## Parte 01 - Porque Orientação a Objetos?

Vivemos num mundo de objetos. Esses objetos existem na natureza, nas entidades feitas pelo homem, nos negócios e nos produtos que usamos. Eles podem ser categorizados, descritos, organizados, combinados, manipulados e criados. Conseqüentemente, não é surpresa que um enfoque orientado a objetos fosse proposto para a criação de software, ou seja uma abstração que nos permite modelar o mundo em modos que nos ajudam a melhor entendê-lo e a navegar nele.

A grande maioria dos métodos empregados como modelo para o desenvolvimento de software, em geral, é baseada em duas abordagens: na decomposição funcional e na modelagem de dados.

Verifica-se que tais métodos estão constituídos segundo um paradigma estruturado, onde se têm visões dissociadas em que se distingue claramente os processos de dados. Os processos possuem uma modelagem própria e os dados idem. Em algum ponto, nos métodos tradicionais empregados como análise estruturada e análise essencial, as abordagens de processos e dados se cruzam, uma vez que na realidade os processos acessam os dados, de alguma forma, o método deve retratar com fidelidade o que ocorre no ambiente real. Tanto nos métodos da análise estruturada quanto na análise essencial, o Diagrama de Fluxo de Dados é empregado na representação da funcionalidade; porém, como elemento necessário que dá subsídios às funções, também aparecem os dados nos depósitos de dados; posteriormente, ou paralelamente, desenvolve-se uma modelagem de dados, em geral, empregando-se o Diagrama de Entidade de Relacionamento.

No paradigma orientado a objetos, os dados e as funções são vistos de forma agregada, não ocorrendo uma modelagem separada para cada um desses componentes; portanto, sob esse ponto de vista, a orientação a objetos favorece uma modelagem mais natural, que melhor retrata a realidade, pois processos e dados estão coligados, não se encontram dissociados em nenhuma atividade real.

A orientação a objetos fundamenta-se em princípios que não são novos. Especialmente como modelo para o desenvolvimento de software, a orientação a objetos possui uma propriedade sinergética, na qual seus componentes podem ser arranjados para melhor espelhar a solução sistêmica dada a um problema do usuário; o qual deve previamente ser entendido na fase de levantamento dos requisitos.

O paradigma da orientação a objetos surgiu primeiramente com a programação de computadores no final dos anos 60, com a linguagem SIMULA. Nos anos 70, era parte importante da linguagem SMALLTALK, desenvolvida pela Xerox. Esse paradigma só estreou na análise de sistemas no final da década de 80.

No início dos anos 90, o paradigma da orientação a objetos passou a ocupar lugar de destaque no desenvolvimento de software. Três aspectos foram relevantes para a ascensão da orientação a objetos, contrapondo-se aos métodos estruturados: a unificação de conceitos entre as fases de análise e programação, o grande potencial de reutilização do software e a facilidade de manutenção. A orientação a objetos também se apresentou com a esperança de suprir algumas das preocupações da indústria do software: a necessidade de criar softwares corporativos muito mais rapidamente, mais confiáveis e a um custo baixo. Os meios para conseguir essa proeza encontram-se nas características apresentadas pelo paradigma da orientação a objetos, o que leva a um salto quantitativo e qualitativo na produção do software.

A característica básica dos métodos orientados a objetos, que se apresenta como uma grande vantagem quanto a sua utilização, é a façanha de terem unificado os formalismos utilizados na análise, projeto e programação. Os conceitos envolvidos são os mesmos, independentemente da fase

no desenvolvimento do software, o que, teoricamente, vem facilitar a forma de comunicação entre o pessoal técnico envolvido no projeto.

A orientação ao objeto se dedica a desenvolver um modelo orientado a objetos do domínio da aplicação. Os objetos identificados refletem entidades e operações que estão associadas com o problema a ser resolvido.

O enfoque da orientação a objetos, é uma visão sobre um mundo como uma coletânea de objetos que interagem entre si, apresentando características próprias que são representadas pelos seus atributos e operações.

## O QUE É UM OBJETO?

Quando nos damos conta das coisas que nos cercam, intuitivamente constatamos a presença de vários objetos. É também muito natural classificar esses objetos a partir de vários pontos de vista: forma, utilidade, etc. Percebe-se que cada objeto tem características segundo sua estrutura e funcionalidade. Ao examinar mais cuidadosamente determinado ambiente, verifica-se que existem objetos que são semelhantes.

Um objeto é uma entidade que possui um estado e um conjunto definido de operações que operam nesse estado. O estado é representado por um conjunto de atributos de objeto. As operações associadas com o objeto fornecem serviços para outros objetos, que requisitam esses serviços quando uma computação é necessária. Os objetos são criados de acordo com uma definição de classe de objetos, que serve com um modelo para criar objetos. Essa classe apresenta declarações de todos os atributos e operações que devem ser associados a um objeto dessa classe. A notação utilizada para as classes de objetos é definida na UML.

Objeto é uma unidade de software que consiste em atributos, e de métodos.

Algumas propriedades dos objetos:

- Estado: Diz respeito à situação em que pode estar um determinado objeto. O estado depende da natureza do objeto;
- Comportamento: Qualquer objeto apresenta um comportamento. O comportamento é o meio pelo qual o objeto passa de um estado para o outro. Normalmente, isso dá mediante uma condição/ação a qual será sempre um método a ser executado;
- Identificação: Todo o objeto é identificável, portanto, são distinguíveis entre si.

## CLASSE DE OBJETOS

Uma classe representa um conjunto de objetos da mesma característica. Cada um deles é uma instância da classe, dado a sua existência, são diferenciados por sua identidade. Uma instância herda as características da classe a que pertence.

No conceito de orientação a objeto, classe é um conceito que encapsula as abstrações de dados e procedimental necessárias para descrever o conteúdo e comportamento de alguma entidade do mundo real.

Classe é uma coleção de objetos que podem ser descritos com os mesmos atributos e as mesmas operações.

## ENCAPSULAMENTO

Encapsulamento é o conceito que faz referência à ocultação ou empacotamento dos dados e procedimentos dentro do objeto. Não há dados ou procedimentos fora de um objeto. Cada objeto poderá conter atributos, e procedimentos que guardará dentro de si.

O encapsulamento de operações e atributos, atribui vida própria ao objeto.

## ACOPLAMENTO DINÂMICO, HERANÇA E POLIMORFISMO

Ao receber uma mensagem, o objeto verificará se existe um serviço, também chamado de método, que defina seu comportamento perante essa mensagem. A mensagem sempre aciona um método correspondente no objeto que recebeu a mensagem.

Pode ser que não exista no objeto acionado, um método de acordo com a inovação feita pela mensagem. Caso o objeto não encontre um método dentro de seu encapsulamento, verificará em seus ramos de herança aquelas superclasses que tenham o método invocado.

Esse mecanismo de busca de serviços é chamado de *acoplamento dinâmico*.

Mensagens iguais, destinadas a objetos diferentes, podem gerar comportamentos diferentes. Para uma mesma mensagem, objetos diferentes podem responder ou agir de forma diferenciada; a isso chamamos polimorfismo.

## BENEFÍCIOS DO PARADIGMA DA ORIENTAÇÃO A OBJETOS

Espera-se uma ampla gama de benefícios com os quais a aplicação da orientação a objetos possa vir a trazer no desenvolvimento do software. A expectativa é que venha a superar todos os benefícios atingidos pelos métodos convencionais de construção de software ainda utilizados.

- Modelagem mais natural: A aplicação dos conceitos da orientação a objetos na análise de sistemas permitirá modelar a empresa ou as áreas da aplicação de uma forma mais natural, visto que os recursos a serem aplicados retratam com mais facilidade o mundo real. A capacidade de retratar o universo pesquisado é moldado em diagramas que refletem exatamente o arranjo dos objetos no mundo real. A própria transição entre etapas na construção do software, aplicando-se o paradigma orientado a objetos, são refinações sucessivas com os mesmos conceitos e linguagem.
- Reutilização: Diante da forma como são projetados os recursos do software, é possível atingir a maximização na reutilização. Várias classes projetadas constituirão bibliotecas, que poderão ser acionadas por outros projetos diferentes daqueles que a originaram.
- Projetos mais rápidos com qualidade: Em função da característica da reutilização, uma vez que existam bibliotecas que ofereçam classes com recursos necessários, novos projetos utilizarão esses componentes pré-existentes. Em conseqüência, se espera que novas construções devam demorar menos tempo do que se feitas de outra forma. O fato das bibliotecas serem utilizadas por diversos projetos, implica que os recursos lá existentes estarão testados e com sua funcionalidade aprovada; portanto, além da velocidade na construção de novos projetos, acabam colaborando para a inexistência de erros, aumentando o grau de qualidade do software.
- Codificação mais simples de programas: Também dado à estrutura de como se apresenta o paradigma da orientação a objetos, a codificação de métodos reduz a complexidade na construção do código dos programas. Isso traz simplicidade no momento de codificar, testar e manter o software.

É importante destacar que os benefícios da tecnologia orientada a objetos são ampliadas se ela é adotada no início e ao longo de todo o processo de engenharia de software.

## UML – Unified Modeling Language

A UML é uma linguagem de modelagem para documentar e visualizar os artefatos que especificamos e construímos na análise e desenho de um sistema.

A UML é uma linguagem de modelagem, totalmente orientada a objetos, que une as melhores práticas e metodologias da Engenharia de Software. É considerada a sintaxe geral para criar um modelo lógico de um sistema. Ela é utilizada para descrever pontos de um sistema e da forma como ele é percebido de várias visões durante a análise e sua arquitetura. É uma linguagem que visa capturar conhecimento e expressar esse conhecimento. Seu propósito é a modelagem de sistemas, documentar de maneira interativa e visual, proporcionar melhor compreensão e sinergia entre o analista e o cliente envolvido no processo de desenvolvimento.

## Modelagem Visual

A modelagem visual é o uso de notações de design gráficas e textuais, semanticamente ricas, para capturar design de software. Uma notação, como a UML, permite que o nível de abstração seja aumentado, enquanto mantém sintaxe e semântica rígida. Dessa maneira, a comunicação na equipe de design melhora, à medida que o design é formado e revisado, permitindo ao leitor raciocinar sobre ele e fornecendo uma base não ambígua para a implementação.

## Por que Modelamos?

Um modelo é uma visão simplificada de um sistema. Ele mostra os elementos essenciais do sistema de uma perspectiva específica e oculta os detalhes não essenciais. Os modelos podem ajudar das seguintes maneiras:

- Ajudando na compreensão de sistemas complexos;
- Explorando e comparando alternativas de design a um baixo custo;
- Formando uma base para implementação;
- Capturando requisitos com precisão;
- Comunicando decisões sem ambigüidade.

## Ajuda na compreensão de sistemas complexos

A importância dos modelos aumenta à medida que os sistemas se tornam mais complexos. Um aplicativo pequeno, criado por uma única pessoa em alguns dias, pode ser facilmente compreendido em sua totalidade. No entanto, um sistema de comércio eletrônico com dezenas de milhares de linhas de código-fonte, ou um sistema de controle de tráfego aéreo de centenas de milhares de linhas de código, não pode mais ser facilmente entendido por uma única pessoa. A construção de modelos permite a um desenvolvedor se concentrar na visão geral, entender como os componentes interagem e identificar falhas fatais.

Alguns exemplos de modelos são:

- Casos de Uso para especificar comportamento de forma não ambígua;
- Diagramas de Classes e Diagramas de Modelo de Dados para capturar design;
- Diagramas de Transição de Estado para modelar comportamento dinâmico.

A modelagem é importante, pois ajuda a equipe a visualizar, construir e documentar a estrutura e o comportamento do sistema, sem se perder na complexidade.

**Exploração e comparação de alternativas de design a um baixo custo**

Modelos simples podem ser criados e modificados a um baixo custo para explorar alternativas de design. Idéias inovadoras podem ser capturadas e revisadas por outros desenvolvedores, antes de investir em um desenvolvimento de código caro. Quando associada ao desenvolvimento iterativo, a modelagem visual ajuda os desenvolvedores a avaliar mudanças de design e a comunicar essas mudanças a toda a equipe de desenvolvimento.

**Formação de uma base para implementação**

Atualmente, muitos projetos empregam linguagens de programação orientadas a objetos para obter sistemas reutilizáveis, tolerantes a mudanças e estáveis. Para obter essas vantagens, é ainda mais importante usar tecnologia de objetos em design. O Rational Unified Process (RUP) produz um modelo de design orientado a objetos que é a base para a implementação. Com o suporte das ferramentas adequadas, um modelo de design pode ser usado para gerar um conjunto inicial de código para implementação. Isso é referido como "engenharia direta" ou "geração de código". Os modelos de design também podem ser aprimorados para incluir informações suficientes para criar o sistema.

A engenharia reversa também pode ser aplicada para gerar modelos de design a partir de implementações existentes. Isso pode ser usado para avaliar implementações existentes.

"A engenharia round-trip" combina as técnicas de engenharia direta e reversa para garantir código e design consistentes. Combinada com um processo iterativo e com as ferramentas certas, a engenharia round-trip permite que o código e o design sejam sincronizados durante cada iteração.

**Captura de requisitos com precisão**

Antes de criar um sistema, é essencial capturar os requisitos. A especificação dos requisitos usando um modelo preciso e não ambíguo ajuda a garantir que todos os envolvidos possam entender e concordar com os requisitos. Um modelo que separa o comportamento externo do sistema da implementação o ajuda a se concentrar no uso pretendido do sistema, sem ficar perdido nos detalhes de implementação.

**Comunicação de decisões sem ambigüidade**

O RUP (Rational Unified Process) usa a UML, que é uma notação consistente que pode ser aplicada tanto para a engenharia de sistemas quanto a engenharia de negócios. Uma notação padrão serve aos seguintes papéis:

- "Serve como uma linguagem para comunicar decisões que não são óbvias ou que não podem ser deduzidas do próprio código."
- "Fornece semântica rica o suficiente para capturar todas as decisões estratégicas e táticas importantes."
- "Oferece uma forma concreta o suficiente para que as pessoas raciocinem e para que as ferramentas sejam manipuladas."

UML representa a convergência da melhor prática em modelagem de software por toda a indústria de tecnologia de objetos.

## RUP

O RUP (Rational Unified Process) é um framework genérico para processos de desenvolvimento de software, criado pela empresa Rational Software Corporation, que está fortemente centrado na arquitetura, funcionalidade (caso de uso) e o desenvolvimento iterativo e incremental (inspirado no ciclo de vida espiral de Boehm), que aplica a UML, para o projeto e a documentação.

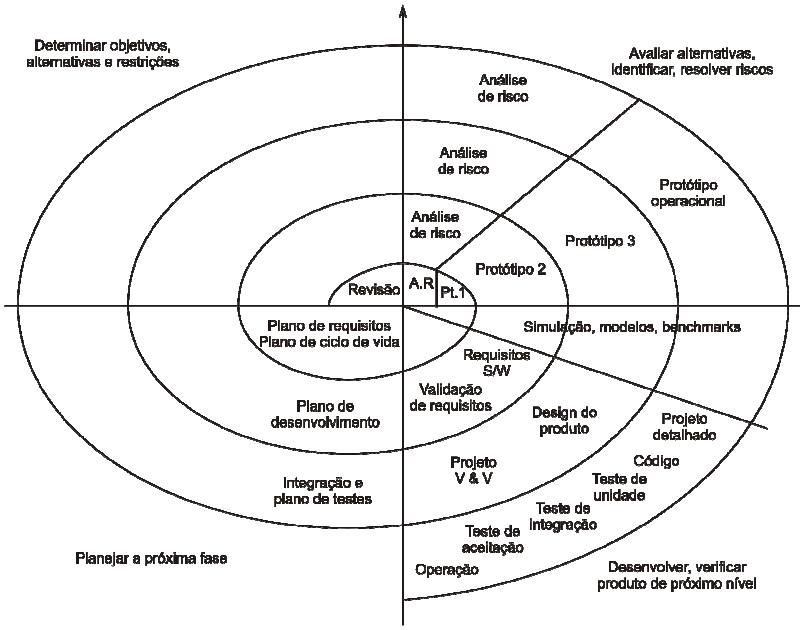


Figura 01 - Espiral de Boehm.

O RUP é um processo de desenvolvimento de software e, como tal, descreve os papéis e as atividades que cada membro da equipe de projeto deve desempenhar ao longo do ciclo de desenvolvimento do software e os produtos que devem ser gerados como resultado destas atividades, os chamados artefatos.

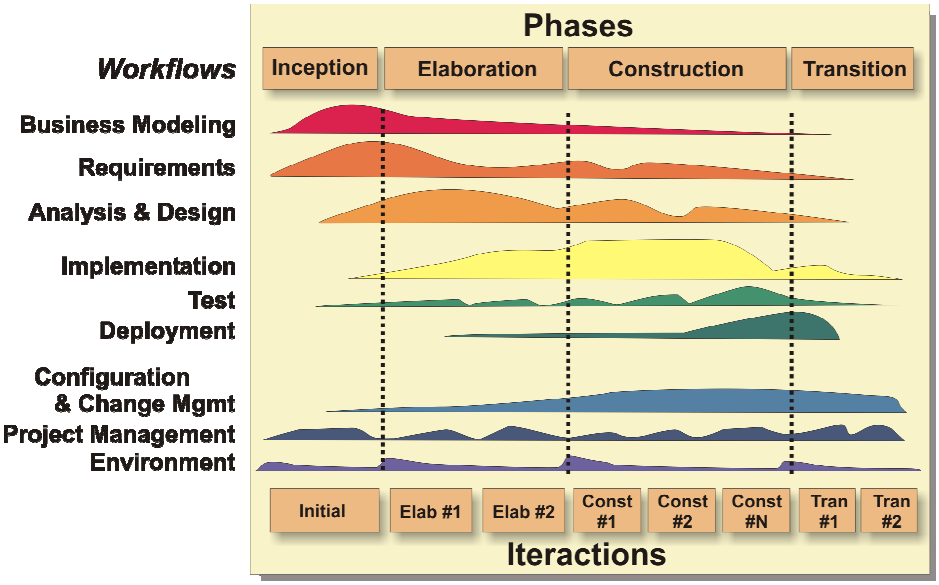


Figura 02 – *Rational Unified Process* (RUP).