e Ambiente

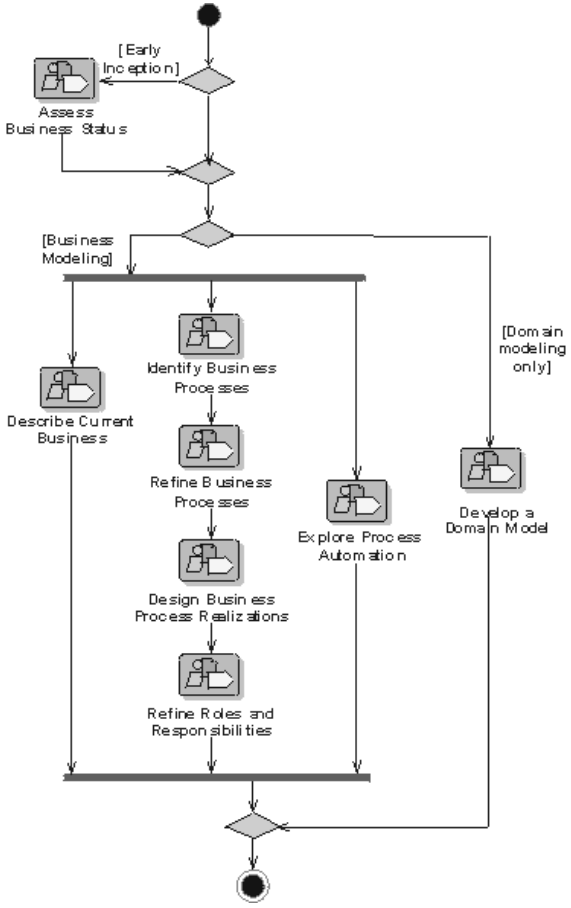


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Modelagem de Negócio

Propósito:

- Entender a estrutura e a dinâmica da organização.
  - Identificar os problemas atuais e analisar possíveis melhorias.
  - Derivar os requisitos necessários para atender os objetivos.
- Prover uma linguagem comum para ambas as comunidades.
  - Clientes, usuários e desenvolvedores.
- Dar tratamento a dificuldades na reengenharia do negócio:
  - Problemas na comunicação entre engenharia de negócio e a engenharia de software.
    - O produto final da engenharia de negócio não é a entrada no processo de desenvolvimento do software.

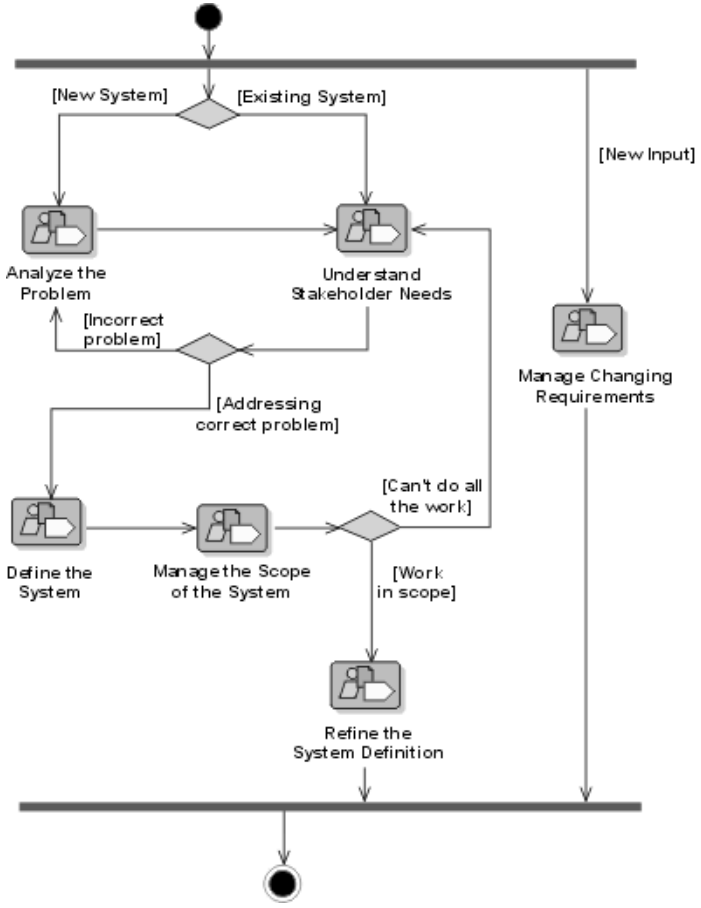


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Requisitos

Propósitos:

- Descrever ***o quê*** o sistema deve fazer.
  - Promover o entendimento dos requisitos do sistema aos desenvolvedores.
  - Permitir que desenvolvedores e clientes cheguem a um acordo.
  - Definir fronteiras do sistema.
    - O quê o sistema deverá ou não fazer.
- Prover uma base para estimar o custo e tempo de desenvolvimento do projeto.
- Organizar e documentar funcionalidades e restrições.
- Rastrear e documentar desafios e decisões.

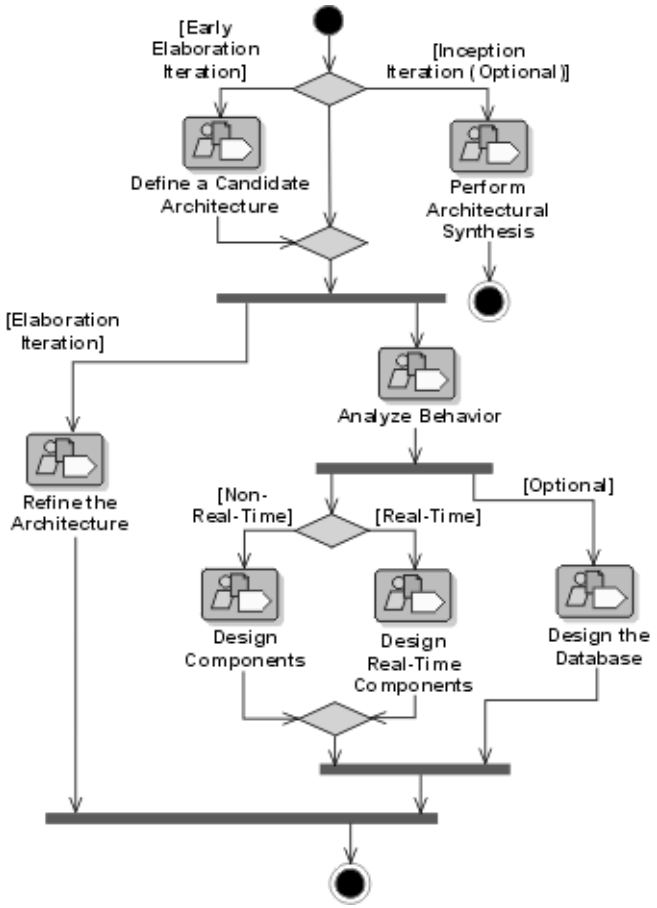


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Análise e Design

Propósito:

- Transformar os requisitos em design para a implementação do sistema.
- Desenvolver uma arquitetura para o sistema.
- Adaptar o design para se tornar compatível com o ambiente de implementação.
- Projetar performance.

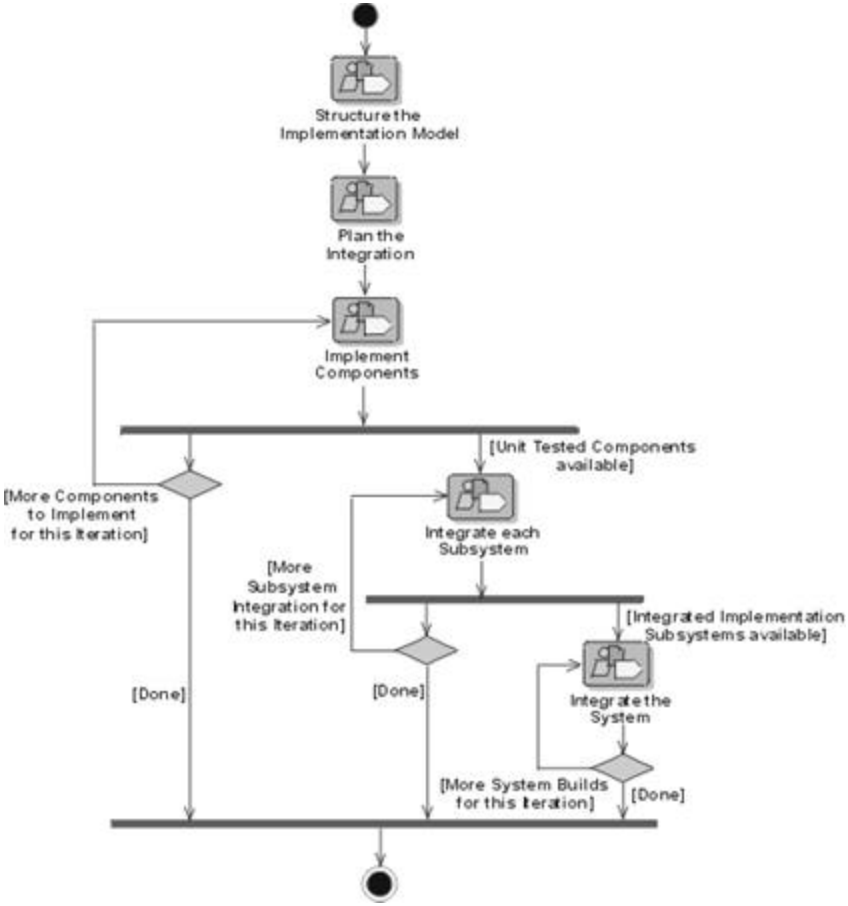


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Implementação

Propósito:

- Definir a organização dos artefatos de código.
  - São organizados em camadas ou subsistemas de implementação.
- Implementar classes e objetos em termos de componentes.
  - Código fonte, binários, executáveis...
- Testar componentes.
  - Testes de unidade.
- Integrar componentes.

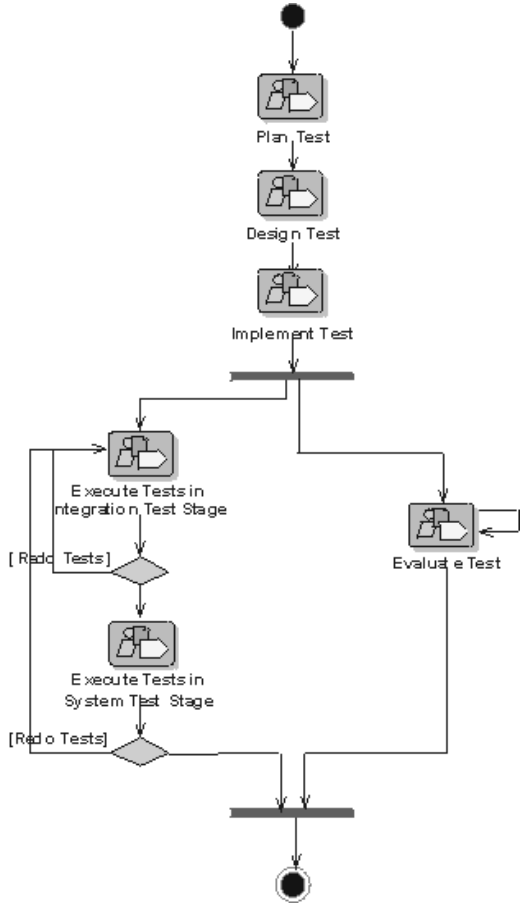


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Testes

Propósito:

- Verificar a interação entre os objetos.
- Verificar a integração entre os diversos componentes.
- Verificar se todos os requisitos foram implementados corretamente.
- Identificar e garantir que todos os defeitos foram consertados.

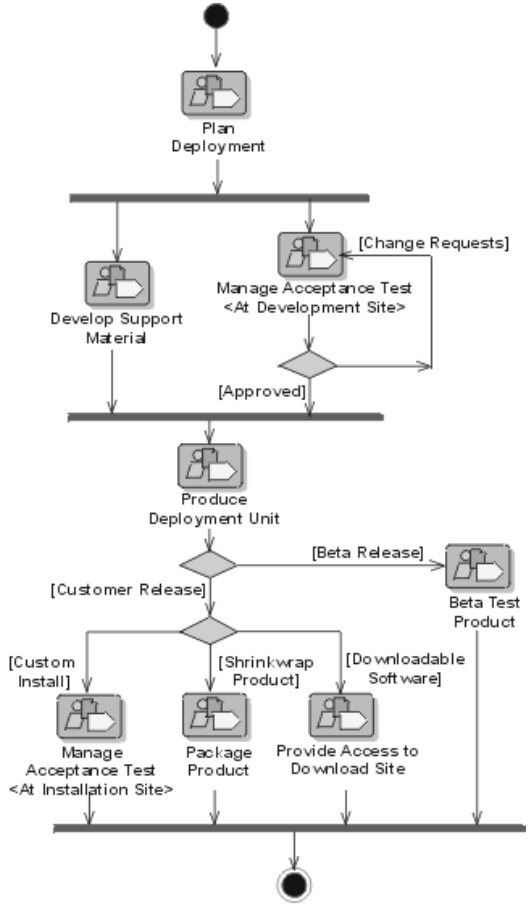


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Deployment

Propósito:

- Descrever as atividades associadas a entrega do produto ao usuário.
- Executar teste beta antes do produto ser entregue.
- Objetiva produzir versões de produtos (*releases*).

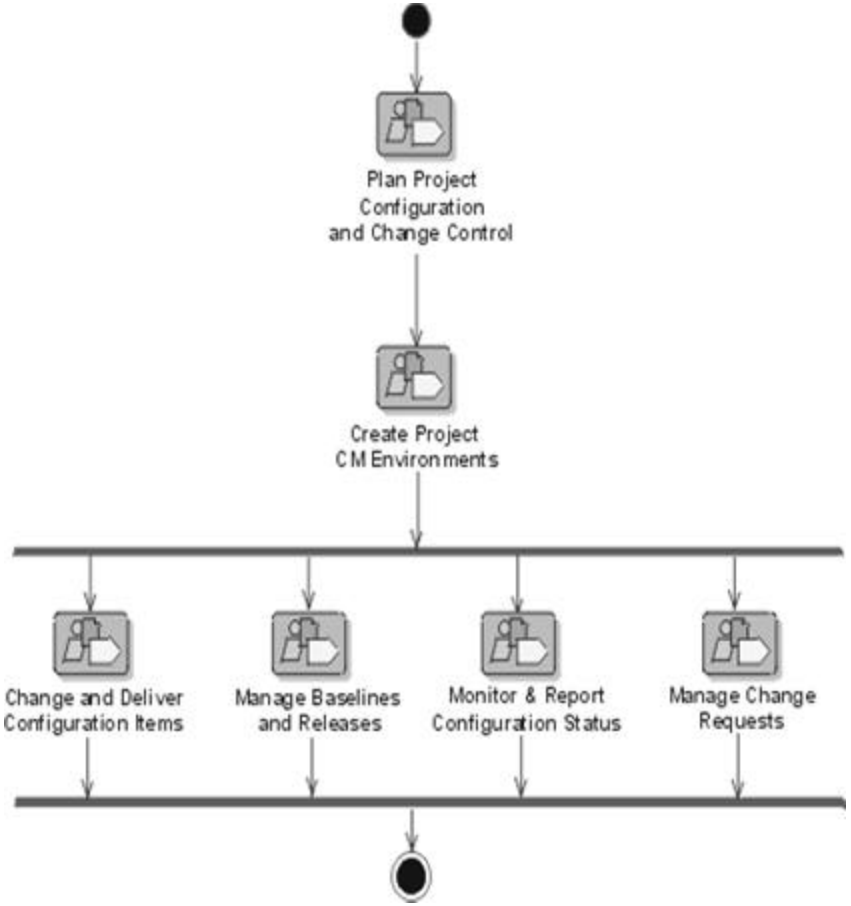


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Gerenciamento de configuração e mudanças

Propósito:

- Descrever como controlar os inúmeros artefatos produzidos por diversas pessoas que trabalham em um mesmo projeto.
- Evitar problemas tais como:
  - Atualizações simultâneas
  - Notificações de alterações
  - Múltiplas versões



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Gerenciamento do projeto

Propósito:

- Prover guias para planejamento, alocação de recursos, execução e monitoramento.
- Prover guias para o gerenciamento de riscos.

Arte de balancear:

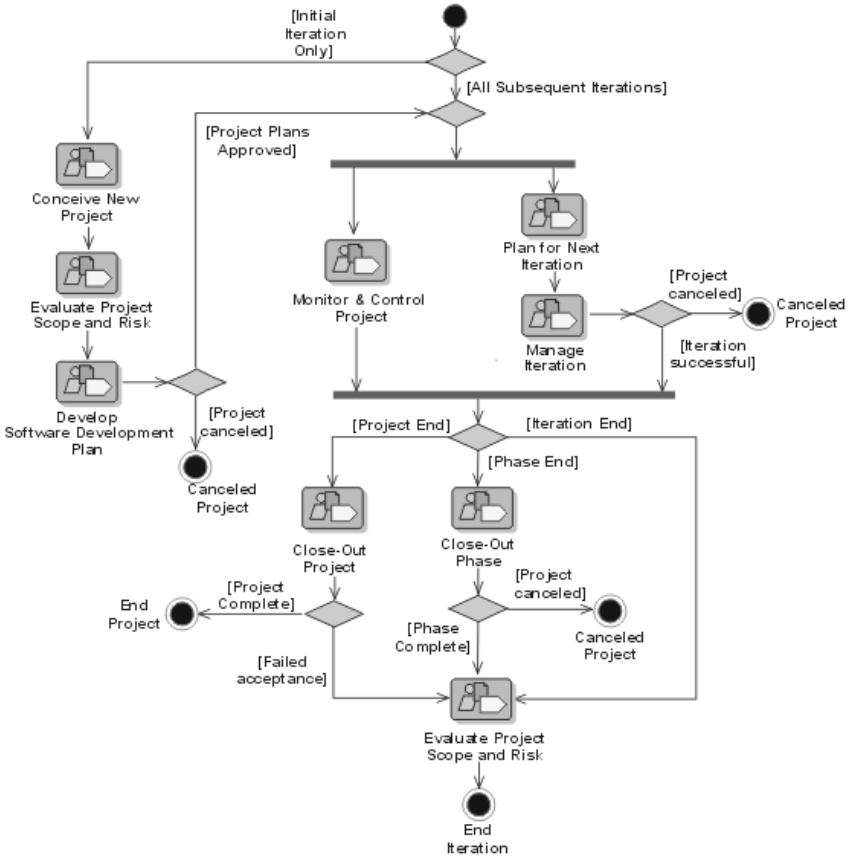
- Objetivos competitivos e conflitantes.
- Riscos.
- Necessidades do usuário.

Enfatizar:

- Gerenciamento de riscos.
- Definir plano de iteração.
- Monitorar o processo.

Não cobre as seguintes questões:

- Gerencia de pessoas.
- Orçamento.
- Contratos.

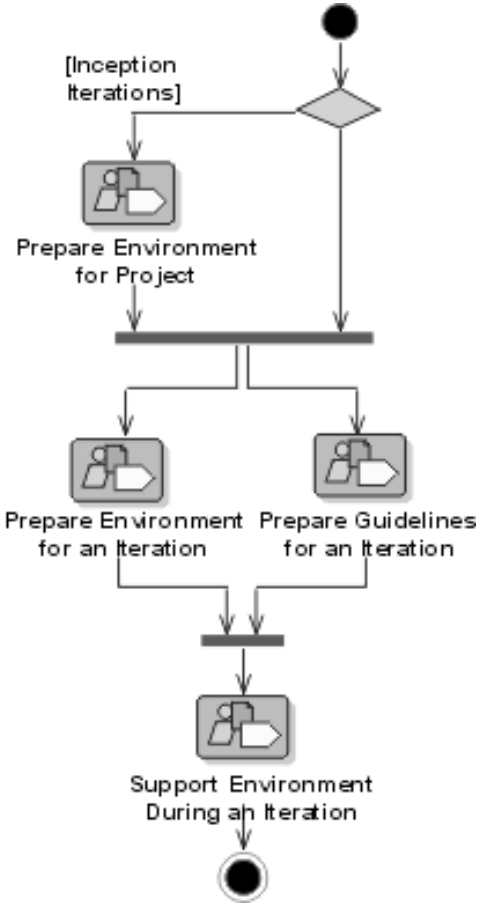


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Ambiente

Propósito:

- Dar enfoque às atividades necessárias para a configuração do processo e ferramentas de suporte ao desenvolvimento.
- Descrever atividades necessárias para a criação de guias.
- Prover ferramentas e processos que dêem suporte à equipe de desenvolvimento.

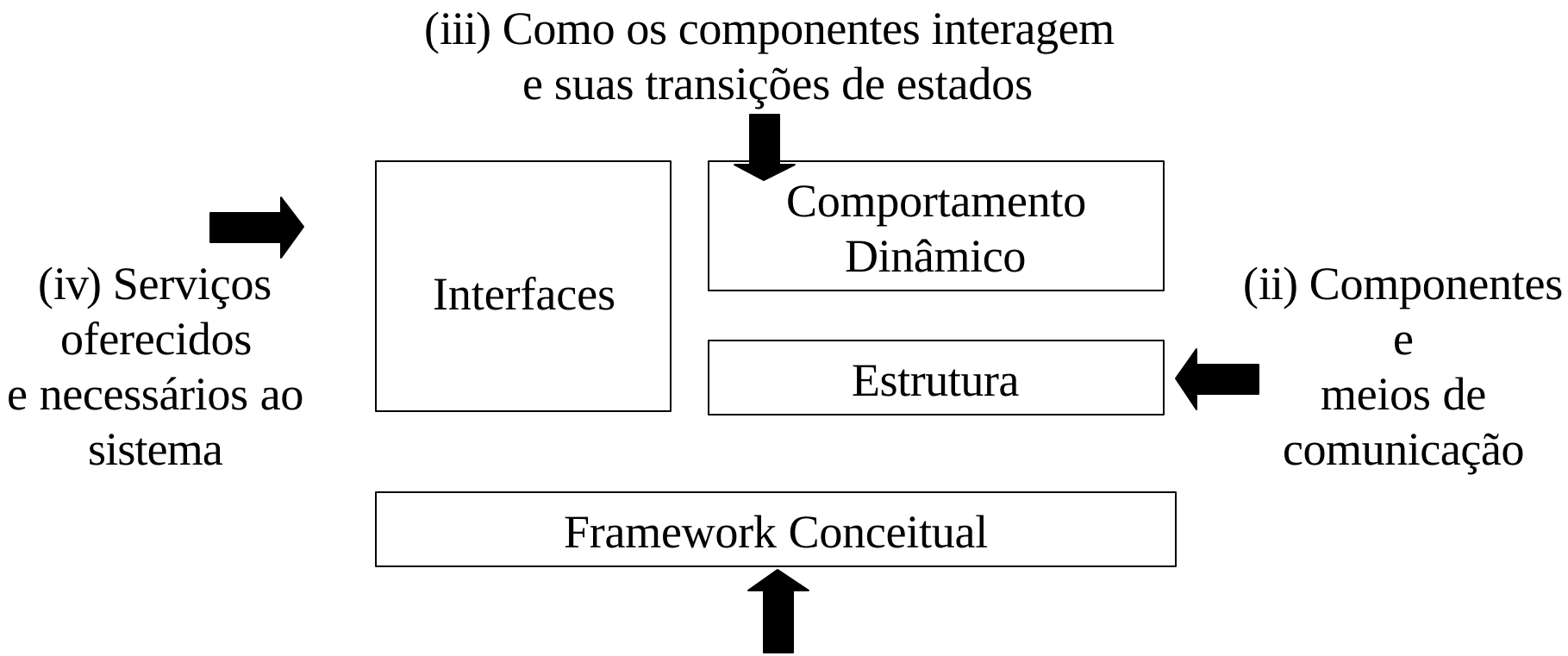


Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Introdução à UML

Gustavo Robichez de Carvalho
Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Elementos de Arquitetura de Software



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

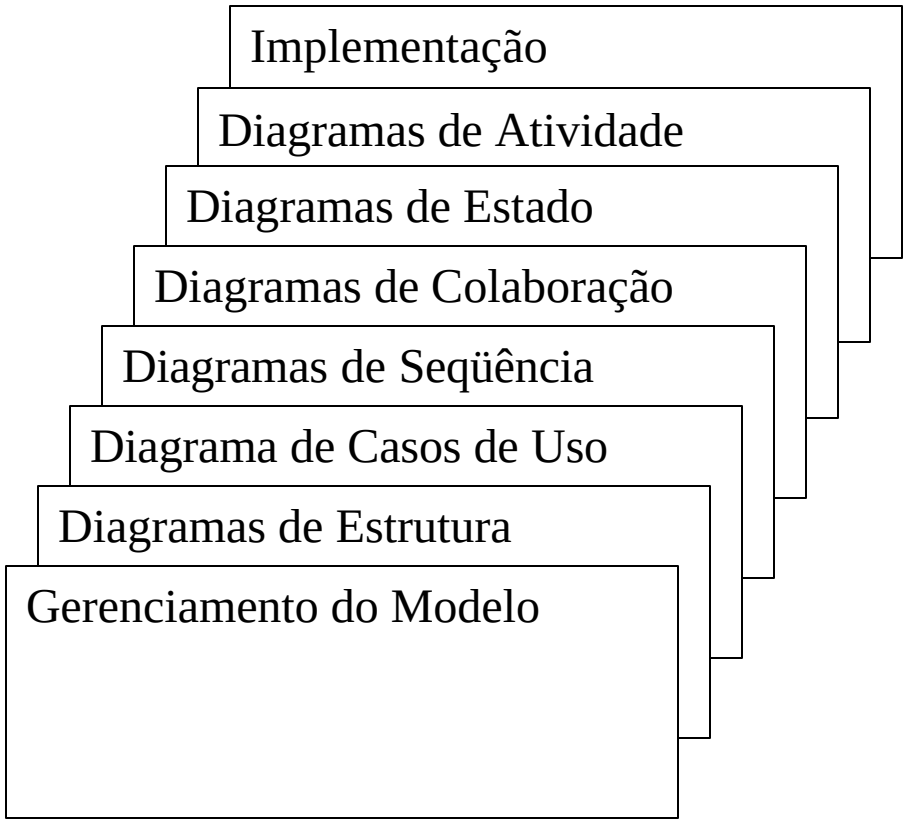
UML – *Unified Modeling Language*

Linguagem de modelagem padrão OMG para o desenvolvimento de sistemas OO

Características

- Modelagem Visual de sistemas de software, i.e.
  - Gráfica e operacional
  - Ao invés de textual e declarativa
- Combina notação de métodos existentes
  - Booch, OMT e OOSE (Jacobson)
  - Influenciado por Fusion, Odell, etc
- Não é um método (nem é específico a um método)
  - i.e. sem processo definido, guidelines...

UML é grande (e complexo)



Object Constraint Language (OCL)

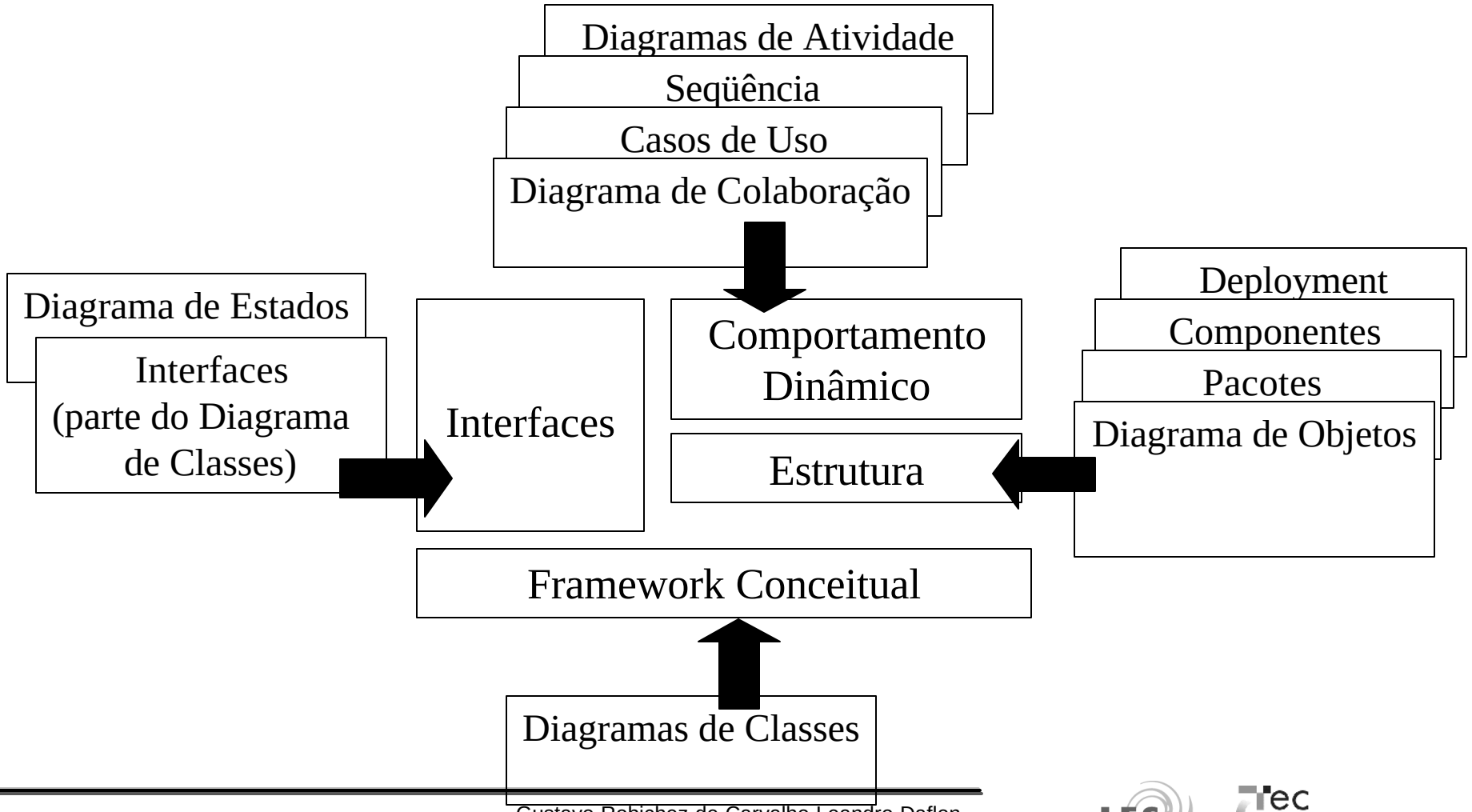
Mecanismos de Extensão

Geral

Possui 8 categorias de notações gráficas, mais OCL (Object Constraint Language) e mecanismos de extensão

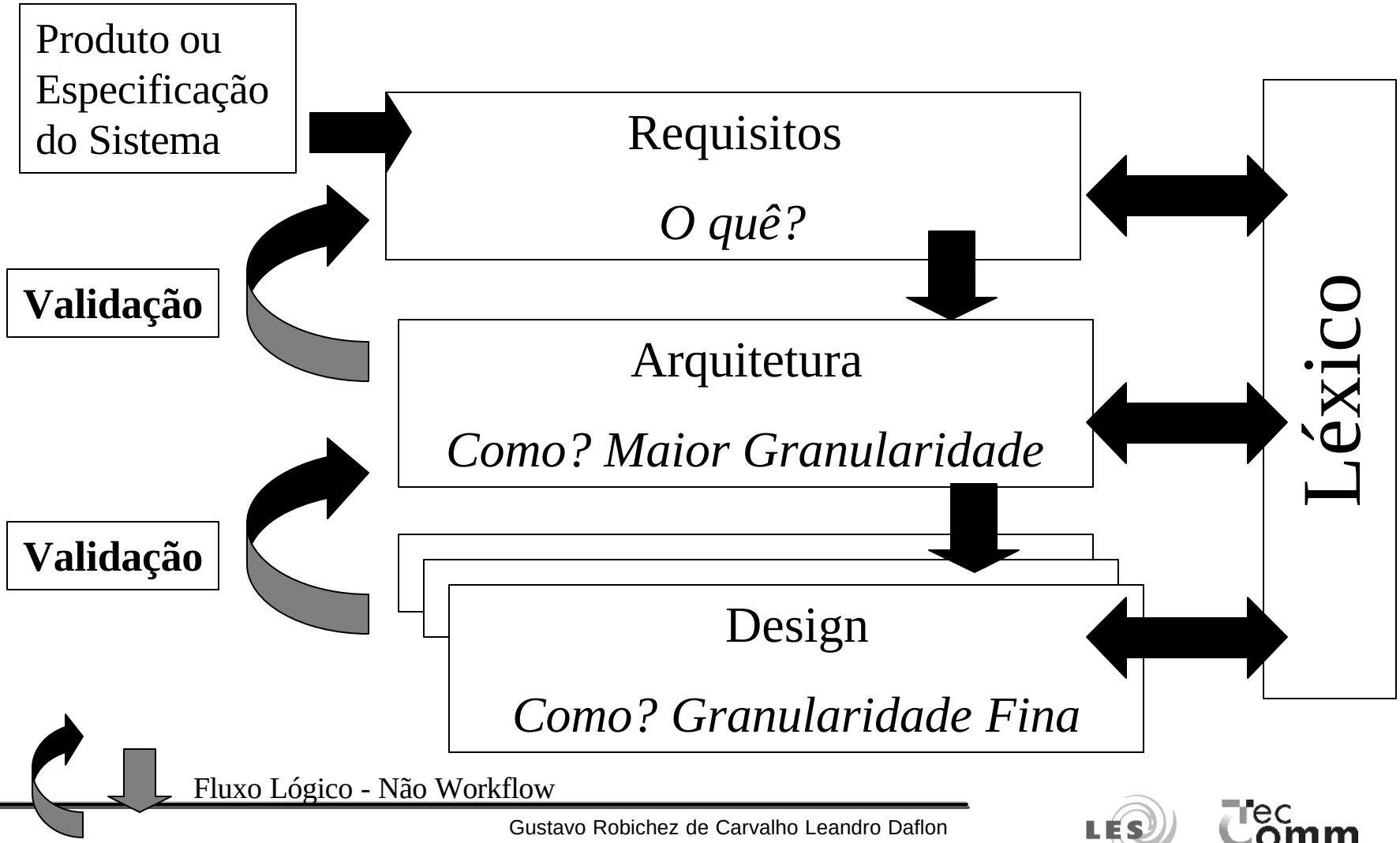
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Para que servem as notações de UML



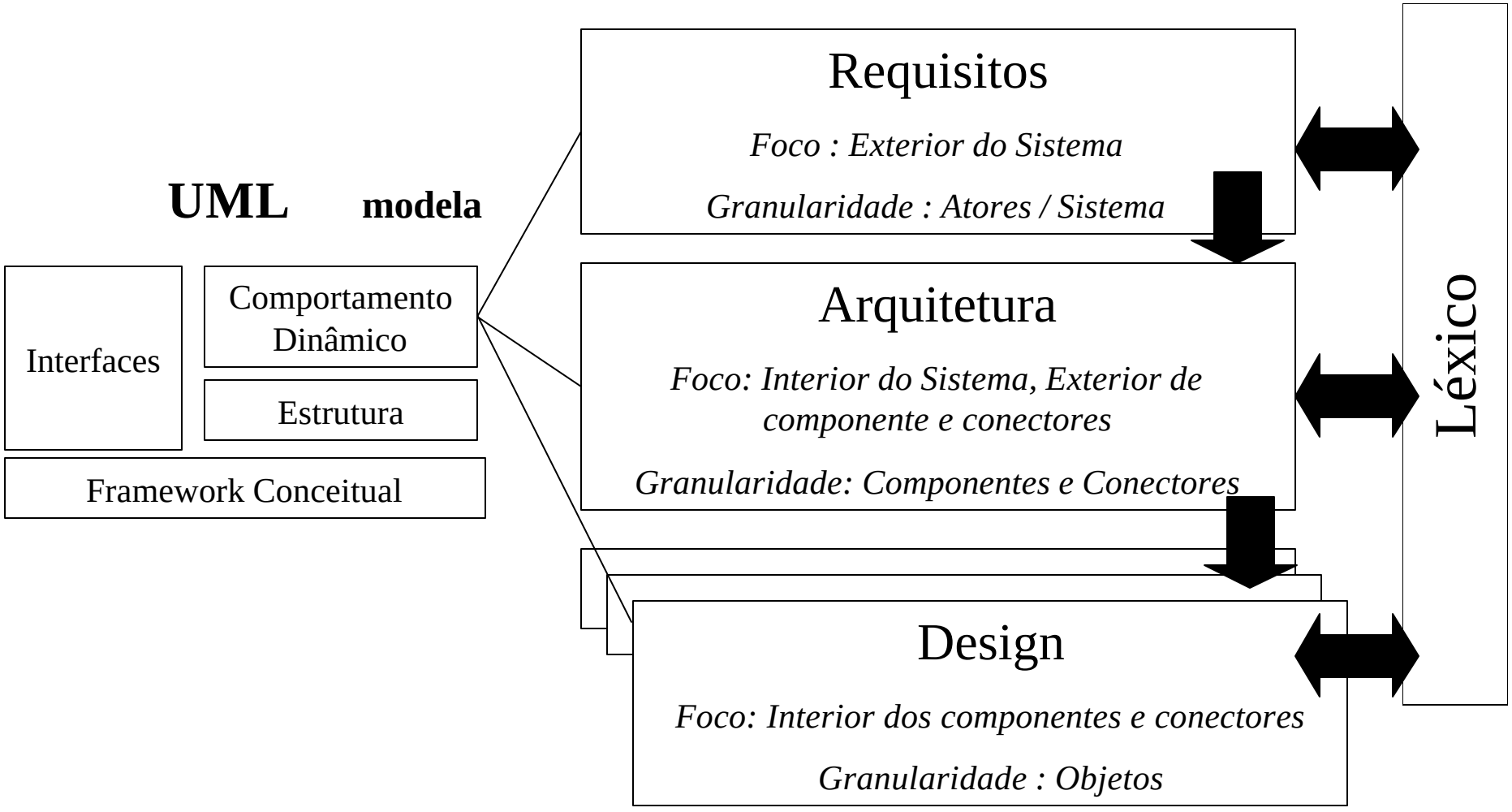
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Processo de Desenvolvimento Típico



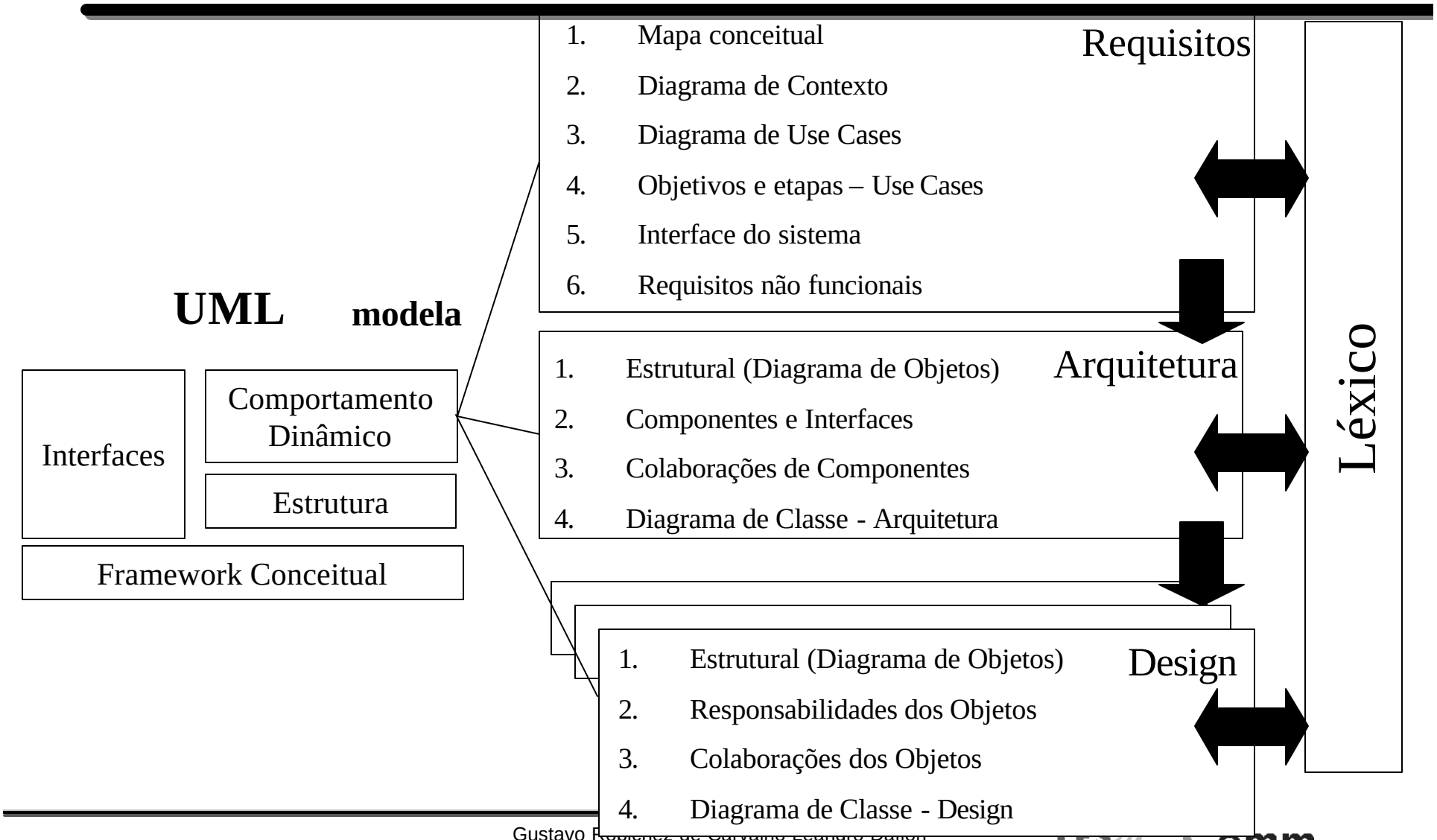
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

UML e Processo de Desenvolvimento



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Artefatos



Gustavo Robichez de Carvalho Leandro Dalton
{guga, daflon}@les.inf.puc-rio.br

Introdução a orientação a objetos

Gustavo Robichez de Carvalho
Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Complexidade

Sistemas cada vez mais complexos

- Tamanho, novas plataformas, novas tecnologias.
- Facilidade de uso implica em complexidade de construção
- Complexidade do domínio da aplicação
- Dependência das organizações
- Complexidade do processo de desenvolvimento
  - Gerência de atividades e documentos
  - Necessidade de equipes

Complexidade

Estratégias para se administrar a complexidade

- Decomposição (Modularidade)
- Abstração
- Encapsulamento
- Classificação (Hierarquias)

Decomposição

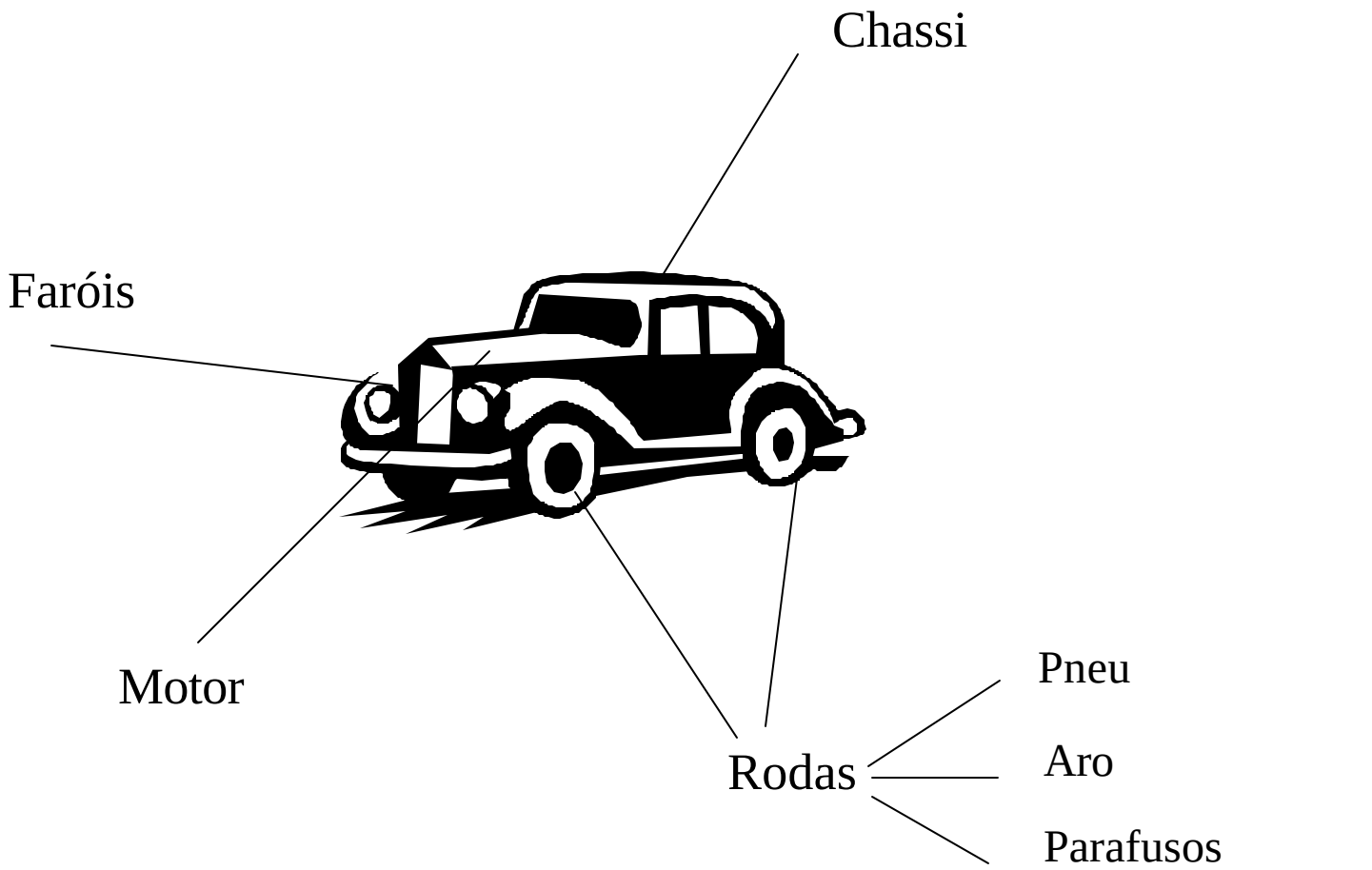
"Dividir para conquistar"

Decomposição sucessiva de problemas em subproblemas de mais fácil tratamento

A decomposição pode ser feita sob o ponto de vista

- de um elemento e suas partes
- de um procedimento e suas etapas

Decomposição



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Cenário Exemplo

"João deseja enviar flores para Maria mas ela mora em outra cidade. João vai, então, até a floricultura e pede a José, o floricultor, para que ele envie um *bouquet* de rosas ao endereço de Maria. José, por sua vez, liga para uma outra floricultura, da cidade de Maria, e pede para que as flores sejam entregues."

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Nomenclatura

João precisa resolver um problema;

Então, ele procura um *agente*, José, e lhe passa uma *mensagem* contendo sua *requisição*: enviar rosas para Maria;

José tem a *responsabilidade* de, através de algum *método*, cumprir a *requisição*;

O *método* utilizado por José pode estar *oculto* de João.

- José só disponibiliza uma interface

Modelo OO

Uma ação se inicia através do envio de uma *mensagem* para um *agente* (um *objeto*) responsável por essa ação.

A mensagem carrega uma *requisição*, além de toda a informação necessária (*argumentos*) para que a ação seja executada.

Quando o *agente receptor* da *mensagem* a aceita, ele tem a *responsabilidade* de executar um *método* para cumprir a *requisição*.

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Objetos

Estão preparados para cumprir um determinado conjunto de requisições.

Recebem essas requisições através de mensagens.

Possuem a responsabilidade de executar um método que cumpra a requisição.

Possuem um *estado* representado por informações internas.

Classes

O conjunto de requisições que um objeto pode cumprir é determinado pela sua classe.

A classe também determina que método será executado para cumprir uma requisição.

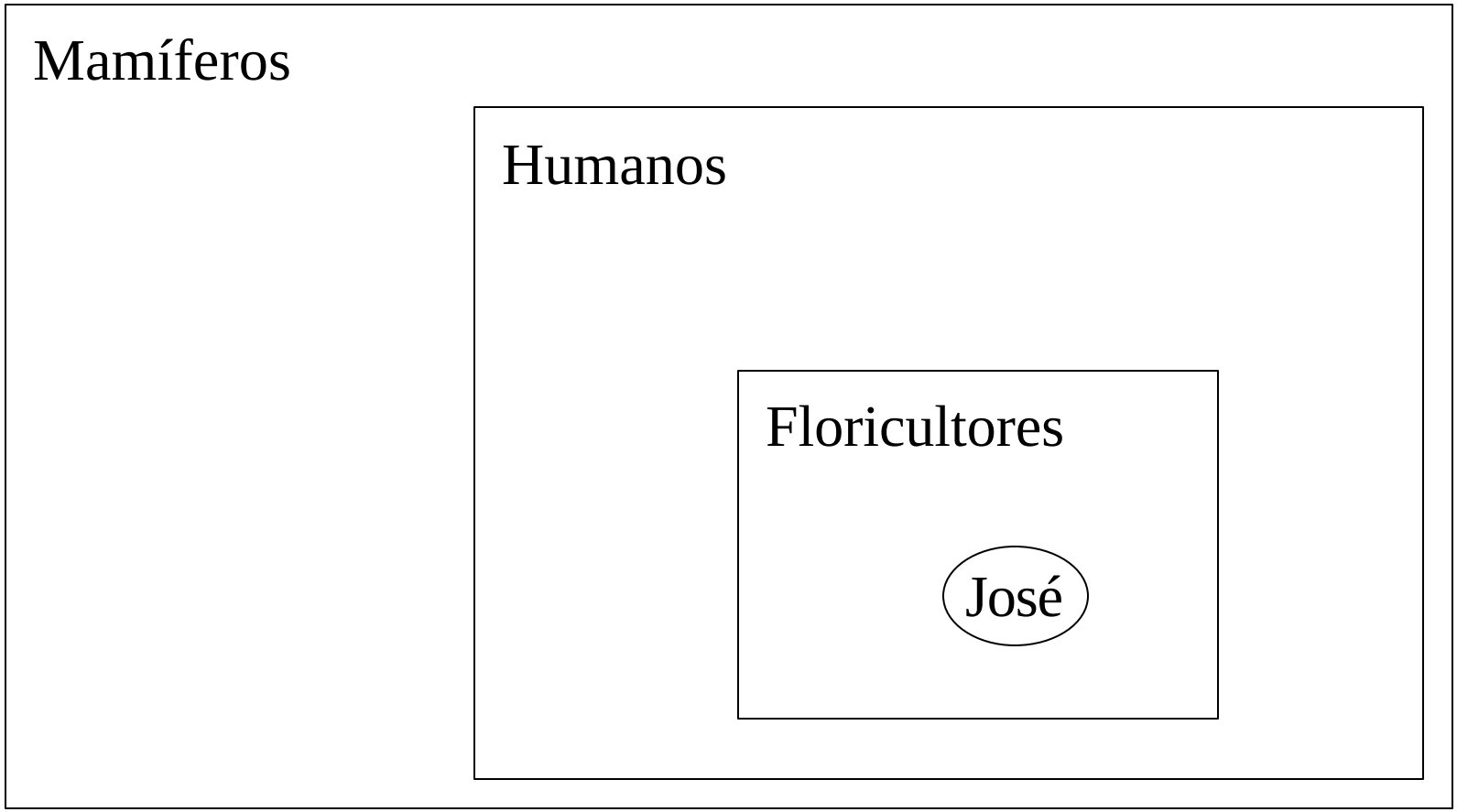
A classe especifica que informações(*estados*) um objeto armazena internamente.

Objetos são *instâncias* de classes.

Classes podem ser compostas em hierarquias, através de *herança*.

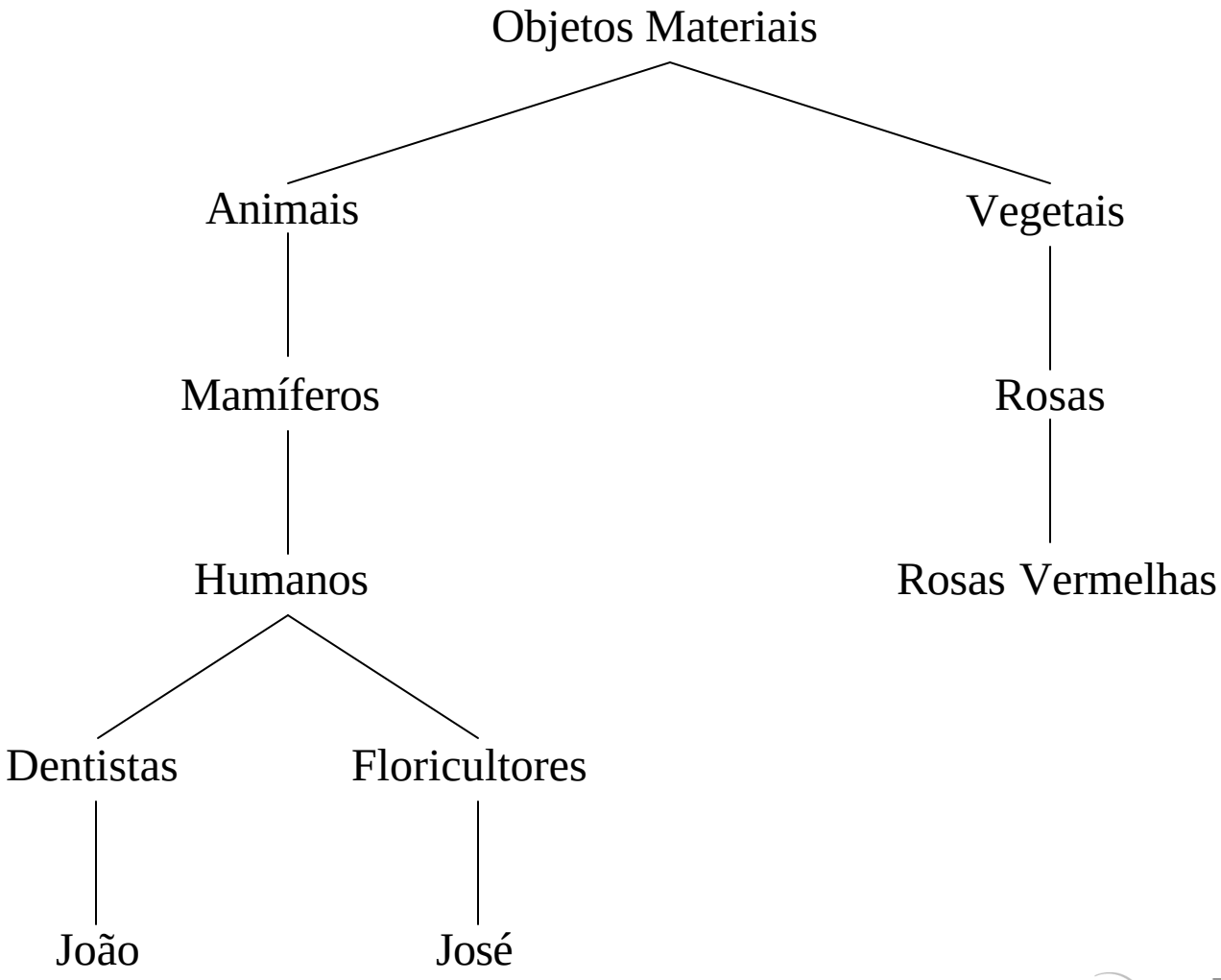
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Hierarquia de Classes (Herança)



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Hierarquia de Classes (Herança)



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Resumo

Agentes são objetos;

Ações (computações) são executadas através da troca de mensagens entre objetos;

Todo objeto é uma instância de uma classe;

Uma classe define uma interface e um comportamento;

Classes podem estender outras classes através de herança.

Orientação a objetos

Objetos ou Instâncias

Métodos ou Mensagens

Encapsulamento

Classes

Variáveis da Classe X Variáveis da Instância

Métodos da Classe X Métodos da Instância

Relacionamentos

Identificando Objetos

Classes Abstratas

Polimorfismo

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

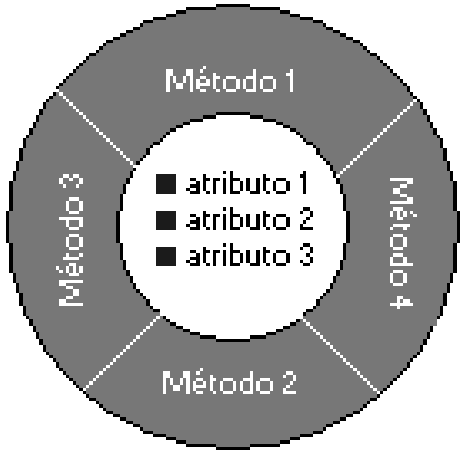
Objetos ou Instâncias I

Todo objeto possui:

- Estado
- Comportamento

Todo objeto de software:

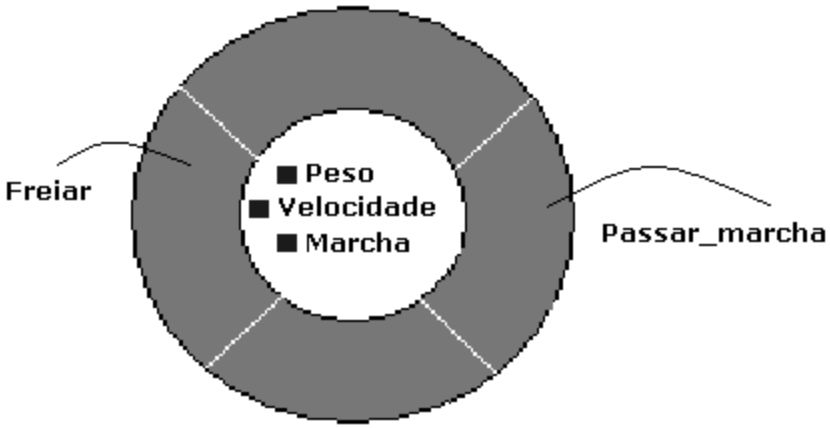
- mantém seu estado em variáveis
- implementam seus comportamentos com métodos



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Objetos ou Instâncias II

Exemplo: bicicleta



O objeto tem total controle sobre o acesso a seus métodos e suas variáveis

O objeto pode disponibilizar algumas variáveis e métodos e esconder outras variáveis e métodos

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

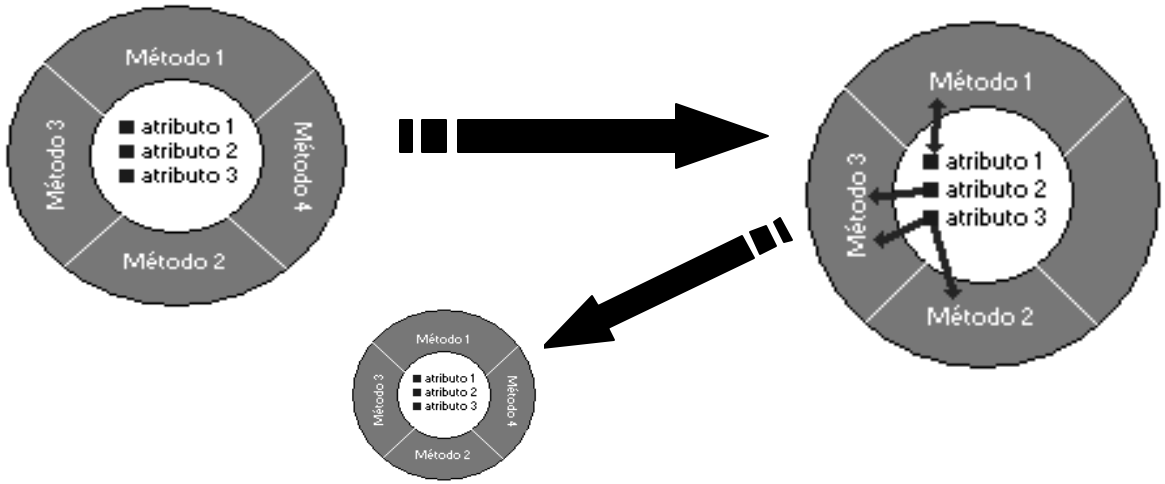
Métodos ou Mensagens I

Um método pode ou não retornar um outro objeto

- Pode alterar ou verificar o estado (atributos) de um objeto

Execução de um método é a reação à recepção de uma mensagem

Deve operar com os atributos do próprio objeto e atributos recebidos como parâmetro



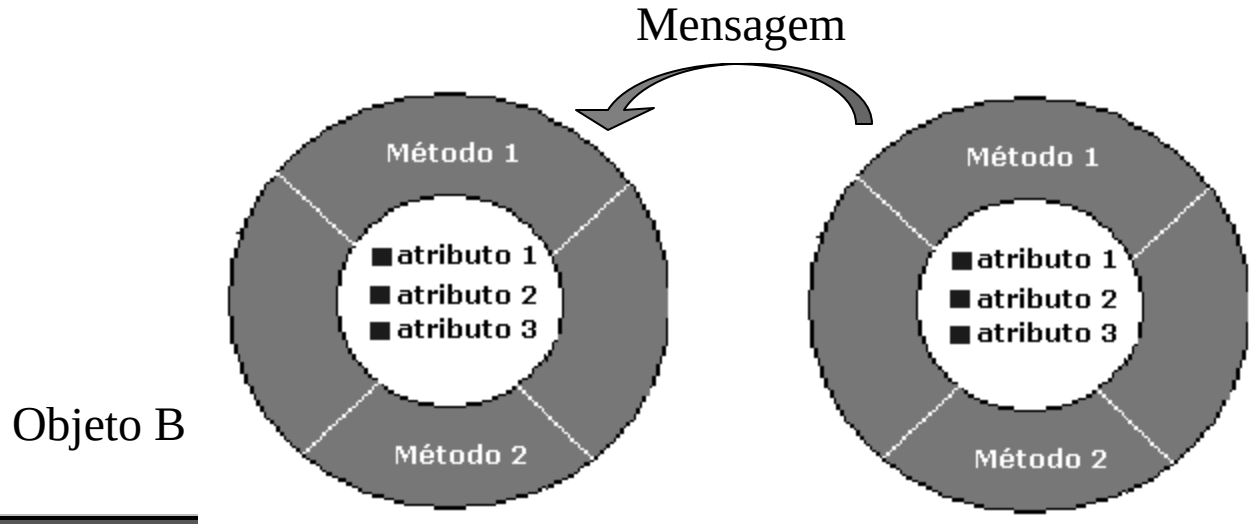
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Métodos ou Mensagens II

Objetos de software interagem e se comunicam uns com os outros através do envio de mensagem

Quando um objeto A quer que o objeto B faça uma ação, o objeto A envia uma mensagem ao objeto B

"A pede para B fazer Ação" = "A pede para B executar Método X"



Objeto B

Objeto A

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

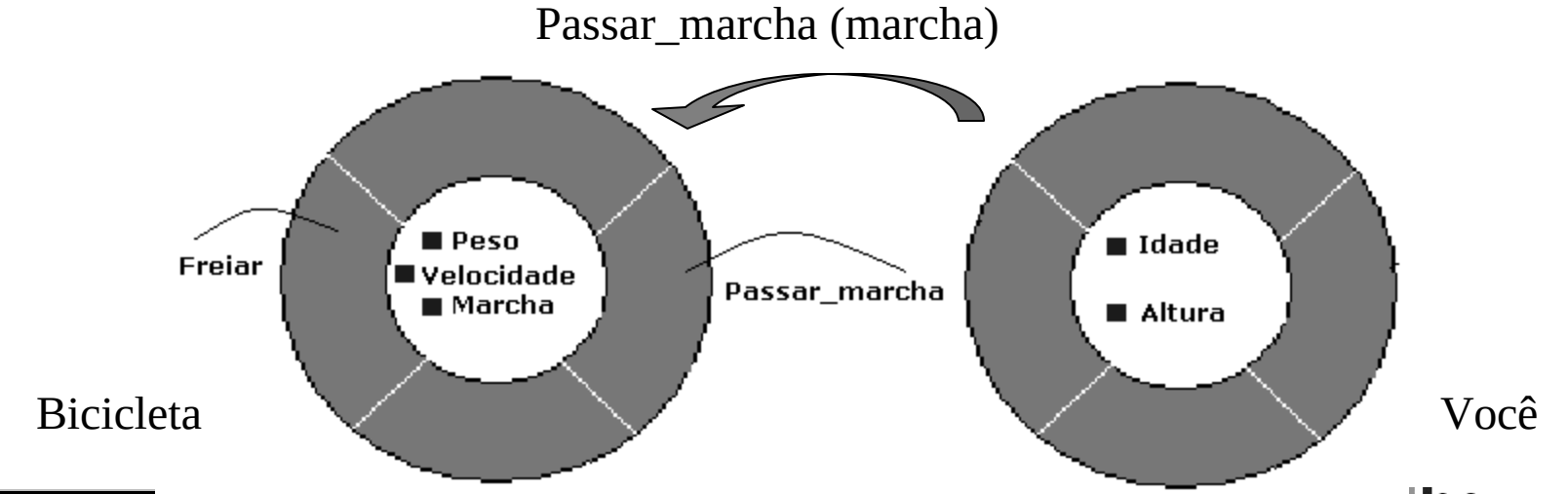
Métodos ou Mensagens III

Três componentes compreendem uma mensagem:

- o objeto para o qual a mensagem é enviada
- o nome do método a ser executado
- parâmetros necessários para execução do método

"A pede para B fazer Ação" =

"Você pede para a Bicicleta Passar_Marcha(marcha)"



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

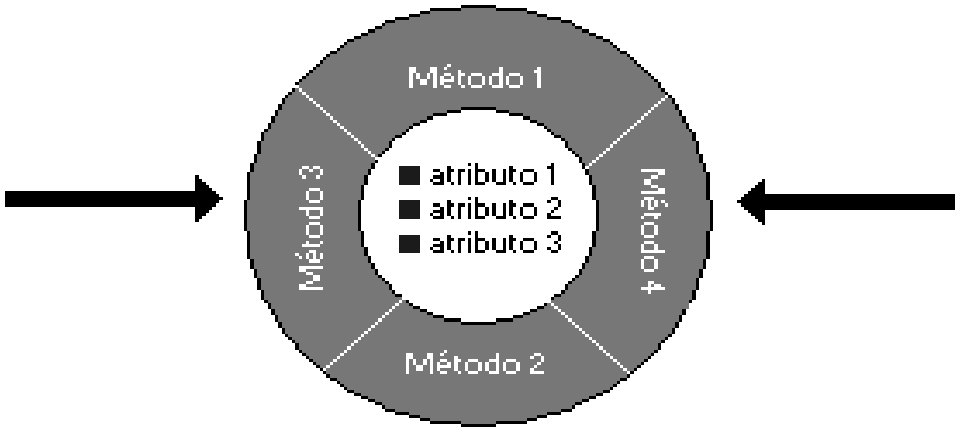
Encapsulamento

Usado para esconder detalhes de implementação

Não é necessário saber como a classe está implementada para chamar o método necessário.

Basta saber a interface do método

Os atributos de um objeto só devem ser manipulados pelos métodos do próprio objeto (orientação a objeto pura).



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Restrição de acesso (métodos e atributos)

Public: sem restrição. Qualquer objeto pode acessar tal método/atributo

Private: apenas o objeto que possui o método/atributo pode acessá-lo

- OBS: se usado em um atributo de uma super-classe, suas sub-classes não pode manipular. Para que a sub-classe possa manipular tal atributo é necessário usar o *protected*

Protected: apenas os objetos das classes do mesmo pacote podem acessar o método/atributo

- OBS: caso seja usado em um atributo de uma super-classe, as sub-classes não necessitam estar no mesmo pacote (exceção a regra)

Default de uso:

- atributo -> *private* (encapsulamento)
- método -> *public*
- Classe -> *public* (sem o *public* a classe só é visível no pacote)

Se em Java não for especificada a restrição de acesso o default é:

- atributo, método, classe -> protected

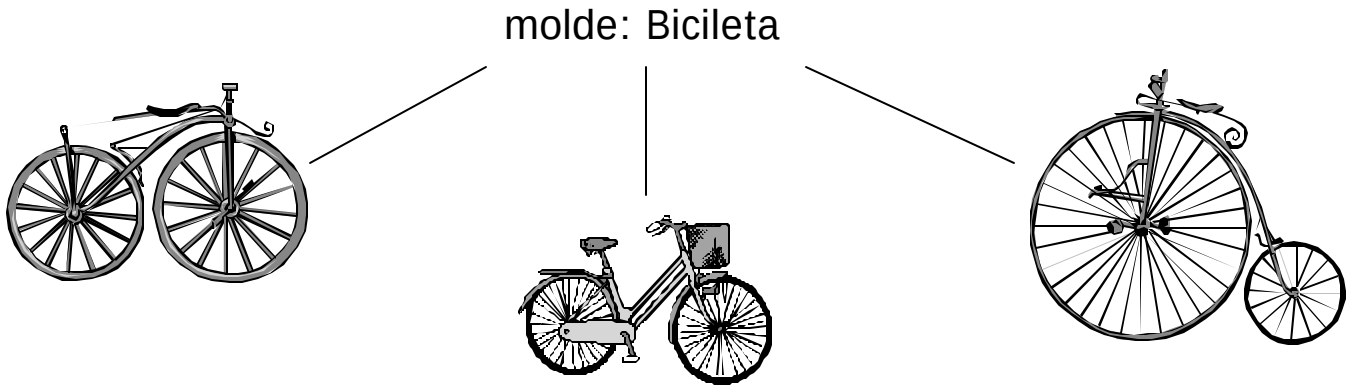
Classe

No mundo real sua bicicleta é apenas uma das existentes no mundo

As bicicletas possuem estado e comportamento em comum

“Molde” para construção da bicicleta: classe

Terminologia OO: sua bicicleta é uma instância da classe *Bicicleta*.



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Classe X Instância I

Para criar o objeto bicicleta no mundo OO é necessário:

- criar uma classe de bicicletas (molde)
- instanciar um objeto a partir da classe criada.

Com um molde você gera vários objetos parecidos. Todos os objetos possuem os mesmos atributos e os mesmos métodos (implementação dos métodos é a mesma para todos os objetos)

Os valores dos atributos podem ser modificados por cada objeto

Quando uma instância da classe é gerada, um objeto é criado e o sistema *aloca* memória para suas variáveis

Classe X Instância II

Pássaro

Características:	*Comportamento*:
cor das penas	voar
formato do bico	piar
velocidade de vôo	

Identidade: 'Beija-flor Biju'

Características:

cor das penas: azuis
formato do bico: fino
velocidade de vôo: rápida

Comportamento:

voar
piar

Identidade: 'Minha pomba'

Características:

cor das penas: cinza
formato do bico: curto
velocidade de vôo: média

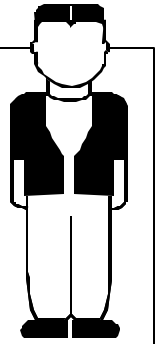
Comportamento:

voar
piar

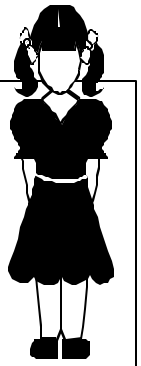
Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Exemplo

Classe: Empregado
atributos:
nome
endereço
salário = R$ 300,00
métodos:
atualizarInfo(nome, end, salário)
forneceInformações()



Objeto: Empregado A
atributos:
nome = João
endereço = R. Maria 12
salário= R$ 300,00
métodos:
atualizarInfo(nome, end, salário)
fornecerInformações()



Objeto: Empregado B
atributos:
nome = Ana
endereço = R. D. José 30
salário= R$ 300,00
métodos:
atualizarInfo(nome, end, salário)
fornecerInformações()

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Variáveis da Instância X da Classe

Variáveis da Classe (*static*)

Só existe uma cópia de cada variável
Todos as instâncias acessam a mesma cópia das variáveis
São criadas apenas uma vez assim que o sistema reconhece a classe

Variáveis da Instância

Existe uma cópia para cada instância criada
Cada instância acessa a sua cópia
São criadas sempre quando uma nova instância é gerada

Métodos de Instância X de Classe

Métodos de Classe (*static*)

Podem acessar apenas variáveis da classe
Podem ser acessados pela classe ou pela instância

Métodos de Instância

Podem acessar variáveis da classe e da instância
Só podem ser acessados por instâncias

Classe abstrata

Pelo menos um de seus métodos está *declarado* mas não têm *implementação* associada.

- Método *abstrato*: método sem implementação

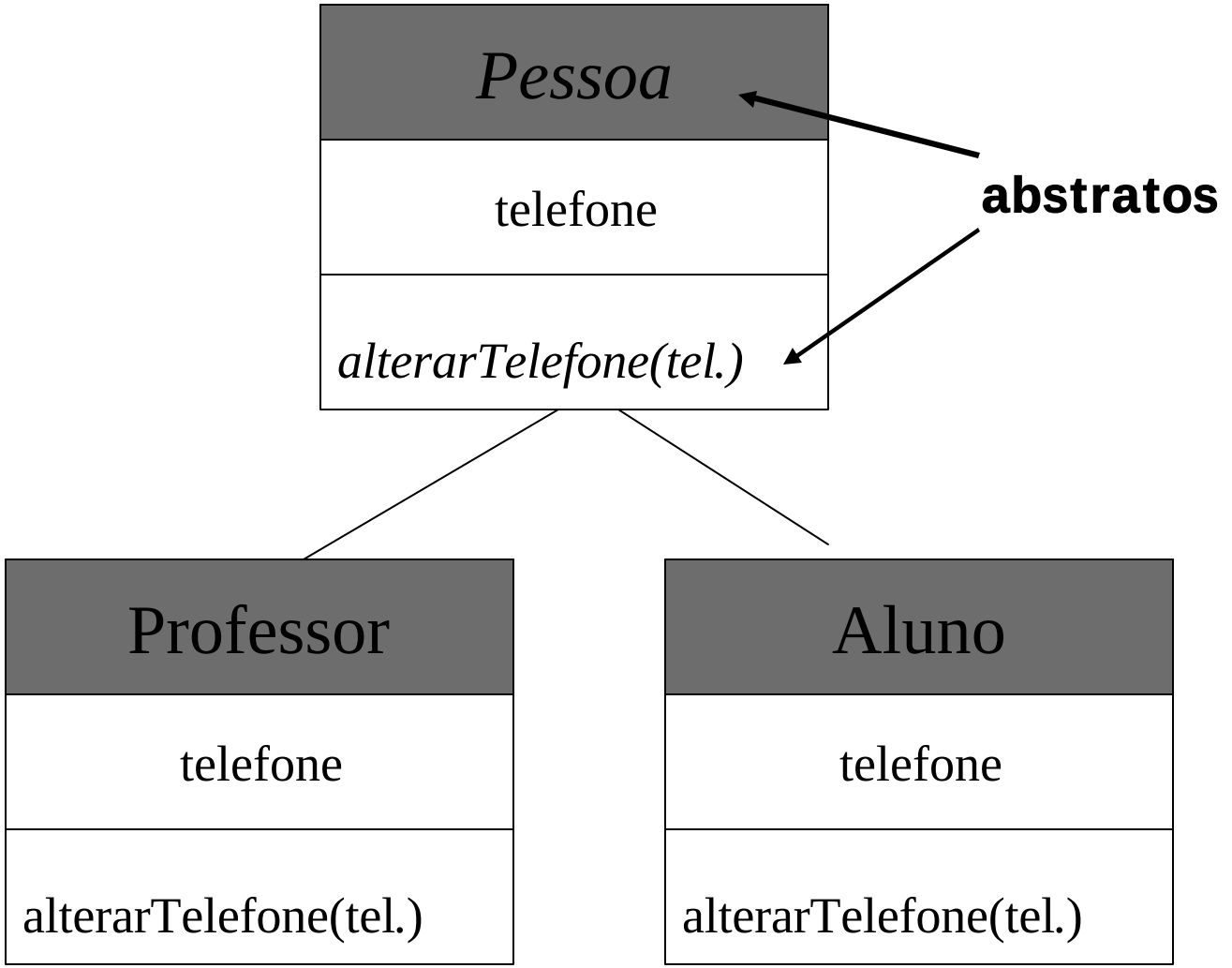
Não gera instância

Só pode ser usada como base para outras classes

- herança

Cada subclasse deverá implementar o método abstrato da superclasse (classe abstrata)

Exemplo



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Polimorfismo

Polimorfismo é a capacidade de um objeto tomar diversas formas.

A capacidade polimórfica decorre diretamente do mecanismo de herança.

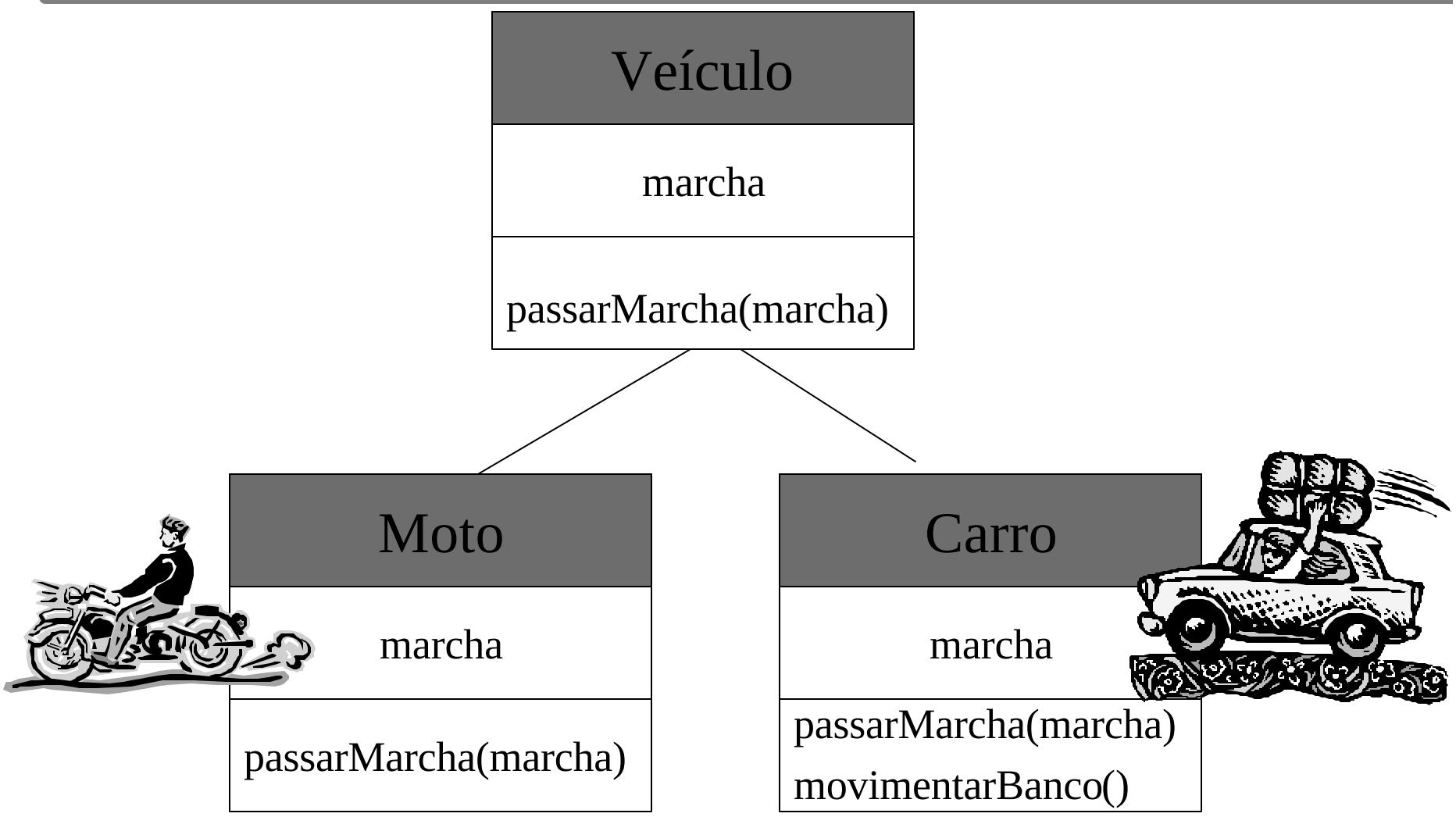
Ao estendermos ou especializarmos uma classe, não perdemos compatibilidade com a superclasse.

A sub-classe de **Pessoa**, **Aluno**, é compatível com ela, ou seja, um aluno, além de outras coisas, *é* uma pessoa.

Isso implica que, sempre que precisarmos de uma pessoa, podemos usar um aluno em seu lugar. Mas o contrário não é verdade.

Polimorfismo é o nome formal para o fato de que quando precisamos de um objeto de determinado tipo, podemos usar uma versão mais especializada dele. Esse fato pode ser bem entendido analisando-se a árvore de hierarquia de classes.

Exemplo



Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Herança Múltipla

Java não permite herança múltipla com herança de código.

Java implementa o conceito de *interface*.

É possível implementar múltiplas interfaces.

Em Java, uma classe *estende* uma outra classe e *implementa* zero ou mais interfaces.

Para implementar uma interface em uma classe, usamos a palavra **implements**.

Membros da Interface

Uma vez que uma interface não possui implementação, devemos notar que:

- suas variáveis devem ser públicas, estáticas(variável de classe) e constantes (não podem ser alteradas);
- seus métodos devem ser públicos e abstratos(devem ser implementados).

Como esses qualificadores são fixos, não precisamos declará-los.

```
interface Stack {
  boolean isEmpty();
  void push(int n);
  int pop();
  int top();
}
```

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Interface

```
class StackImpl implements Stack {
  private int[] data;
  private int top_index;
  StackImpl(int size) {
    data = new int[size];
    top_index = -1;
  }
  public boolean isEmpty() { return (top_index < 0); }
  public void push(int n) { data[++top_index] = n; }
  public int pop() { return data[top_index--]; }
  public int top() { return data[top_index]; }
}
```

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Relacionamentos entre Classes

Para que objetos se comuniquem eles precisam se relacionar

Tipos de Relacionamentos:

- Associação
  - Agregação
    - Composição
  - Dependência
- Generalização / Herança

Relacionamentos (continuação)

Associação:

- descreve uma relação entre duas classes
  - Usuário possui bicicleta

Agregação:

- descreve o relacionamento entre um todo e sua parte
- são indicadas por frases do tipo “tem um”, “é parte de”
  - Uma teclado é parte de um *notebook*

Relacionamentos (continuação)

Generalização / Herança

- descreve o relacionamento entre classes definidas a partir de outras classes.
- toda a subclasse herda os atributos e os comportamentos definidos na superclasse
- as subclasses não estão limitadas a estes estados e comportamentos.
  - Uma *mountain bike* é uma bicicleta

subclasse superclasse

Gustavo Robichez de Carvalho Leandro Daflon
{guga, daflon}@les.inf.puc-rio.br

Identificando Objetos

Em uma sala existe um conjunto de objetos físicos que podem ser facilmente identificados, modelados e classificados como objetos OO.

Mas em um problema onde o espaço é uma aplicação de software, os objetos podem não ser facilmente encontrados.

Os objetos podem ser identificados analisando-se o problema ou fazendo um "*parser* gramatical" do texto contendo a descrição do problema

Objetos são determinados sublinhando-se cada substantivo ou oração (parte de uma frase)

Identificando Objetos (continuação)

Candidatos a objeto:

- *Entidade externa* que produz ou consome informação para ser usada por um sistema computacional (ex.: outros sistemas, *devices*, pessoas)
- *Coisas* que são parte do domínio da informação do problema (ex.: um sinal, uma carta, um display)
- *Ocorrências ou eventos* que acontecem no contexto da operação do sistema (ex.: a propriedade de uma transferência, a finalização de uma série de movimentos de um robô)
- *Papeis(roles)* desempenhados por pessoas que interagem com o sistema (ex.: gerente, engenheiro, vendedor)
- *Unidades de organização* que são relevantes a uma aplicação (ex.: divisões, grupo, time)
- *Lugares* que estabelecem o contexto do problema (ex.: galpão, estaleiro)
- *Estruturas* que definem a classe de um objeto (ex.: sensor, computador, veículo 4-rodas)