Análise e Design

Orientados a Objetos

Hélio Engholm Jr.

Novatec

Editor: Rubens Prates
Capa: Carolina Kuwabata
Revisão gramatical: Marta Almeida de Sá
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-340-6

Histórico das impressões:

Junho/2013 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec
MP20130603

CAPÍTULO 1

Introdução à modelagem orientada a objetos

Historicamente, temos um cenário que registra muitos problemas e falhas em projetos de desenvolvimento de software, como também nos produtos por eles entregues aos clientes que contratam empresas e profissionais, para o desenvolvimento de softwares customizados para suas necessidades específicas. Entre essas falhas podemos citar a demora na entrega dos artefatos do projeto, a falta de qualidade desses, as dificuldades encontradas na manutenção do software entregue, o desempenho desse aquém das expectativas dos contratantes e a falta de confiabilidade nos resultados apresentados. Esse histórico demonstra a necessidade de melhoria nos processos de desenvolvimento de software utilizados pelas empresas de tecnologia da informação contratadas para esse fim, como também na qualidade dos produtos por elas desenvolvidos e entregues.

As empresas desenvolvedoras de software necessitam de mecanismos que permitam maior produtividade e qualidade dos produtos desenvolvidos. Existem diversas disciplinas que, em conjunto, permitem obter esses objetivos. Este livro aborda as disciplinas relacionadas à análise e ao design orientados a objetos, que são extremamente importantes no desenvolvimento de software mais robusto, eficaz e com maior qualidade e extensibilidade.

São apresentados conceitos introdutórios de modelagem de sistemas e vários aspectos considerados nas fases de análise, design e implementação de software, fases essas contidas em metodologias de desenvolvimento de software utilizadas no mercado.

Com esta apresentação, tenta-se demonstrar a importância da utilização de modelagem e design na construção de sistemas ao apresentar as vantagens de utilizá-los juntamente com as melhores práticas de mercado relacionadas ao desenvolvimento de software. Também procura-se demonstrar a importância do levantamento e gerenciamento eficaz dos requisitos nos projetos para que esses sejam bem-sucedidos.

O desenvolvimento de software sem utilização de processos definidos, orientação a objetos e melhores práticas pode trazer uma série de consequências negativas ao produto final do desenvolvimento de software. Entre elas, podemos citar:

- softwares de difícil manutenção, tanto corretiva quanto evolutiva;
- dificuldade de se reutilizar código que foi desenvolvido, permitindo a reutilização de código mal-elaborado, sujeita à geração e propagação de erros em outras partes do sistema em desenvolvimento;
- sistemas com baixa performance e escalabilidade inadequada;
- baixa eficiência no desenvolvimento, com analistas desenvolvendo as mesmas funcionalidades diversas vezes;
- falta de confiança nos dados apresentados pelo sistema, fazendo com que usuários deixem de utilizar o sistema por não confiarem nas informações apresentadas;
- baixa qualidade do código-fonte;
- implantação de sistemas repletos de erros, constantemente entrando em manutenção e interrompendo o trabalho dos funcionários, atendimento a clientes e/ou funcionalidades disponibilizadas a clientes;
- sistemas que podem não atender às expectativas dos stakeholders;
- softwares com alto custo de manutenção;
- softwares de difícil utilização, que não atendem às necessidades dos stakeholders;
- grande insatisfação dos usuários;
- prejuízos de imagem junto a clientes;
- fracasso do projeto.

Nessa relação de consequências negativas, a depreciação da imagem dos profissionais envolvidos e da empresa em que eles trabalham possui um valor imensurável, podendo trazer grandes dificuldades na intensificação e prospecção de novos negócios para todos os envolvidos.

Um dos objetivos primordiais deste livro é o de aumentar a competência dos leitores na entusiasmada tarefa de se realizar o design eficaz de sistemas, produzindo-se um software de maior qualidade, reutilizável e escalável.

1.1 Perspectiva do cliente

Clientes geralmente estão preocupados com os benefícios que os sistemas trarão ao seu negócio e ao seu dia a dia, considerando quais funcionalidades esses sistemas oferecem, se existirá redução de custos operacionais com a sua utilização e de que maneira eles irão agilizar e facilitar seus trabalhos diários.

A princípio, eles não possuem e nem desejam possuir conhecimentos técnicos relacionados ao processo de desenvolvimento de sistemas. Por isso cabe aos profissionais de TI garantir que a equipe de desenvolvimento e de qualidade tenha todas as informações necessárias e uma visão clara do que deverá ser desenvolvido, para que seja entregue ao cliente a solução desejada e com a qualidade por ele esperada.

Além dos requisitos de negócio definidos pela área de negócios das empresas, os clientes também desejam aqueles requisitos que são conhecidos como não funcionais, apesar de a maioria deles não conhecer essa definição. Requisitos não funcionais são aqueles que impõem restrições e definem atributos de qualidade ao sistema, tais como velocidade de processamento e respostas rápidas, facilidade na manutenção do código, capacidade de atender a centenas ou milhares de usuários conectados ao mesmo tempo. A seguir enumeramos os requisitos não funcionais normalmente conhecidos e percebidos pelos usuários finais.

- **Usabilidade**: considera a facilidade de uso do sistema pelos usuários.
- **Confiabilidade**: considera o quão confiável é a utilização do sistema, representando a confiabilidade das informações apresentadas e

mostrando se ele possui mecanismos de proteção para situações de contigência, como backup de dados e tempo de retorno à operação.

- **Desempenho**: relacionado ao desempenho esperado do sistema.

Para um projeto ser bem-sucedido, a equipe que o desenvolve deve ter o escopo do projeto muito bem definido. Aqui entra uma área de conhecimento muito importante para que projetos sejam bem-sucedidos: o levantamento e gerenciamento de requisitos juntamente com o controle efetivo de mudanças.

Para o cliente é importante que o sistema atenda às suas necessidades, realizando suas funcionalidades rapidamente e produzindo resultados confiáveis. Para ele não importam os aspectos técnicos envolvidos no desenvolvimento da solução nem os aspectos de manutenção do sistema em produção.

Imagine que o cliente precisa verificar se o seu cliente tem alguma restrição no Serasa. Ele apenas deseja digitar o CPF dessa pessoa em alguma interface e executar a consulta. Geralmente ele não está interessado em saber como essa consulta será tecnicamente implementada, ele deseja apenas ter essa funcionalidade para uso em seu dia a dia e espera que o resultado esteja correto e seja fornecido sempre o mais rápido possível.

Esse exemplo demonstra a necessidade de existir um processo de refinamento de requisitos, em que o analista de requisitos deve especificar cenários operacionais com base nas necessidades dos interessados, analisando-se o documento de visão e os requisitos funcionais. Os cenários operacionais representam fluxos completos de operação para os usuários finais.

Existem algumas características relacionadas a sistemas que são desejadas pelos clientes em geral. Dentre elas, podemos citar:

- menor custo de propriedade;
- menor tempo de espera para se obter o sistema;
- maior qualidade do sistema;
- fidedignidade das informações apresentadas e das operações realizadas;
- confiabilidade;
- maior disponibilidade.

Os softwares possuem uma característica intrínseca que aumenta os custos das empresas ao adquirir, utilizar e manter sistemas em operação – conhecida como custo total de propriedade. Esse custo é conhecido em inglês como TCO (Total Cost of Ownership) e é aquele que a empresa assume ao manter um sistema em utilização por seus funcionários. Imagine uma empresa que pretende utilizar determinado software em 500 estações de trabalho, sendo que esse sistema deve ser instalado e configurado em cada uma delas por um analista de suporte. Agora imagine que a instalação leva cerca de uma hora e que a empresa não utiliza nenhuma solução de mercado para instalação e atualização automática de software. Nesse cenário, torna-se fácil imaginar o custo envolvido na instalação e configuração desse aplicativo para uso na empresa.

Agora imagine que o sistema apresenta algum problema de funcionamento ou que foi gerada uma nova versão contendo novas funcionalidades ou alterando as já existentes. Novamente será necessário contratar profissionais para realizar a atualização em todas as máquinas. Isso faz parte do TCO, e todas as empresas têm por objetivo diminuir esse custo.

Outro detalhe é que os clientes nunca querem ou nunca podem esperar muito tempo para que a solução fique disponível para ser utilizada, deixando clara a necessidade de usar procedimentos para agilizar o desenvolvimento e a manutenção de sistemas.

A disciplina de análise e design orientada a objetos pode auxiliar significativamente a equipe a atingir objetivos comuns a todos os clientes.

1.2 Perspectiva dos analistas e desenvolvedores

Ao contrário dos clientes e das áreas de negócio, os analistas e desenvolvedores envolvidos no projeto precisam, além da visão clara do problema de negócio que deve ser resolvido, da visão técnica de como o software deverá ser desenvolvido e também do planejamento da solução que deverá ser desenvolvida.

Os analistas envolvidos nesse planejamento precisam visualizar e comunicar os detalhes de seu planejamento e a solução proposta a uma variedade de envolvidos com perfis distintos, desde seus clientes até os envolvidos na construção e nos testes do sistema.

Nas últimas décadas, tivemos várias propostas para documentar, modelar e desenvolver sistemas. Este livro aborda o paradigma orientado a objetos (POO) e a linguagem de modelagem unificada (UML – Unified Modeling Language) em todos os exemplos e nas práticas relacionadas à análise e ao design de sistemas. Esse tipo de modelagem nos permite criar soluções elegantes, compartilhar ideias e acompanhar as soluções em desenvolvimento por todo o ciclo de desenvolvimento de software.

Em empresas que possuem uma metodologia de desenvolvimento de sistemas institucionalizada com responsabilidades definidas, geralmente há desenvolvedores que possuem a tarefa de implementação do software e que deveriam receber especificações completas do que será desenvolvido e de como será desenvolvido.

Essas especificações devem fornecer todas as informações que o programador necessita para elaborar o código e são desenvolvidas por profissionais que devem se preocupar em especificar o sistema para corresponder não só aos requisitos que atendem às áreas de negócio do cliente, mas também aos requisitos não funcionais.

Além dos requisitos não funcionais comentados anteriormente, os desenvolvedores devem se preocupar com outros não visualizados pelos usuários, como por exemplo:

- **Escalabilidade**: como a arquitetura do sistema suporta o aumento da concorrência dos recursos do próprio sistema.
- **Suportabilidade**: existência de diferentes sistemas operacionais ou plataformas, processamento distribuído, diferentes protocolos de comunicação.
- **Restrições de projeto**: associadas a plataformas e/ou tecnologias a serem utilizadas.
- **Requisitos de implementação**: associados a restrições relacionadas ao código ou à construção do sistema.
- **Requisitos de interface**: características especiais para integração com outros sistemas ou de interface operacional com usuários humanos.
- **Requisitos físicos**: associados a requisitos de hardware, tais como configurações físicas de rede.

Na verdade, estamos falando sobre tarefas relacionadas à modelagem e construção de software. Em uma matriz de responsabilidades, cabe aos profissionais de análise e design definir como o sistema deve ser construído para se atingir os objetivos relacionados tanto aos requisitos de negócio quanto aos requisitos não funcionais, sendo que os desenvolvedores devem construir o código utilizando-se das especificações criadas por aqueles profissionais.

1.3 Processo de modelagem. Para que modelar?

O desenvolvimento de sistemas está sempre associado à resolução de um problema ou de uma necessidade em uma situação real, além da automação de processos, por meio de programas de computador. Nesse sentido, a tarefa de representar um fato real para ser tratado por um programa de computador torna-se um trabalho extremamente oneroso e complexo, caso se deseje fazê-lo considerando todos os detalhes do mundo real.

A modelagem nada mais é do que a representação de situações reais usando modelos simplificados que, quando utilizados, trazem resultados eficazes no desenvolvimento do produto final do projeto. Esses modelos simplificados devem nos mostrar o sistema por vários pontos de vista, assim como um projeto de arquitetura, que é apresentado com imagens em diferentes perspectivas.

No processo de modelagem, os profissionais utilizam apenas as características do processo real que são de fato relevantes ao problema, ignorando todas as outras, a fim de deixá-lo o mais simples possível, sem complicá-lo com fatos desnecessários ao problema que deve ser resolvido. Desse modo, o processo de modelagem sempre gera uma visão simplificada dos problemas.

Por meio da modelagem é possível gerar um modelo simplificado e eficiente que represente o problema real que o sistema deverá resolver. Esse modelo deverá servir de base para o desenvolvimento do produto final.

Por exemplo, em um projeto de um sistema bancário, podemos concluir que muitas informações dos clientes são irrelevantes para determinadas funcionalidades, por exemplo, o mapeamento genético dos clientes é irrelevante para todas as funcionalidades conhecidas atualmente nesses

sistemas, não agregando nenhum valor para o negócio do cliente. Desse modo, em nosso sistema e nos dados por ele armazenados, em momento algum será considerado esse mapeamento.

Com relação às ações do objeto cliente, nosso modelo de desenvolvimento vai considerar apenas as ações que agregam valor ao sistema a ser desenvolvido, tais como realizar depósito, sacar dinheiro ou aplicação financeira e realizar transferências monetárias, desconsiderando as ações de andar ou falar, que não agregam valor ao sistema que será desenvolvido.

Considere o modelo como parte da especificação do que deverá ser desenvolvido, mostrando todas as características do produto final que agregam valor a esse. De fato, a modelagem aparece em várias disciplinas das áreas de conhecimento humano e é utilizada por toda a indústria. Lembre-se, por exemplo, da física quando estudamos o modelo de movimento retilíneo uniforme. Esse tipo de movimento somente existirá se for criado artificialmente pelo ser humano, mas pode trazer resultados satisfatórios em algumas situações específicas.

1.4 Processo de Levantamento e Gerenciamento de Requisitos

Os requisitos são importantes desde o início do projeto até a sua fase de homologação. Nenhuma equipe pode iniciar seus trabalhos de construção nem terá condições de avaliar e testar algo de que não tenha conhecimento prévio. Eles devem descrever todos os aspectos relacionados ao produto de software a ser desenvolvido, desde aspectos relacionados às funcionalidades previstas para os usuários até os relacionados à performance, escalabilidade, manutenção, segurança, extensão e usabilidade, entre outros.

O levantamento de requisitos é, na verdade, um processo evolutivo. Esse processo pode ter como fonte inicial uma Declaração de Trabalho ou um Request for Proposal enviado para a empresa, solicitando uma proposta de desenvolvimento de uma solução de software. Desse modo, os requisitos evoluem durante o andamento do projeto, partindo dos requisitos iniciais e evoluindo por meio de entrevistas junto às fontes de requisitos e de mudanças sofridas durante o projeto. Esse processo de evolução do escopo do projeto reforça a necessidade do gerenciamento eficaz dos requisitos do projeto, disciplina prevista no gerenciamento de projetos.

Uma certeza do desenvolvimento de software é que requisitos sempre mudam. Apesar de bem-vindas, as mudanças causam alterações no produto final do projeto e precisam ser gerenciadas adequadamente e aprovadas formalmente pelo cliente antes de serem atendidas. Essa necessidade fez surgir no desenvolvimento de software a disciplina de Gerenciamento de Mudanças, que controla de forma eficaz todas as mudanças solicitadas no projeto. Como ponto de partida, pergunta-se: quem está solicitando a alteração possui autoridade para solicitar mudanças? Caso tenha, essa pessoa deve se apresentar ao responsável pelo projeto no cliente e a quem tem autoridade para tomar decisões, e devem ser avaliados os impactos nos custos, prazos e riscos e também a qualidade da alteração que está sendo solicitada. E somente com o *De Acordo* formal do cliente pode-se então dar início à implementação das mudanças.

Outro aspecto importante relacionado a requisitos é garantir que o escopo do projeto não seja alterado com a inclusão aleatória de requisitos adicionais sem um controle adequado de mudanças. A delimitação das fronteiras do sistema é essencial para prevenir a inclusão de requisitos relacionados a outros sistemas.

Com relação ao escopo de projetos, é preciso se proteger de dois efeitos conhecidos internacionalmente como *Scope Creep* e *Golden Plating*.

Scope Creep e *Golden Platen* ocorrem quando pessoas envolvidas no projeto decidem adicionar descontroladamente funcionalidades e requisitos não previstos no escopo inicial do projeto sem realizar o processo formal de gerenciamento de mudanças. A não utilização dessa disciplina pode permitir que funcionalidades não alinhadas aos objetivos do projeto sejam adicionadas a ele, podendo ameaçar o prazo, a qualidade, os custos envolvidos e a satisfação do cliente. Desse modo, deve-se levar em consideração um lema de mercado: "não importa quão pequeno seja nosso projeto nem quão pequena seja a mudança, o processo de gerenciamento de mudanças deve sempre existir". De acordo com o relatório de 2010 da agência inglesa Arras People, de gerenciamento de programas e projetos, *scope creep* é responsável por 11,8% dos fracassos de projetos. Devemos nos lembrar de que a qualidade do produto que está sendo desenvolvido está diretamente relacionada ao que foi solicitado e especificado pelo cliente.

O processo de levantamento de requisitos deve verificar os objetivos do projeto e todas as particularidades a serem disponibilizadas pelo sistema a ser desenvolvido, com a preocupação de que o sistema atenda às necessidades e expectativas da empresa.

1.5 Paradigma Orientado a Objetos

Utilizaremos o Paradigma Orientado a Objetos nas técnicas relacionadas à análise e ao design de sistemas apresentadas neste livro. O foco desse paradigma não está nos procedimentos que o sistema irá contemplar ou nas informações que ele irá manipular e/ou armazenar, mas, sim, nos objetos que se relacionam para gerar o valor esperado pelo cliente. Esse tipo de abordagem é diferente das abordagens utilizadas no modelo proposto pela análise e pelo design estruturado e da abordagem utilizada em diagramas entidade relacionamento, por exemplo. O Desenvolvimento de Software Orientado a Objetos (DSOO) utiliza esse paradigma nas fases de análise, design e implementação do sistema envolvido no projeto.

A utilização desse paradigma é tida atualmente como a melhor estratégia para reduzirmos a dificuldade recorrente no processo de modelagem do mundo real do domínio do problema, por meio de um conjunto de componentes de software que seja o mais fiel na representação desse domínio.

O capítulo 5 explica os conceitos de orientação a objetos e sua aplicabilidade no desenvolvimento de sistemas.

1.6 UML e o ciclo de desenvolvimento de software

A linguagem unificada de modelagem (Unified Modeling Language – UML) foi criada por Jim Rumbaugh, Grady Booch e Ivar Jacobson, na tentativa de disponibilizar ao mercado uma nova linguagem de modelagem orientada a objetos que pudesse ser adotada como um novo padrão de linguagem de modelagem. Foi aprovada como um padrão pela OMG (Object Management Group) em 1997 e tem sido utilizada desde então em projetos de desenvolvimento orientados a objetos.

A UML fornece diagramas de modelagem que podem ser utilizados nos processos do ciclo de desenvolvimento de software, documentando e especificando o sistema. É uma linguagem utilizada para documentar, especificar e até construir softwares, caso sejam utilizadas ferramentas adequadas para tal como o Enterprise Architect da Sparx Systems.

A seguir serão tratados os diagramas que podem ser utilizados nos processos existentes no ciclo de desenvolvimento de software.

1.6.1 Diagramas de caso de uso

Utilizados nas fases de levantamento de requisitos, análise e design, os diagramas de caso de uso mostram as funcionalidades do sistema e seus usuários, conhecidos na UML como atores do sistema. Esses usuários podem ser usuários humanos ou outros sistemas que estejam integrados ao mesmo, fornecendo ou consumindo funcionalidades e/ou recursos do mesmo.

Devido ao fato de os diagramas de casos de uso mostrarem graficamente todas as funcionalidades a serem contempladas no sistema, eles podem ser considerados como um contrato com o cliente. Depois de devidamente aprovados por eles, tudo o que não estiver representado nesses diagramas não faz parte do escopo do projeto, devendo ser tratado como mudança de escopo, impactando, portanto, nos prazos, custos e riscos do projeto.

A figura 1.1 mostra um exemplo de um diagrama de caso de uso, relacionado a um sistema utilizado em um restaurante fictício. Esse exemplo mostra apenas casos de uso necessários para uma funcionalidade relacionada ao encerramento da conta de uma determinada mesa.

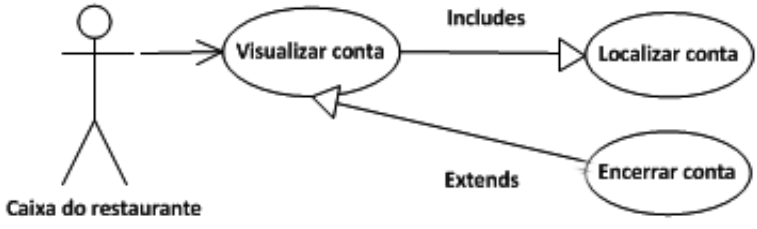

Figura 1.1 – Diagrama de casos de uso de um sistema de restaurante.

1.6.2 Diagramas de atividades

São utilizados na fase de levantamento de requisitos e na fase de design para identificar fluxos de processos dentro do sistema, mostrando o fluxo de controle de uma atividade para outra dentro das funcionalidades do sistema.

Esse diagrama mostra usuários e atividades, permitindo a visualização dos perfis de usuário do sistema. A figura 1.2 mostra um exemplo de diagrama de atividades.

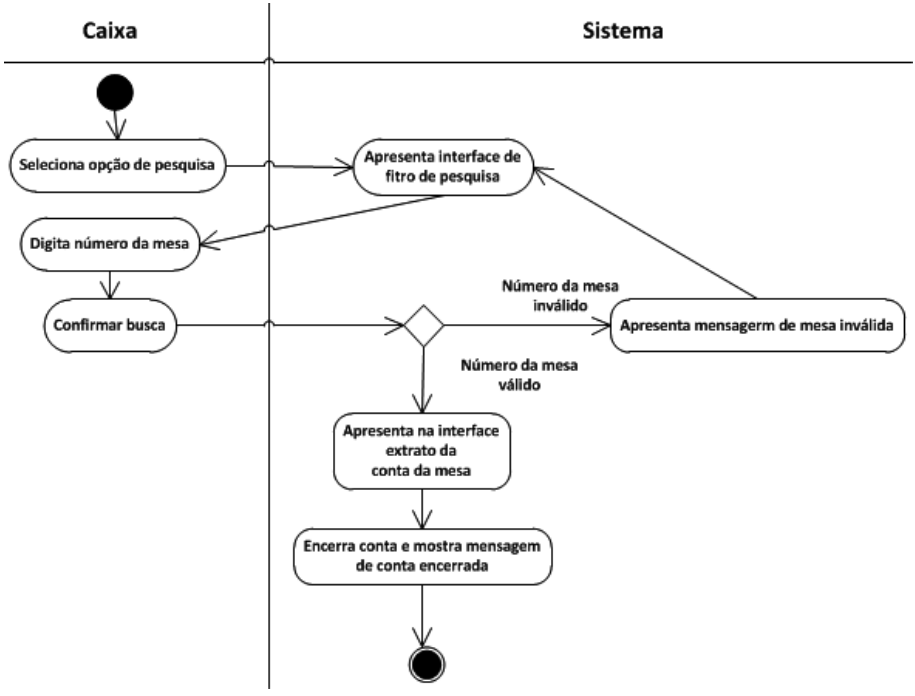

Figura 1.2 – Diagrama de atividades de caso de uso Encerrar conta de restaurante.

Observe como é extremamente simples entender o fluxo de atividades existentes em determinada funcionalidade. Esse tipo de diagrama é facilmente entendido por qualquer pessoa que o visualize. Podemos utilizá-lo para atender a vários objetivos, tais como compreender o que o cliente deseja, documentar o sistema, orientar o programador sobre a maneira que a funcionalidade deverá ser implementada e a equipe de testes sobre como o sistema deverá responder.

Observe que esse diagrama exemplo identifica a matriz de responsabilidades das funcionalidades. Isso permite que o programador e o homologador saibam exatamente quem deverá executar o quê.

1.6.3 Diagrama de classes

O diagrama de classes é um dos mais importantes do paradigma orientado a objetos e pode ter sua definição iniciada já na fase de análise, apenas identificando as classes candidatas. Esse diagrama registra o modelo de domínio da aplicação, especificamente o relacionamento dos objetos de dados com o sistema, o relacionamento entre eles e as operações que cada um deles pode executar. Modelo de domínio é considerado como sendo as abstrações-chave do sistema, mostrando o relacionamento ou a colaboração entre elas. O capítulo 7 apresentará e tratará esse assunto com detalhes.

As classes podem ser vistas como templates de objetos similares, possuindo os dados e as ações dos objetos que elas representam. Desse modo, classes capturam os objetos em detalhes. A figura 1.3 mostra um diagrama de classes candidatas de nossa modelagem do sistema para restaurantes, cujo diagrama de casos de uso foi apresentado no item 1.6.1. Chamamos de classes candidatas devido ao fato de a modelagem ser um processo evolutivo, onde novas classes aparecem e outras podem se mostrar desnecessárias.

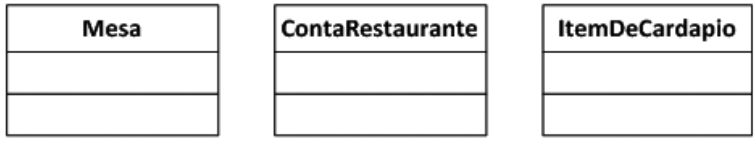

Figura 1.3 – Diagrama de classes candidatas de sistema para restaurantes.

1.6.4 Diagrama de sequência

É algumas vezes utilizado na fase de levantamento de requisitos, mas normalmente é produzido na fase de design para mostrar objetos em movimento e a interação entre eles, ajudando a documentar e a fazer entender o relacionamento entre os objetos do sistema.

Esse diagrama pode ser muito útil na disciplina de testes ao se criar planos de teste e executar testes no sistema. Como todo diagrama UML, ele pode ser tão completo ou tão simplificado quanto se julgue necessário. A figura 1.4 mostra um exemplo desse diagrama em nosso exemplo de modelagem, relacionado ao sistema desenvolvido para restaurantes e lanchonetes. Observe que esse diagrama mostra o relacionamento dos objetos levantados para o caso de uso e as respectivas classes que farão parte do sistema.

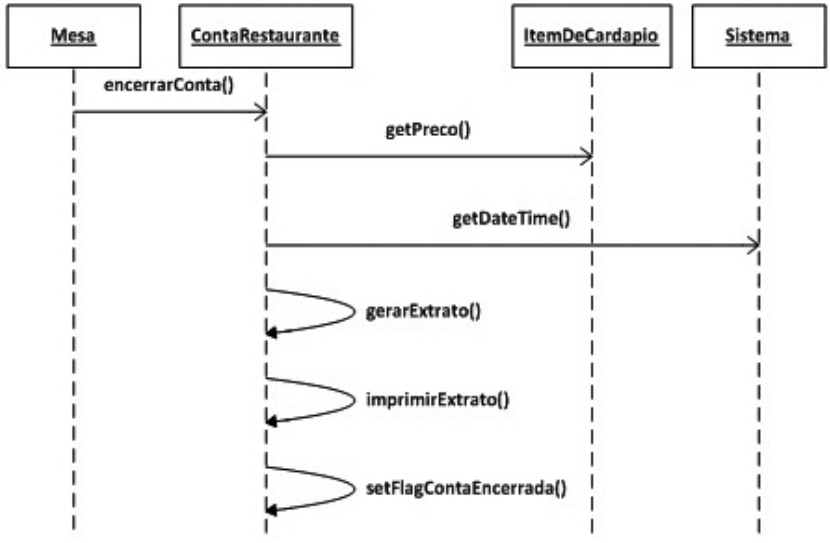

Figura 1.4 – Diagrama de sequência do caso de uso Encerrar conta de restaurante.

1.6.5 Diagrama de colaboração

De maneira semelhante ao diagrama de sequência, o diagrama de colaboração mostra a colaboração dinâmica entre objetos, sendo mais apropriado para os casos onde se desejar dar ênfase ao contexto do sistema. Esse diagrama é mostrado como um diagrama de relacionamento de objetos, dando ênfase às mensagens trocadas entre objetos.

A figura 1.5 mostra um exemplo de diagrama de colaboração de caso de uso relacionado à matrícula de aluno em um curso. Na prática, existem ferramentas de mercado que apenas com um comando transformam o diagrama de sequência no de colaboração, e vice-versa.

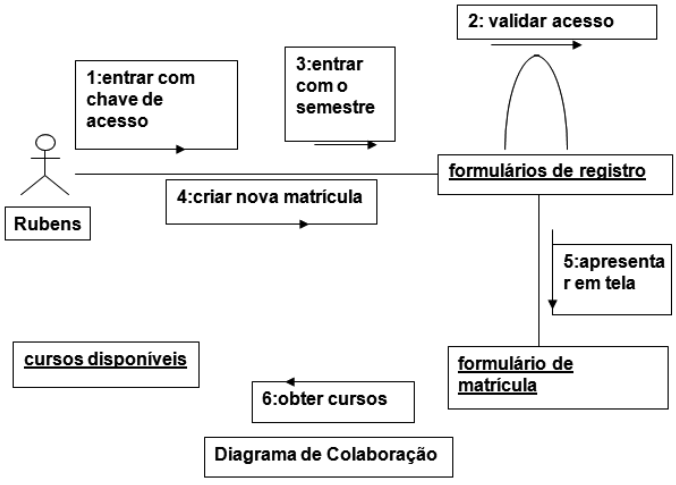

Figura 1.5 – Diagrama de colaboração de cadastro em matrícula de curso.

1.6.6 Diagrama de pacotes

Pacotes são utilizados na organização de modelos, diagramas e objetos/componentes de nossa aplicação. Na implementação dos sistemas orientados a objetos tornam-se espaços conhecidos em inglês como namespaces, responsáveis por armazenar uma série de objetos agrupados para utilização pela aplicação a ser disponibilizada como produto final.

Devemos incluir em nosso código em implementação os pacotes que possuem determinado objeto para podermos utilizá-lo. Como exemplo, imagine que ficou definido no design da aplicação que um determinado componente para validação de CPFs de nome CPFBean deva estar no pacote utilPackage. A fim de utilizar aquele componente, precisamos incluir em nosso código esse pacote para que ele possa ser localizado pela aplicação.

A figura 1.6 mostra um exemplo simplificado de diagrama de pacotes e a figura 1.7, o detalhamento do pacote Sistema, mostrando algumas das classes definidas dentro desse pacote que serão transformadas em objetos para utilização pela aplicação.

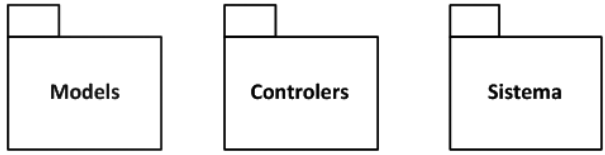

Figura 1.6 – Exemplo de diagrama de pacotes.

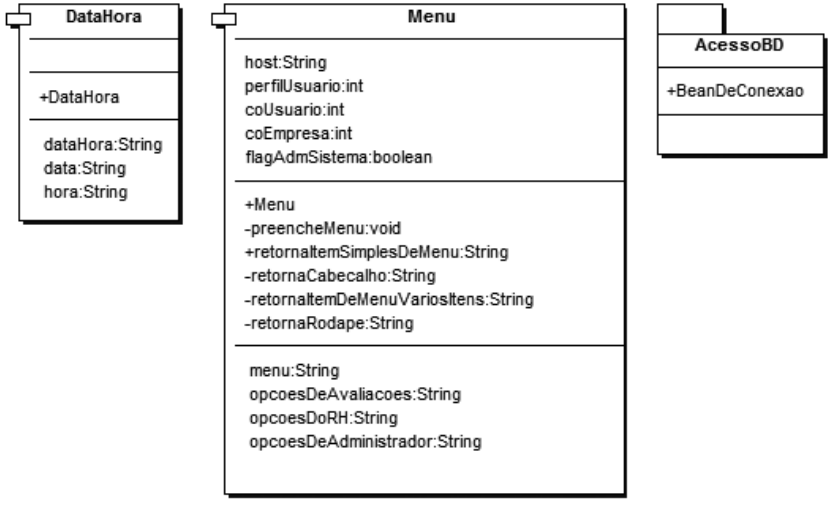

Figura 1.7 – Detalhamento do pacote Sistema.

1.6.7 Diagrama de implantação

Elaborado na fase de design para indicar como o sistema deverá ser implantado e hospedado, mostrando como os recursos serão distribuídos pelos recursos físicos de hardware. A figura 1.8 mostra um exemplo desse diagrama mostrando, inclusive, detalhes de hardware a serem utilizados para implantação do sistema.

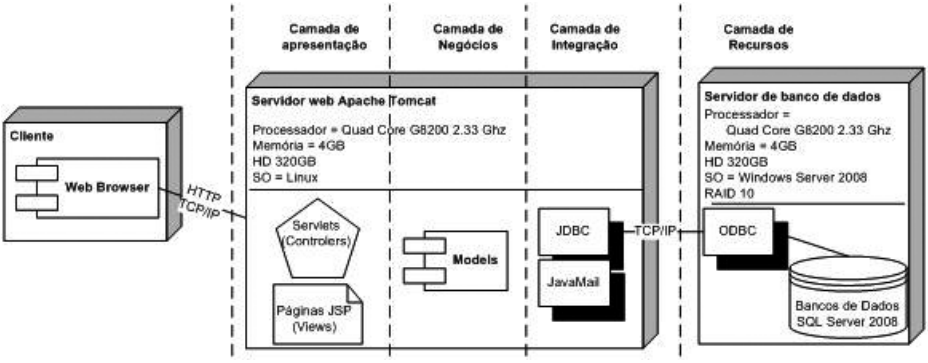

Figura 1.8 – Diagrama de implantação.

1.6.8 Diagramas por fase

Para a fase de análise, os diagramas de caso de uso e de atividades geralmente são mais que o suficiente e ajudam na documentação do sistema e na validação do entendimento do problema a ser atendido pela solução. Já para a fase de design podemos ter os diagramas de classe e sequência, sendo que, quando justificável, também o de estado e de pacotes.

Devemos nos lembrar de que os extremos são ruins. Não ter diagrama nenhum e ter todos os diagramas são extremos indesejados. No primeiro caso, devido à ausência de modelos que representam a solução a serem utilizados no entendimento do projeto, e no segundo, devido ao tempo e ao custo relacionados ao desenvolvimento de todos os diagramas.

1.7 Processo de desenvolvimento de software

O desenvolvimento de software pode ser visto como uma série de transformações que, iniciando no modelo mental dos envolvidos no projeto (stakeholders), resulta no sistema a ser entregue ao cliente. A figura 1.9 mostra a sequência de transformações que, partindo do modelo mental dos envolvidos no projeto, gera o código como produto final.

O ponto de partida de alguns projetos é um artefato conhecido no mercado como Declaração de Trabalho, que mostra em alto nível os objetivos e macrorrequisitos do projeto. Esse documento originado pelo

patrocinador ou iniciador do projeto normalmente descreve as necessidades, os produtos ou serviços que deverão ser providos como resultado final do projeto.

Figura 1.9 – Transformação do modelo mental para o sistema final.

Em projetos externos, esse documento pode ser recebido pelas empresas de desenvolvimento de software como parte de um processo de licitação para contratação do desenvolvimento. Se existente, esse documento elaborado antes de o projeto ser iniciado deve ser considerado como artefato de entrada no processo de levantamento e refinamento de requisitos. O apêndice A apresenta um exemplo de Declaração de Trabalho relacionado ao estudo de caso utilizado nos exemplos do livro, relacionado ao Banco Omega, um banco familiar fictício que se encontra em crescimento e precisa automatizar gradativamente suas operações bancárias.

Processos de desenvolvimento de software tradicionalmente incluem os seguintes subprocessos:

- elicitação de requisitos;
- análise dos requisitos;
- arquitetura;
- design;
- implementação;
- testes;
- implantação.

A seguir serão apresentados detalhes introdutórios das fases de análise, arquitetura e design envolvidas no desenvolvimento de sistemas orientados a objetos (DSOO). A partir do capítulo 6, veremos com detalhes essas três fases que ocorrem nos processos não *Ad hoc* de desenvolvimento de sistemas (Modelos informais utilizados pelo desenvolvedor de software).

1.7.1 Fase de análise

Na fase de análise é preciso ter uma ideia muito clara do problema de negócio a ser tratado e também do que deve ser feito, sem se preocupar com a questão de como o software será implementado e de quais são suas características técnicas.

Entre as diversas tarefas previstas para esta fase, podemos enumerar:

- estabelecer uma visão clara do problema de negócio;
- verificar as tarefas que o sistema deve contemplar;
- entender o vocabulário do cliente;
- levantar os recursos existentes no cliente (plataforma, bancos de dados, sistemas legados que podem necessitar de integração com o novo sistema etc.) ;
- verificar a melhor solução para o problema de negócio;
- definir tarefas.

Já nesta fase podemos utilizar a UML como linguagem para documentar e divulgar as informações adquiridas pela equipe do projeto. Daremos exemplos disso no capítulo 6.

Em uma fase de análise orientada a objetos, podemos identificar os objetos candidatos e a interação entre eles, identificar objetos que possam armazenar regras de negócio do sistema, distribuir o comportamento de funcionalidades entre objetos, identificar aqueles que podem realizar fluxo de eventos como workflows e elaborar o diagrama de classes inicial do sistema. Todo o resultado desse trabalho será utilizado como ponto de partida para a fase de design do sistema.

1.7.2 Arquitetura de software

As mudanças relacionadas à Engenharia de Software têm aumentado a importância da arquitetura de software para garantir aspectos relacionados a requisitos não funcionais, como escalabilidade, desenvolvimento de aplicações distribuídas e performance.

A arquitetura possui um nível de abstração mais alto que o design, o qual se preocupa em projetar os detalhes da aplicação. Enquanto o design se preocupa com os requisitos funcionais, a arquitetura se preocupa com os requisitos não funcionais.

Nesse processo, o arquiteto de software deve propor uma arquitetura para o sistema. Devido ao fato de haver vários tipos de arquitetura, essa decisão deve se basear em quesitos como:

- plataformas utilizadas pela empresa e se elas estarão integradas com a nova solução a ser desenvolvida;
- tipo de plataforma que a empresa deseja utilizar;
- sistemas legados aos quais o novo sistema poderá ou deverá ser integrado;
- tipo de interface que se deseja utilizar para os usuários do sistema;
- onde o sistema será hospedado;
- caso necessário, tipo de distribuição do sistema.

Na seleção e definição da arquitetura, cabe ao arquiteto de software identificar os riscos relacionados à aplicação a ser desenvolvida e analisar as opções de arquitetura, além de analisar as opções que minimizem os custos do projeto sem deixar de atender às expectativas relacionadas a esse projeto. Também cabe a ele garantir que sua proposta de arquitetura atenda aos requisitos não funcionais do cliente, relacionados à confiabilidade, ao desempenho e à escalabilidade pelo sistema a ser desenvolvido. Algumas vezes o cliente prefere correr algum risco do que arcar com os custos associados à eliminação dos riscos identificados. Imagine o custo relacionado à duplicação de um ambiente para minimizar o risco de indisponibilidade do sistema aos seus usuários. Muitas empresas preferem correr o risco de indisponibilidade do que pagar por essa duplicação.

1.7.3 Fase de design

Enquanto na fase de análise verificamos o que deverá ser feito, na fase de design precisamos definir como a solução será desenvolvida para atender às necessidades e expectativas do projeto. Ela deve caracterizar o sistema

com base nos requisitos de tecnologia, selecionando a que melhor atende às necessidades do projeto. O design descreve a solução minuciosamente, especificando detalhes da tecnologia selecionada e de que maneira ela será utilizada. Na abordagem orientada a objetos com UML, devemos produzir nessa fase todas as especificações técnicas que serão utilizadas pelos desenvolvedores na construção do sistema. Essas especificações técnicas podem possuir vários diagramas previstos na UML, sendo que diagramas de atividade, de classes e de sequência geralmente são suficientes para produzir uma especificação com qualidade.

A fase de design deve atender aos seguintes objetivos:

- resolver o problema de negócio;
- verificar elementos de suporte que farão o sistema funcionar;
- definir uma estratégia para implementação do sistema;
- produzir especificações detalhadas do sistema;
- produzir casos de teste.

Na fase de design, o modelo de classes inicial desenvolvido na fase de análise deve ser especificado com grande nível de detalhes. Também deve ser produzido um detalhamento completo da solução (Arquitetura da solução, banco de dados, linguagem de programação, padrões de design utilizados etc.). Como produtos dessa fase podemos ter também diagramas de sequência e colaboração no nível de projeto, diagramas de estado, quando necessário, diagramas de componente e diagramas de distribuição.

Veremos no próximo capítulo como esses subprocessos apresentados se enquadram dentro de algumas metodologias de desenvolvimento de sistemas utilizadas pelo mercado.