INF319 — Projeto e Implementação Orientados a Objetos

L.E. Busato

IC, UNICAMP

Roteiro

1. Engenharia de Software
2. Orientação a Objetos
3. Projeto Orientado a Objetos

Aplicação (Programa)

Definições

Uma **aplicação** é um programa (software) projetado e implementado para resolver um problema pertencente a um domínio de conhecimento.
Um **domínio de conhecimento** é um conjunto delimitado de conhecimentos e/ou processos.

Exemplos de domínios e problemas

Domínio	Problema(s)
Transporte	Calc. rota ótima, sw veículos autônomos
Comércio	warehouse management system (intra, inter)
Gestão Acadêmica	Horário e Alocação de salas
Gestão Hospitalar	Gestão de sala cirurgica
Finanças	Avaliação de portfolio
Robótica	Visão, Movimento, etc

A engenharia de software é um conjunto de conhecimentos e práticas que organiza e conduz a construção de aplicações para um domínio.

Qual é o conceito mais importante para um engenheiro de software?

Abstração

Abstração

Abstrações procuram capturar, na visão de quem abstrai, atributos essenciais de uma realidade, de um problema. Podem ser organizadas hierarquicamente e são **domínio dependentes**. A definição de uma abstração depende do domínio e contexto em que é criada, logo pessoas diferentes frequentemente constroem abstrações diferentes para uma mesma realidade (domínio).

Abstração

Algo que existe somente como idéia.

Processo mental que constrói um mapeamento um-para-muitos.

Abstração	Objeto, Processo (Domínio do Problema)
Veículo	Carro, Avião, Trem, Navio,...
Carro	atributos relevantes comuns aos carros
Corpo Humano	Sistemas Nervoso (SN), Circulatório, Respiratório, ...
SN	SN Central, SN Periférico
SN Central	Cérebro, Medula espinhal
Folha Pgto	Processo de pagamento (função, tempo, valor, contrato)
Sist. Hidrológico	Chuvas, Evaporação, Absorção, ...

Problema

Pessoas diferentes concebem abstrações diferentes

Dado um time de pessoas e um problema que deve ser resolvido via implementação de uma aplicação, a seguinte situação sempre ocorre:
Cada pessoa do time interpreta o problema de forma diferente e, portanto, cria uma abstração (solução) diferente. A maior parte das vezes essas abstrações conflitam entre si.
Como a ES resolve esse problema?

Engenharia de Software

Existe para resolver o problema das múltiplas abstrações (modelos) existente na construção de software.
Múltiplas abstrações devem convergir, no menor tempo e custo possíveis, para uma implementação correta, de fácil compreensão, de fácil manutenção e evolução e com alto desempenho em qualquer escala.

Dimensões da Solução

- Pessoas
- Organização (Processos)
- Tecnologia

Como induzir a convergência para **uma** abstração?

Controlar a especificação

- Uso de linguagem informal: Português, Inglês, etc. Multiplas interpretações (imprecisa).
- Ling. Formal: Precisa. Requer treinamento e não é necessáriamente mais fácil de compreender. Pode ser traduzida para código. UML (visual), linguagens derivadas de lógica matemática (promela (SPIN), temporal logic of actions (TLA), etc).

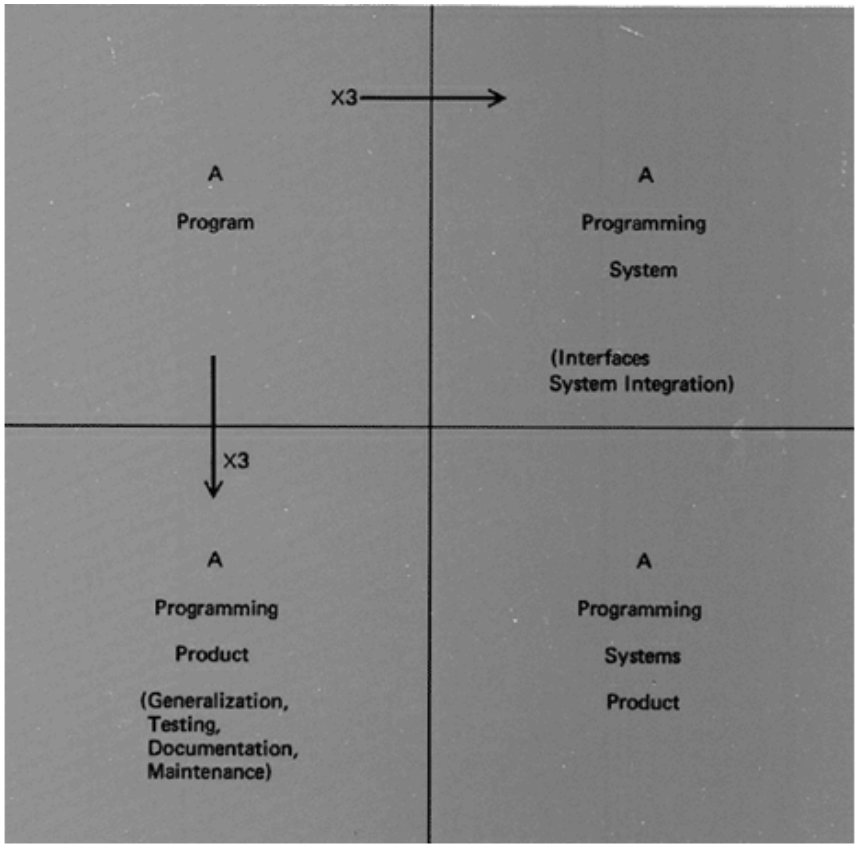
Solução atual: Misto de informal e formal (Inglês, UML). Adoção de Domain Driven Design. Orientação a Objetos. Automação de processo, e.g., jira (atlassian), trello, slack, azure devops, github issues, etc.

A aplicação é um programa ou um sistema?

- A program (programa): texto e executável normalmente escrito por uma pessoa.
- A programming product (produto): programa testado, documentado, mantenível, extensível.
- A programming system (componente): Conjunto de programas com interfaces bem definidas, foco em reutilização.
- A programming systems product (produto de componentes): Sistema com muitos componentes que pode ser executado, testado, reparado e evoluído por vários times.

Crédito (texto e figura): The Mythical Man-Month, Frederick P. Brooks Jr.

A aplicação é programa ou um sistema?



Um programa ou um sistema?

Lição aprendida

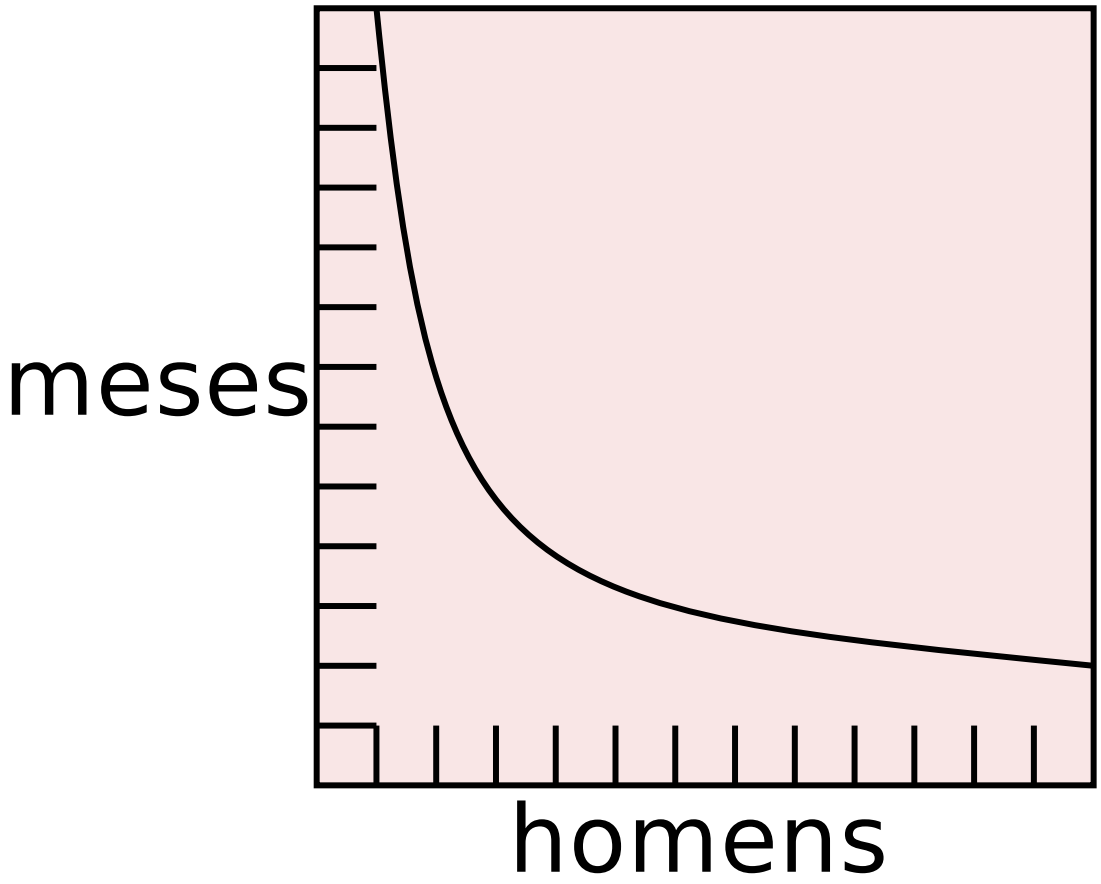
O desenvolvimento de sistemas tende a ter um custo que é, no mínimo, uma ordem de grandeza maior que o custo de se desenvolver um programa individualmente.

Estatísticas de ES mostram que o custo de desenvolvimento de uma aplicação pode ser muitas ordens de grandeza maior que o custo de desenvolvimento de um protótipo para a mesma aplicação.

Solução atual: Desenvolvimento ágil, continuous integration, continuous deployment (CI/CD), etc.

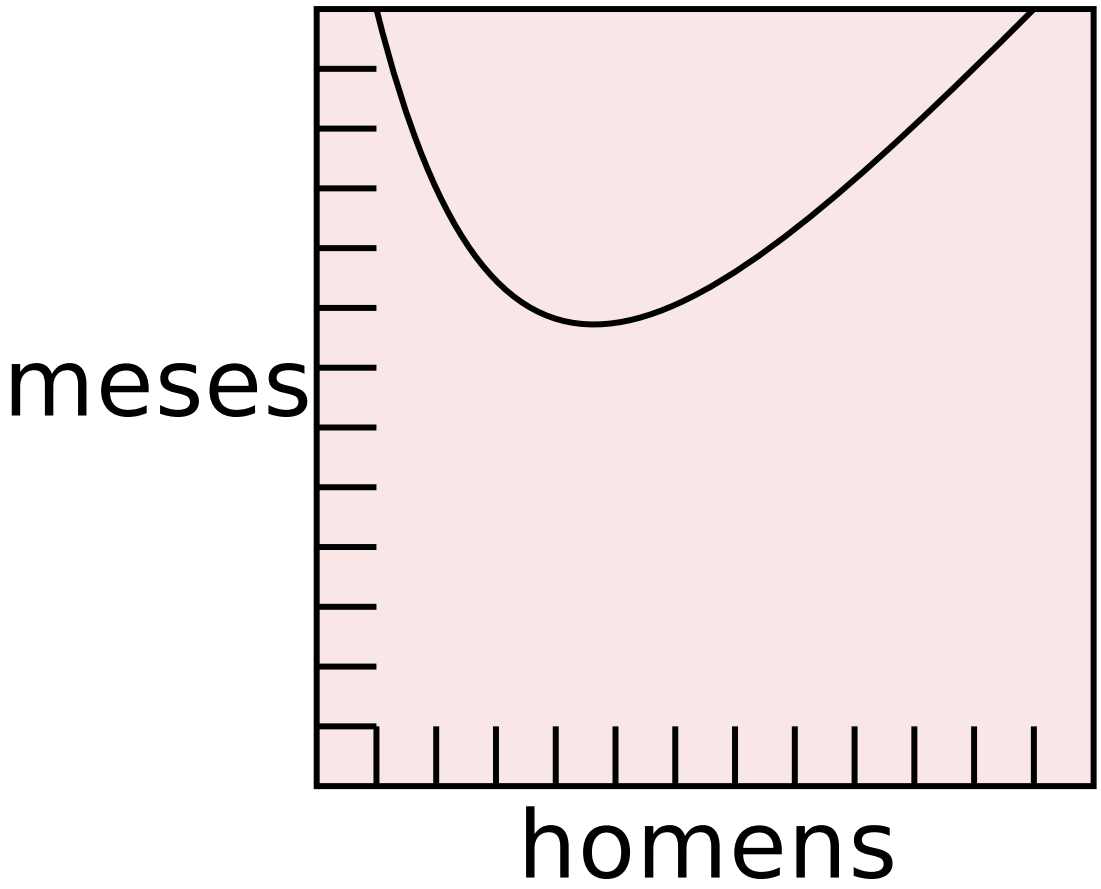
É possível reduzir o custo da comunicação entre pessoas?

Atividades perfeitamente particionáveis.



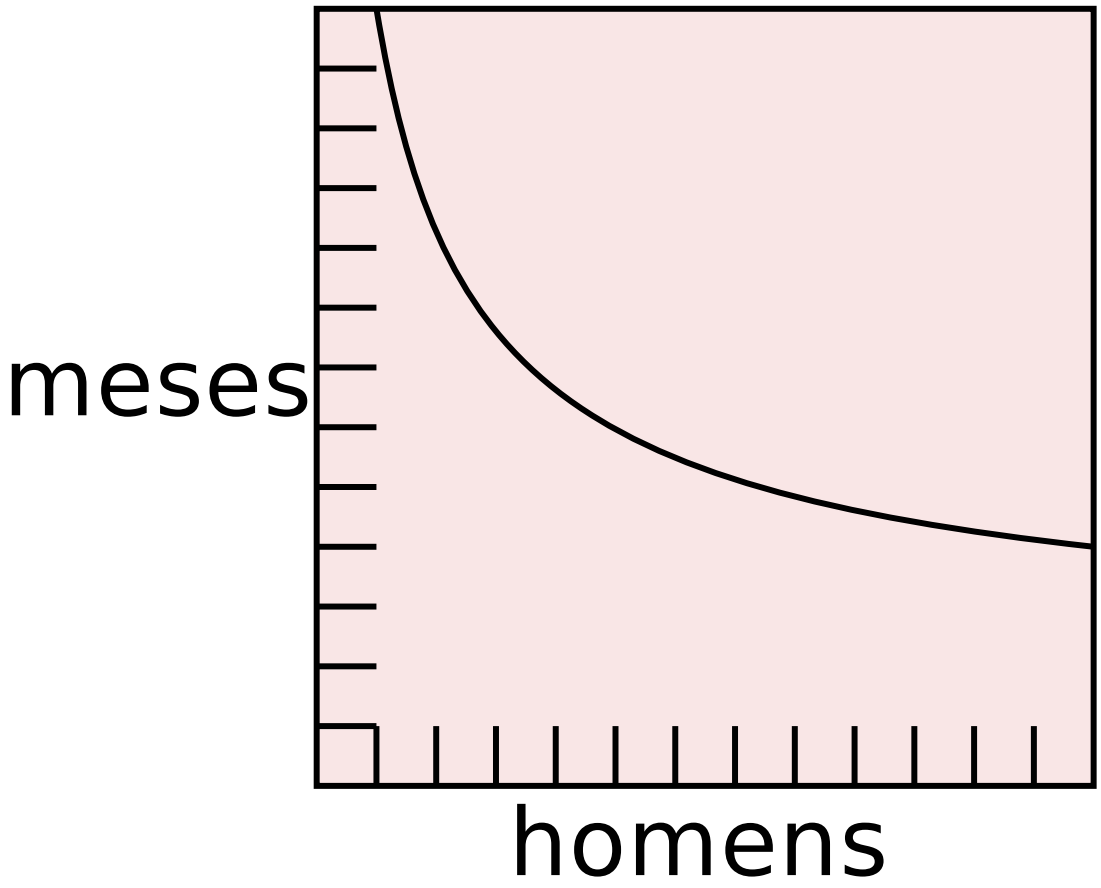
É possível reduzir o custo da comunicação entre pessoas?

Atividades não particionáveis (fortemente interdependentes).



É possível reduzir o custo da comunicação entre pessoas?

Atividades particionáveis que pedem comunicação. Caso típico para times de desenvolvimento.



É possível reduzir o custo da comunicação entre pessoas?

Tarefas

Se a conclusão de tarefas exige comunicação entre os membros do time, então há um limite a partir do qual a adição de pessoas ao time reduz sua eficiência ao invés de aumentá-la.

Em processos ágeis, o número de pessoas por time ágil é função de estudos da eficiência de comunicação em função do tamanho do time. Escalar ágil eficientemente é um dos desafios de empresas de produção de software. O que o conceito de micro-serviços tem a ver com esse problema?

Engenharia de Software

Dimensões da Solução

- Pessoas: convergência de abstração, comunicação eficiente
- Organização (Processos): processos ágeis (scrum, xp, etc)
- Tecnologia: automação de processos.

Engenharia de Software

A ES vista do alto:

`while (true)`

dev Análise:
Problema (Especificação) $\rightsquigarrow$ Modelos (Estrutural, Dinâmico, Funcional).

dev Projeto:
Modelos $\rightsquigarrow$ Arquitetura da Aplicação e Projeto de Objetos.
Algoritmos e Estruturas de Dados.
Implementação e Teste (debugging).

op Implantação (Monitoração de defeitos)

dev Manutenção (Defeitos são priorizados)

Engenharia de Software

A ES vista à olho nu:

Componentes

- Tecnologia: C, java, python, rust, go, c++, c#, visual basic, javascript, R, ...
- Tecnologia: frameworks, stacks, ferramentas para devops, etc, ...
- Processo: Existem muitos: cascata, espiral, RAD, RUP, XP, *go horse*. Tendência atual: processo ágil.
- Pessoa: Deve aprender processo e tecnologia. Comunicação. Trabalho em time.

Engenharia de Software

A ES vista usando um microscópio:

Pessoa, Processo, Tecnologia

Cada um dos *componentes* é estudado em detalhe. Domínio da linguagem de programação. Atualmente, IA pode fazer o papel de "pair programmer", uma prática recomendada em "extreme programming."

Atualmente

- Pessoa: interação direta (redes sociais e profissionais [marketing pessoal]), adaptação (remoto x presencial), agilidade (scrum, xp, etc). Uso de IA;
- Processo: dominância de ágil e suas variações. nicho: estruturado, clean room, etc. Uso de IA.
- Tecnologia: virtualização (processo e comunicação), componentização (reuso), alta disponibilidade e segurança. HTML, HTTPS: REST. Uso de IA (*vibe coding*!).

Tendência

Consolidação da automação

Ao longo dos anos, a ES tem se beneficiado da automação de seus processos. Hoje uma pessoa que participa de um time de ES tem o apoio de um conjunto enorme de ferramentas que codificam e automatizam o processo de trabalho. Ferramentas baseadas em IA têm aumentado a sua incursão em tarefas de ES, de auxíliares na coleta de requisitos e especificação, passando por codificação, depuração, monitoração e instalação, operação, etc. Por ora, parece que IA produz auxiliares interessantes. Será que algum dia terá um papel mais importante que o de auxiliar?

Existe alguma tarefa que IA ainda não consiga abordar?

Engenharia de Software Moderna

Computação, Comunicação (sistemas distribuídos):

- Computação: containers (docker, kubernetes), serverless (lambda functions);
- Comunicação: HTTPS (over virtualized tcp/ip networks).
- Automação: IaC (Infrastructure as Code): Ansible, Terraform, Puppet, Chef, etc.

Engenharia de Software Moderna

Microserviços

Uma aplicação é composta por um conjunto de componentes distribuídos que podem ser analisados, projetados, desenvolvidos, instalados e escalados de forma autônoma (independente). Arquiteturas baseadas em microserviços apoiam o aumento de escala de produção de aplicações utilizando processo ágeis.

Engenharia de Software Moderna

Ênfase em automação.

- controle de versão: git, github, bitbucket, etc;
- integração contínua, instalação contínua (CI/CD): Jenkins, Gitlab CI, CircleCI, etc;
- configuration management: há sobreposição com IaC: Ansible, Terraform, Puppet, Chef, etc;
- monitoring, logging: Prometheus, Grafana, ELK stack (Elasticsearch, logstash, kibana);
- automated testing: Selenium, JUnit, TestNG, etc;
- automation of collaboration: Slack, Microsoft Teams, Jira, Github, etc.

Milestones (Resumo)

Domínio: Web Applications

- tecnologia: HTML (XML), JSON, etc (representação de estado de um recurso) [1970s]
- tecnologia: HTTP (W3C), URI (URL): resources, identifiers [1999]
- tecnologia: HTTP: GET, POST, PUT, DELETE, OPTIONS, HEAD, TRACE, CONNECT, PATCH
- tecnologia: HTTP: número reduzido de verbos implementa C (post) R (get) U (put) D (delete).
- tecnologia: Representational State Transfer (REST) [2000]
- tecnologia: REST Architectural Style (Roy Fielding): arquitetura baseada em APIs para WEB [2000]
- processo: manifesto ágil [2001]
- processo: development & operation (devops): integração, automação
- pessoas: dev e op passam a ser integrados.

Orientação a Objetos

Paradigma Dominante

É ortogonal à engenharia de sofware, afinal é um paradigma, isto é, um modelo de referência que determina como um sistema é abstraido, definido e implementado.

Paradigma de Objetos

Objeto

Um objeto é um artefato (construção) que encapsula um **estado** e permite a sua modificação através de **procedimentos**. O estado de um objeto é especificado através de um conjunto de **atributos** (variáveis), os seus procedimentos são especificados através de um conjunto de **métodos**. Um objeto é identificado unicamente. Um objeto é uma instância de uma classe. A invocação (execução) de um método é uma *mensagem*. Assim, a interação entre objetos ocorre através da *troca* de mensagens entre eles, isto é, um objeto executa um método do outro objeto. Cadeia de mensagens.

Paradigma de Objetos

Classe

Especificação dos atributos e métodos comuns a um conjunto de objetos. Um classe é fudamentalmente uma abstração que representa um conjunto **potencial** de objetos.

Análise

Domínio da Aplicação

Foco no entendimento e modelagem da aplicação e no domínio em que ela funcionará. Entradas iniciais para o analista são uma especificação do problema e entrevistas com usuários. A saída da análise são modelos que capturam os três aspectos essenciais de um sistema: objetos e seus relacionamentos, o fluxo de controle e transformação funcional dos dados.

Projeto de Sistema

Domínio da Computação

Foco na determinação da arquitetura de software do sistema (aplicação). Usa-se o modelo de objetos como guia para organizar-se o sistema em sub-sistemas. Modela-se a concorrência entre objetos. Escolhem-se os algoritmos base e grupam-se conjuntos de objetos em unidades básicas de execução que serão atribuídas a processos. Decide-se sobre forma de comunicação entre processos, armazenamento de dados e fluxo de execução entre sub-sistemas.

Projeto de Objetos

Domínio da Computação

Durante o projeto de objetos a ênfase migra dos conceitos da aplicação para os conceitos de computação. Escolhem-se algoritmos e estruturas de dados. Em função das escolhas refinam-se todos os modelos resultantes da análise. Decide-se como implementar os relacionamentos, hierarquias, invocações, etc.

Quais são as fases?

- Projeto de sistema (fora do escopo deste curso: INF325, INF331, INF332)
- Projeto de objetos

Projeto de Objetos

Atividades

- Projetar estruturas de dados e algoritmos.
- Definir o fluxo de controle (regras de negócio, algoritmos).
- Refinar heranças, com foco na minimização de código e na eficiência conceitual e de implementação.
- Implementar classes, objetos e relacionamentos de forma eficiente.
- Atenção à granularidade e persistência dos objetos.
- Persistência de dado é diferente de persistência de objeto (SGBDs, in-memory stores, arquivos, etc).

Projeto Orientado a Objetos

Resumidamente

Tradução **iterativa** de modelo para implementação. Cada iteração deve procurar **melhorar** a implementação e, retrospectivamente, o modelo.

Projeto Orientado a Objetos

Ferramentas Conceituais

são idênticas as usadas em análise mas com **foco** na implementação:

- Abstração: distanciamento, isolamento, separação, modularização (dualidade: objeto-classe);
- Hierarquização: organização de um sistema em sub-sistemas que retém alguma relação de ordem entre si (generalização/especialização);
- Encapsulamento (Modularização, Componentização, etc): definição de interface, minimização de comportamento (polimorfismo);
- Abstração de conexão entre objetos, classes: associação, agregação, delegação.

Por que as ferramentas conceituais não mudam?

Quem faz?

Responsáveis

- programador;
- arquiteto de software.

Por que é importante?

Relevância

- Um sistema orientado a objetos se apóia nas definições de classes que são derivadas do modelo de análise.
- A análise se preocupa, predominantemente, com o domínio do problema, logo os modelos podem:
  - Ser ineficientes,
  - Não se adaptar à arquitetura do sistema e/ou da aplicação,
  - Violar requisitos não funcionais,
  - Não serem facilmente particionáveis em módulos, componentes, etc.

Em tese, o projeto de objetos deve solucionar todos esses problemas.