ATIVIDADES DO PROJETO DE SOFTWARE DO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

PROJETO DE SOFTWARE

Podemos observar que os principais modelos da etapa de análise do processo de desenvolvimento de software são: o modelo de casos de uso, o modelo de classes de análise, o modelo de atividades e o modelo de estados.

Atenção

Muitos autores e profissionais de Engenharia de Software preferem usar o termo original do inglês design no lugar de projeto, para não confundir com o homógrafo projeto, no sentido de *project*, cujo significado é diferente de *design*. Neste texto, o termo **projeto** é usado indistintamente no sentido de project ou de design, dependendo do contexto.

Você acha possível implementar um software com os referidos modelos? Provavelmente, faltam mais detalhes. Vamos, agora, tratar desses detalhes.

O conceito de abstração é fundamental no processo de desenvolvimento de software, pois ele é iniciado com especificações e modelos com alto nível de abstração.

A etapa projeto de software diminui o nível de abstração dos diagramas de análise por meio de refinamentos sucessivos, como também exige a criação de novos modelos.

As principais atividades realizadas na fase de projeto serão descritas a seguir.

REFINAMENTO DOS ASPECTOS ESTÁTICOS E ESTRUTURAIS DO SISTEMA

O modelo de classes representa os aspectos estáticos e estruturais do sistema. A partir do modelo de classes gerado na análise, vamos inserir novos elementos que permitirão a implementação das classes, ou seja, aplicaremos refinamentos que possibilitam reduzir o grau de abstração. A Figura 5 exemplifica as abstrações possíveis de uma classe durante o processo de desenvolvimento de software. Ao final, temos uma especificação de classe em condições de ser codificada.

Exemplo

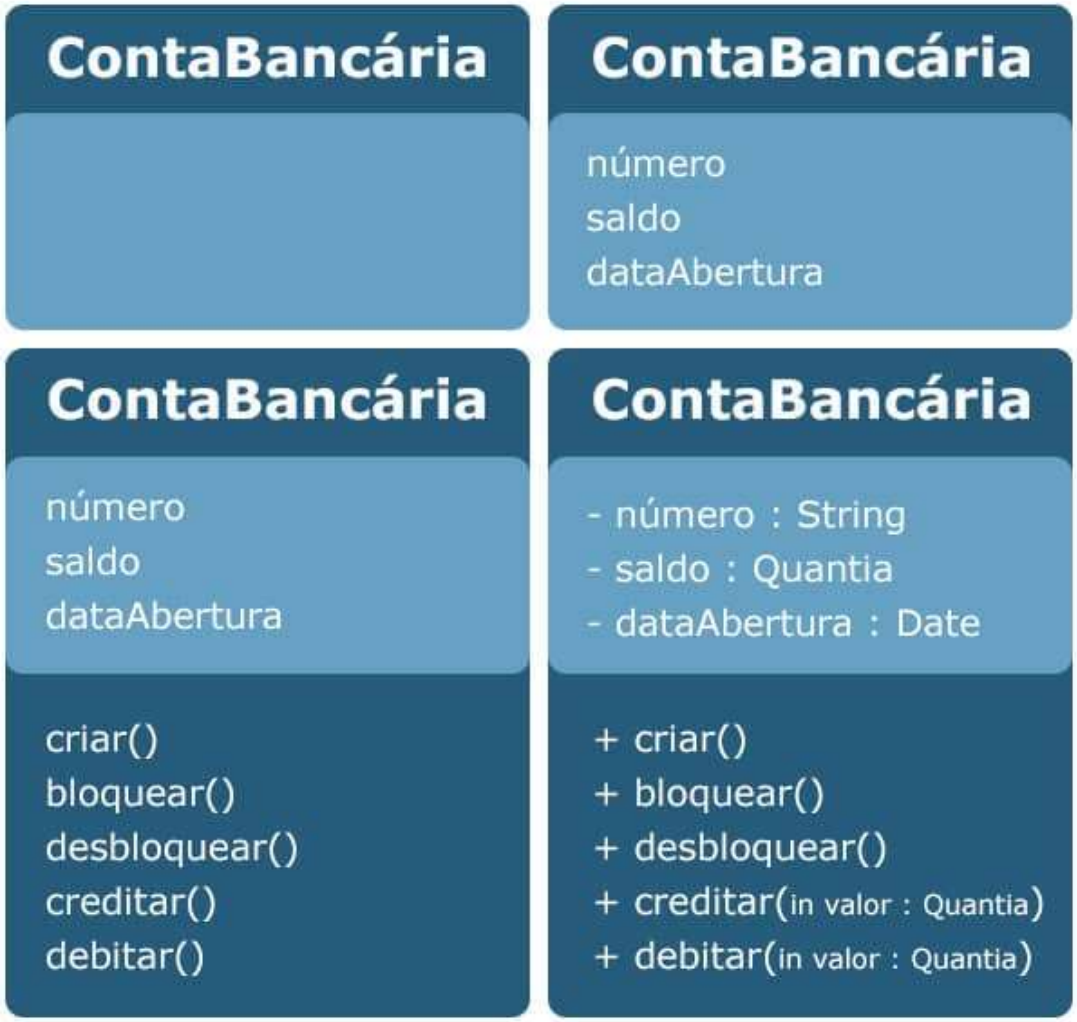

Figura 5 - Exemplo de refinamentos em uma classe.

Vamos analisar os refinamentos: a primeira estrutura indica uma classe e o respectivo nome; a segunda inclui os atributos; a terceira adiciona os métodos ou funções das classes, e a quarta detalha os atributos e métodos. Na análise, não é importante definirmos os tipos de dados, sendo fundamental no projeto, pois a especificação de projeto tem de garantir a implementação, ou seja, a codificação da classe.

Além desse refinamento, novos elementos são inseridos em função das associações entre as classes, das heranças existentes, ou mesmo em virtude da criação de novas classes.

Um aspecto importante é a possibilidade de aplicação de padrões de projeto, ou design *patterns*, no modelo de classes de projeto. Esses padrões representam *templates* de melhores práticas para a solução de determinados problemas comuns, que serão resolvidos mais facilmente quando

implementados pelo programador. A Figura 6 exemplifica o padrão *Factory Method*.

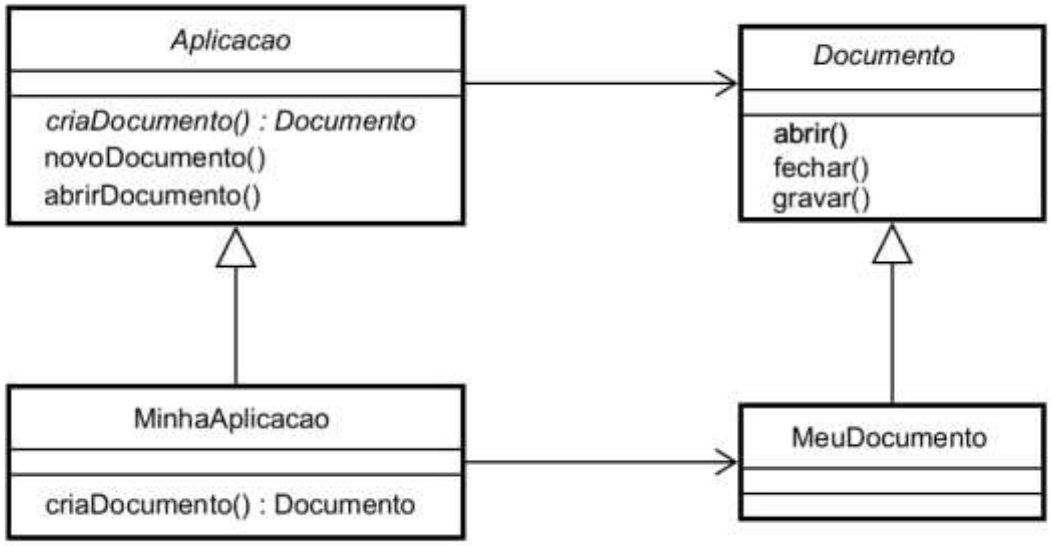

Figura 6 - Exemplo do padrão *Factory Method*.

DETALHAMENTO DOS ASPECTOS DINÂMICOS DO SISTEMA

O que seriam aspectos dinâmicos? Lembre-se: o paradigma dominante na indústria de software é a orientação a objetos, que, de forma simplificada, é a representação do mundo real por meio de objetos que se comunicam por meio de mensagens.

A referida comunicação representa a dinâmica existente entre os objetos e, para a sua representação, temos o modelo de interação, que inclui dois diagramas: diagrama de comunicação e diagrama de sequência. Ambos os diagramas têm a mesma abstração dinâmica com representações distintas e recíprocas.

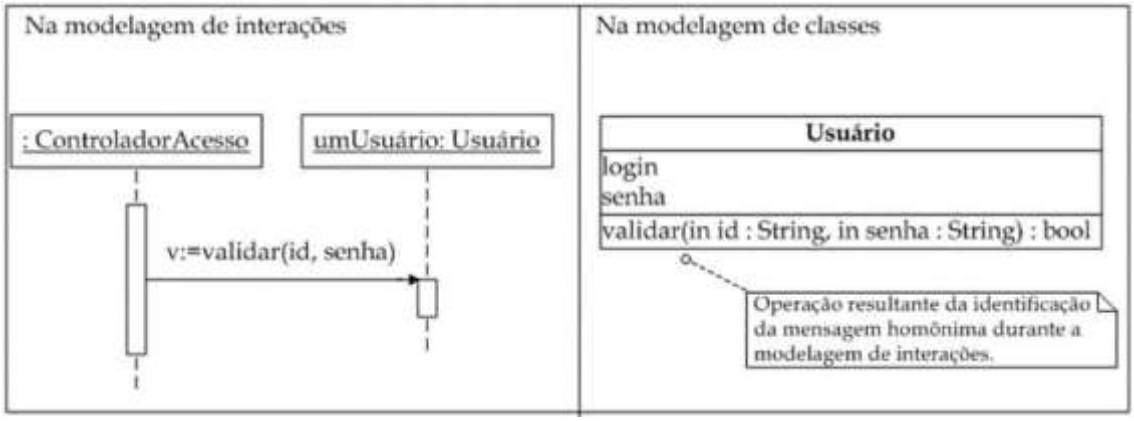

Figura 7 - Diagrama sequência *versus* classes de projeto.

Qual a relação existente entre o modelo de classes e o modelo de interação?

O modelo de interação é desenvolvido usualmente em paralelo com o modelo de classes de projetos, de modo que os métodos implementados nas classes são identificados a partir das mensagens definidas no modelo de interação. A Figura 7 exemplifica, à esquerda, parte de um diagrama de sequência, onde a mensagem “validar” estabelece uma comunicação entre os objetos “controlador acesso” e “usuário”, sendo a mensagem implementada no objeto receptor “usuário”, ou seja, a classe “usuário” inclui o método “validar” com os respectivos parâmetros.

DETALHAMENTO DA ARQUITETURA DO SISTEMA

Uma atividade que aplicamos de forma intensa na Engenharia de Software é a fatoração, ou seja, a decomposição da solução do problema em partes menores, facilitando principalmente o trato da complexidade.

Imaginemos um sistema de software complexo; intuitivamente, temos de realizar a decomposição em subsistemas. O processo do projeto que permite identificar os referidos subsistemas, estabelecendo um *framework* para controle e comunicação entre eles, é denominado projeto de arquitetura.

A arquitetura é definida por meio de duas abstrações: lógica e física.

A Figura 8 ilustra uma divisão lógica comumente aplicada no projeto de arquitetura, sendo composta das seguintes camadas: apresentação, aplicação, domínio e serviços técnicos.

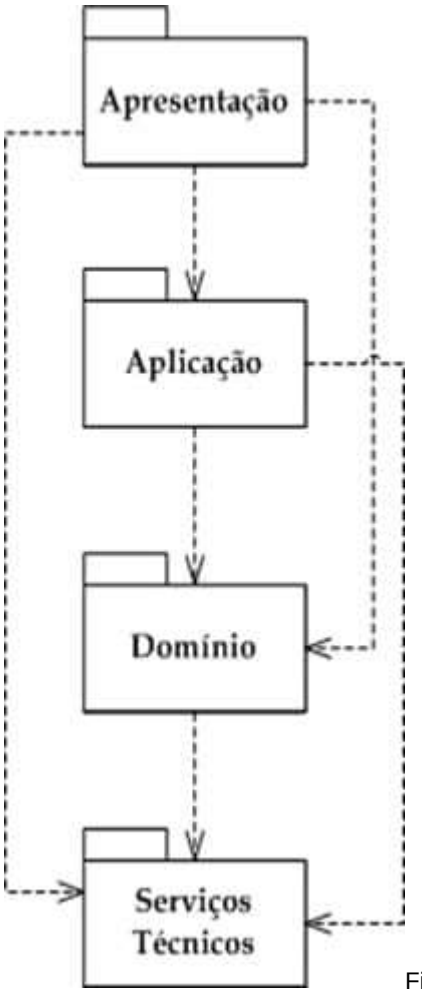

Figura 8 - Camadas de software.

Importante destacar que as camadas mais altas devem depender das camadas mais baixas – essa fatoração permite melhor trato da complexidade, ou seja, decomposição do sistema em partes menores. Outra vantagem é o reúso, pois as camadas inferiores são independentes das camadas superiores. A arquitetura em camadas permite baixo acoplamento, ou seja, mínimo de dependências entre os subsistemas, e alta coesão, isto é, deve haver mais associações entre as classes internamente do que externamente.

A camada de apresentação da Figura 8 inclui as denominadas classes de fronteira, que permitem a instanciação dos objetos de fronteira que interagem com os usuários.

A camada de aplicação inclui as classes de controle, que permitem a instanciação dos objetos de controle que atuam como intermediários entre os objetos da camada de apresentação e os objetos do domínio do problema.

A camada de domínio é composta pelas classes de domínio oriundas do diagrama de classes, que permitem a instanciação dos objetos do domínio do problema – esses objetos podem conter métodos que alteram o próprio estado.

Finalmente, a camada de serviços técnicos pode incluir serviços genéricos, cabendo destaque à camada de persistência de dados.

Assim como existem padrões de projeto aplicados ao projeto de classes (Figura 6), temos também padrões aplicados à arquitetura. Como exemplo, podemos destacar o padrão de arquitetura em camadas denominado MVC (*Model-View-Controller*), formulado na década de 1970 e focado no reúso de código e na separação de conceitos em três camadas interconectadas, tal como ilustra a Figura 9.

O controlador (*controller*) tem funcionalidade para atualizar o estado do modelo, como, por exemplo, uma nota fiscal, ou alterar a apresentação da visão do modelo.

Um modelo (*model*) mantém os dados, notificando visões e controladores quando da ocorrência de mudança em seu estado.

A visão (*view*) permite a exibição dos dados.

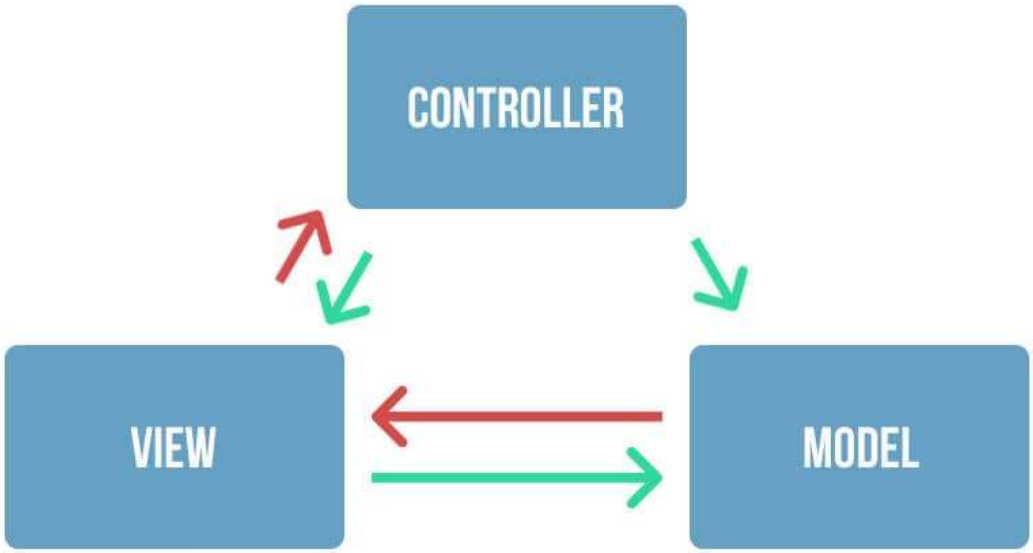

Figura 9 - Arquitetura MVC.

Após a definição lógica da arquitetura, o engenheiro de software precisará definir a arquitetura física por meio do modelo de implantação, que projeta a infraestrutura de hardware necessária para o software projetado.

Você poderia questionar: “O engenheiro de software especifica hardware?”.

Nesse caso, não, mas o projeto tem de definir os nós computacionais necessários para a implantação do software, devendo a especificação do hardware ficar por conta da equipe de infraestrutura. A Figura 10 exemplifica um diagrama de implantação com quatro nós computacionais e as respectivas conexões.

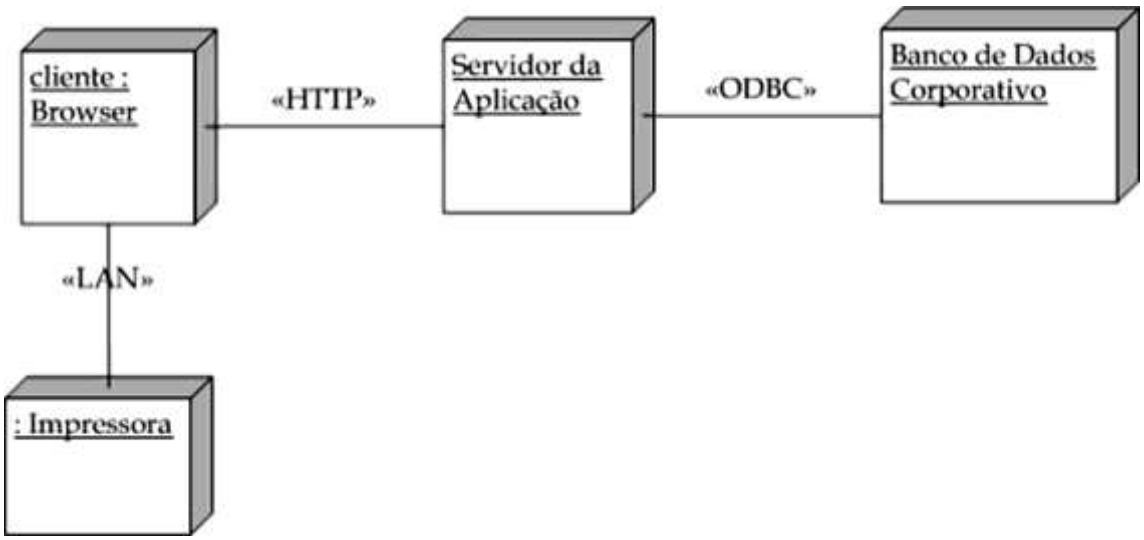

Figura 10 - Diagrama de implantação.

Você lembra qual a composição das camadas apresentadas?

Cada camada é composta por um conjunto de classes, formando componentes diversos. Os componentes podem ser compostos por uma ou mais classes que encapsulam funcionalidades de forma independente – um conjunto de componentes pode compor um sistema complexo. A forma de representar essa “componentização” do software é definida no modelo de implementação. A Figura 11 ilustra o diagrama de componentes embutido no diagrama de implantação da Figura 10, de modo que podemos verificar a alocação por nó computacional dos referidos componentes.

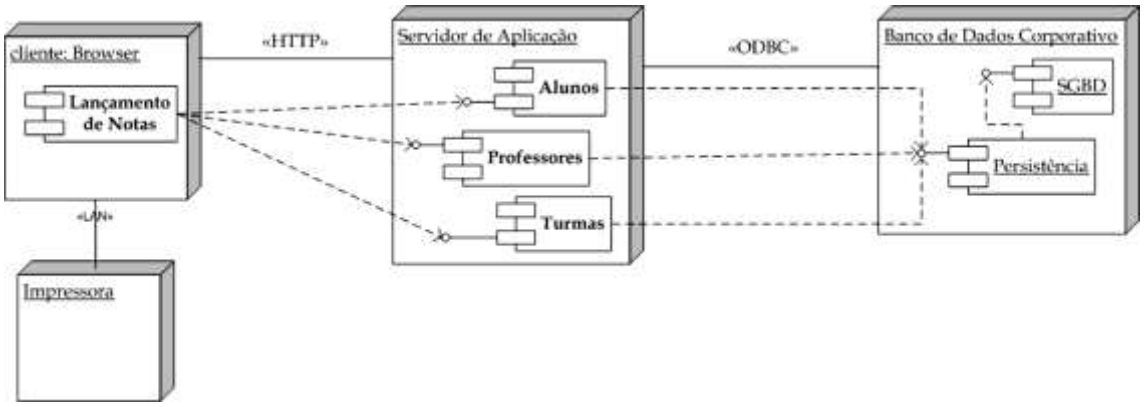

Figura 11 - Diagrama de componentes embutido.

MAPEAMENTO OBJETO-RELACIONAL

O padrão da indústria de software atualmente é o paradigma orientado a objetos, que podem assumir estruturas de dados com elevado grau de complexidade, sendo que a maioria deles, em algum momento, necessita de persistência.

Por outro lado, o padrão das tecnologias de banco de dados ainda é o modelo relacional, ou seja, temos uma estrutura de armazenamento bidimensional denominada tabela.

O que fazer para integrar os dois padrões?

Vamos utilizar o diagrama de classes como modelo conceitual do banco de dados e, a partir de então, gerar o modelo lógico de banco de dados, criando as tabelas necessárias às persistências exigidas pelos objetos. Esse processo é denominado mapeamento objeto-relacional. A Figura 12 ilustra um exemplo de associação muitos-para-muitos em um diagrama de classes, cujo mapeamento gera as três tabelas descritas na Figura 13.

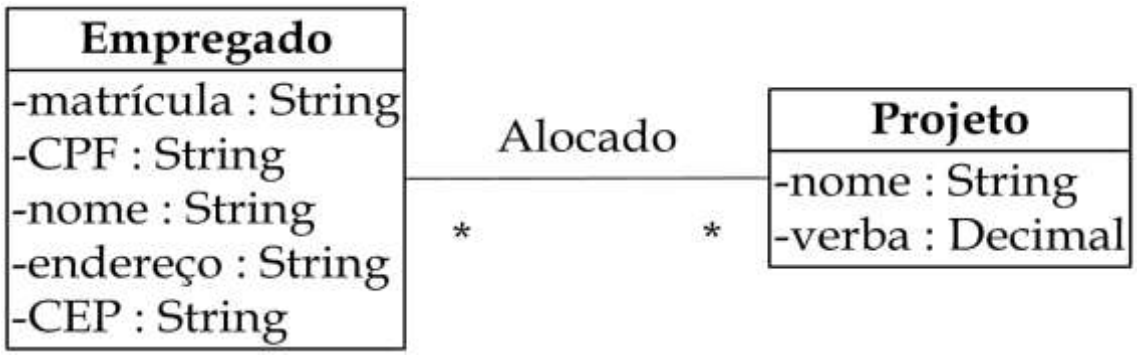

Figura 12 - Exemplo de associação muitos-para-muitos.

Empregado(id,matrícula,CPF,nome,endereço,CEP,idDepartamento)
Alocação(id, idProjeto, idEmpregado)
Projeto(id, nome, verba)

Figura 13 - Resultado do mapeamento muitos-para-muitos.

Realização do projeto da interface gráfica com o usuário

O diagrama de casos de uso define as funcionalidades do sistema, sendo que cada caso de uso representa uma interação completa entre o sistema e um agente externo denominado ator. Nessa interação, precisamos definir uma interface como meio de comunicação homem-máquina. Nesse ponto, o requisito-chave não funcional é usabilidade, portanto é fundamental a participação do usuário no processo de definição das interfaces, ou seja, “deixe o usuário no comando”.

Vamos apresentar possíveis questões relativas à interface que podem surgir: tempo de resposta – em um sistema de venda online com alto tempo de resposta, talvez ocorra uma desistência e uma ida para o concorrente; recursos de ajuda, disponíveis sem que haja necessidade de abandonar a interface; tratamento de erros, por meio de uma linguagem inteligível para o usuário; acessibilidade da aplicação, com foco no atendimento aos usuários com necessidades especiais.

Resumindo

Neste módulo, pudemos avaliar a importância da etapa de projeto no processo de desenvolvimento de software.

Cabe ao engenheiro de software implementar as seguintes atividades que genericamente compõem a etapa de projeto:

- **Refinamento dos aspectos estáticos e estruturais do sistema.**
- **Detalhamento dos aspectos dinâmicos do sistema.**
- **Detalhamento da arquitetura do sistema.**
- **Mapeamento objeto-relacional.**
- **Realização do projeto da interface gráfica com o usuário.**

Ao final desta etapa, podemos iniciar a implementação do software de acordo com as especificações contidas nos modelos de projeto gerados.