ENGENHARIA DE SOFTWARE I

CAPÍTULO 2 - COMO ABSTRAIR INFORMAÇÕES PARA A CONSTRUÇÃO DE SOFTWARES DE ALTA COMPLEXIDADE?

Igor Fernandes Menezes

INICIAR

Introdução

No caminho de aprendizado sobre a engenharia de *softwares*, você já deve ter se perguntado: como funciona a elicitação de requisitos? Neste capítulo vamos compreender as etapas para a elicitação de requisitos de sistemas, para identificar, especificar, classificar, negociar prioridades de execução junto ao cliente, quando for necessário, e aplicar as metodologias de modelagem de sistema dentro da Engenharia de *Software*.

Mas, como especificar requisitos de qualidade? Vamos aprender a mensurar, com qualidade, os requisitos de *software*.

O levantamento dos requisitos é uma das primeiras atividades do desenvolvimento do sistema, seguida pela modelagem, na qual são gerados diagramas, importantes para melhorar a abstração do problema, o que facilita a

construção de uma solução adequada.

Vamos entender como modelar diagramas para o desenvolvimento de *software* e entender o funcionamento da produção, em uma equipe de desenvolvimento de sistemas. E quais são os diagramas que podem ajudar no desenvolvimento de *software*? Quando usar os diagramas? Quais as vantagens da modelagem de diagramas dentro do processo de desenvolvimento? Esses questionamentos vão servir de guia para entender e planejar a arquitetura necessária ao desenvolvimento e funcionamento de um sistema de informação. E também vamos aprender como diferenciar as regras e aplicabilidade dos tipos de servidores na produção do sistema, entendendo as diferenças entre os servidores de desenvolvimento, homologação e produção.

Vamos começar a trabalhar juntos neste assunto, que é muito importante dentro da profissão. Bons estudos!

2.1 Elicitação de requisitos

Para resolver alguns problemas de processos e sistematização de dados e consolidação de informações, o cliente tem a necessidade da construção de um sistema de informação. Essa necessidade se dá, quando o volume de informações é difícil de ser gerenciado, ou a complexidade dos processos deve ser otimizada, para atender demandas em escala.

É neste momento que começa a elicitação de requisitos, ou seja, identificar os requisitos, especificar, classificar e priorizar. Segundo Sommerville (2011, p. 72) “a descoberta de requisitos é o processo de reunir informações sobre o sistema requerido e os sistemas existentes e separar dessas informações os requisitos de usuário e de sistema”.

VOCÊ SABIA?

Para a Engenharia de *Software*, a qualidade do sistema é medida pela concordância da entrega das funcionalidades, após desenvolvidas, em relação aos requisitos de *software*, especificados na fase de análise de sistemas.

Este processo funciona em fases distintas. Vamos entender um pouco mais cada uma delas.

Identificação dos requisitos

Para identificar os requisitos, são aplicadas técnicas (como o *braimstorming*) junto ao cliente, que deve expor todas as necessidades e problemas da empresa, ou do departamento. Normalmente é feita uma reunião entre os *stakeholders*, que são um grupo de pessoas envolvidas no projeto, que têm um interesse em comum, para listar os requisitos de *software*. Nesta fase, também são determinados os requisitos de domínio dos *stakeholders* e da documentação (SOMMERVILLE, 2011).

As técnicas para desenvolver esta fase são variadas e vamos estudar sobre isso mais adiante.

Para garantir a qualidade dos requisitos, é importante conhecer diferentes visões e por isso, devemos envolver profissionais com cargos e experiências diferentes, pois eles vão dar seu parecer sobre um mesmo assunto. Os cargos operacionais dão uma visão do funcionamento de uma determinada área. Os cargos gerenciais podem apoiar soluções em escala, relatórios e formas de otimização de processos. Já os cargos estratégicos precisam ter uma visão gerencial com acesso a informações para tomada de decisão.

Especificação de requisitos

Esta é a fase de elaboração e escrita dos requisitos funcionais, não funcionais e regras de negócio, criando o documento de Análise de Sistema, no qual, as definições ficam registradas. Para Gallotti (2017, p. 14), especificação de *software* é "definir e documentar todos os requisitos necessários para funcionamento do seu *software*". O diagrama de Caso de Uso da UML, que vamos estudar mais à frente, podem ajudar a entender e especificar estes requisitos.

Classificação dos requisitos

Certamente, neste momento, as necessidades e quantidades de requisitos identificados são muitas, e por isso, cabe fazer uma classificação e mensurar as prioridades, junto ao cliente.

A classificação deve ser feita por semelhança e por prioridade. Na primeira, vamos classificar os requisitos semelhantes, ou seja, assuntos parecidos, ou que se relacionam, para que se consiga fazer uma divisão de módulos dentro do sistema.

Já na classificação por prioridade, se define, para cada requisito, uma prioridade, dentre Baixa, Média e Alta. Pode ser que haja algum problema em definir prioridades junto ao cliente – por exemplo, quando ele pensa que tudo é urgente –, e para contornar isso, há técnicas mais eficazes de classificação de prioridades como a de MoSCoW (*Must*, *Should*, *Could* e *Won't*, em português: Precisa ter, Importante ter, Desejado ter e Pouca importância). Na visão de Sommerville (2011, p. 70), a classificação e organização de requisitos

> toma a coleção de requisitos não estruturados, agrupa requisitos relacionados e os organiza em grupos coerentes. A forma mais comum de agrupar os requisitos é o uso de um modelo de arquitetura do sistema para identificar subsistemas e associar requisitos a cada subsistema. Na prática, a engenharia de requisitos e projeto da arquitetura não podem ser atividades completamente separadas.

Priorização e negociação dos requisitos

Neste momento, cabe fazer algumas perguntas, que vão nos ajudar a compreender esta fase.

a) Há, na equipe de desenvolvimento, a quantidade adequada de mão de obra para a quantidade de requisitos identificados?

b) Qual o tempo destinado para o desenvolvimento?

c) Quais são os requisitos, ou módulos fundamentais, para a sobrevivência da empresa?

d) Quais são os requisitos que impactam diretamente na saúde financeira da empresa?

Tendo essas respostas, começa a negociação com o cliente, alinhando as prioridades com a situação atual da empresa, observada nas respostas obtidas.

Aqui, é importante compreender que, com o envolvimento de vários *stakeholders* no processo, é inevitável que os requisitos entrem em conflito. Observe que a atividade é relacionada a essa negociação, pois os conflitos devem ser resolvidos com a participação dos *stakeholders*, que precisam se reunir para chegar a um consenso (SOMMERVILLE, 2011).

Há várias técnicas que podem ser utilizadas para facilitar o levantamento de requisitos, como: entrevistas, reuniões, questionários, *brainstorming*, *role playing* e observação direta.

A técnica mais comum para o levantamento de requisitos são as reuniões e entrevistas, nas quais as informações, desejos e necessidades são expostos. Recomenda-se que as reuniões sejam divididas em duas etapas:

- entendimento das necessidades e coleta de informações;
- apresentação e validação da solução.

Para situações nas quais seja necessário calcular informações de forma quantitativa, pode ser aplicado um questionário para os envolvidos.

Outra técnica importante é o *brainstorming*, muito utilizado na engenharia de *software*, para levantamento de informações, com o objetivo de abstrair a maior quantidade possível de requisitos de *software*. De acordo com Melo (2004, p. 54), "*brainstorming* consiste em uma ou várias reuniões nas quais os participantes sugerem ideias, sem que as mesmas sejam criticadas".

Nesta técnica, é aconselhado ao condutor principal da apresentação, não expressar seu ponto de vista, evitando formar opinião dentro do grupo. É importante que, nesta reunião, participem pessoas de departamentos e formações diferentes, possibilitando enxergar o problema de ângulos variados.

VOCÊ QUER LER?

Bem aplicada, a técnica de *Brainstorm* pode garantir o sucesso do levantamento de requisitos. Você pode aprofundar seus conhecimentos sobre ela no livro “O Brainstorm Eficaz, como gerar ideias com mais eficiência” (ESTEVES, 2017). Você vai ter acesso a estratégias de como selecionar ótimas ideias e como trabalhá-las em grupo, definindo os papeis das pessoas envolvidas.

Com a aplicação do *brainstorming* é possível saber as necessidades de cada departamento dentro de uma organização, sendo possível, em momento posterior, definir e priorizá-las, junto ao cliente, para o desenvolvimento do produto de *software*. As necessidades do cliente em relação ao *software*, na engenharia, são caracterizadas por requisitos de sistema.

Para casos de melhoria no sistema ou identificação de comportamento, pode ser aplicada a técnica de observação direta, ou seja, é observada a maneira pela qual o usuário utiliza o sistema e quais suas rotinas, a fim de abstrair informações que podem facilitar a utilização do sistema, ou novas funcionalidades.

2.2 Introdução a modelagem de sistemas

A modelagem de sistemas é uma das disciplinas do processo de produção de *software*, com maior efetividade na fase de elaboração. É um processo abstrato, que exige experiência e conhecimento, representado por modelos, tabelas, gráficos, diagramas ou fluxogramas, de maneira que cada artefato criado, apresente uma visão diferente para a solução funcional que atenda a necessidade do cliente.

Atualmente, a UML (*Unified Modeling Language*) é a principal técnica de modelagem na construção de sistemas de informação. Além de ajudar os analistas de sistemas a entenderem o problema, os modelos podem ser utilizados para comunicação entre os envolvidos na equipe de produção, usuário do sistema e o cliente.

Os modelos podem ser construídos e utilizados em qualquer fase do processo de desenvolvimento de sistemas, mas, em sua maioria, são criados na fase de levantamento de requisitos. Eles ajudam a especificar o que o sistema deve ou não

fazer, levando a abstrações profundas sobre os assuntos.

VOCÊ SABIA?

Em qualquer local do mundo, a notação UML é a mesma. As tecnologias e metodologias que você aprende e aplica no seu país é aplicada em outras empresas, em qualquer país, mudando apenas o idioma.

Existem modelos diferentes para cada etapa do processo de desenvolvimento. Modelos para entender o problema e o negócio, para especificar os requisitos, modelos lógicos, que podem ser utilizados na codificação e modelos para a fase de implantação do sistema, permitindo sua implantação parcial ou total.

2.1.1 Perspectiva de modelos

Ao modelar um sistema, você deve ter uma visão e criar modelos em visões e perspectivas diferentes.

O analista de sistemas tem, como objetivo, projetar o sistema conforme as necessidades do cliente e informações repassadas por ele, mas pode ocorrer que as informações estejam incompletas ou indeterminadas, cabendo ao analista de sistemas ter habilidade e expertises para buscá-las.

Dentro da equipe de desenvolvimento também existem outros profissionais, mas com visões de qualidade diferentes, veja alguns exemplos:

a) o gerente de projetos busca estimar recursos, prazos, tarefas, custos e informações de projeto;

b) o programador objetiva que a aplicação funcione de forma estabelecida;

c) o *design* está buscando que a tela fique agradável e que tenha usabilidade para o usuário.

Repare que, para cada profissional dentro da equipe, a perspectiva do projeto é diferente. O caminho para o risco é alto, caso o projeto seja encaminhado para o destino incorreto, e a percepção de cada membro da equipe conta muito neste momento. Para reduzir o risco e alinhar a produção, é necessário o desenvolvimento de modelos em diferentes perspectivas, que, de acordo com Ian Sommerville (2011), se classificam em: modelo de contexto, interação, estrutural e comportamental.

- **Modelo de contexto:** perspectiva externa, que modela o ambiente de negócio e tecnológico, no qual o sistema irá funcionar.
- **Modelo de interação:** interação entre sistemas, componentes, módulos, usuários e negócio.
- **Modelo estrutural:** modelagem da estrutura de arquivos, dados e processos dentro do sistema.
- **Modelo comportamental:** modelagem do comportamento do sistema em relação a eventos aplicados em sua utilização.

Para os modelos apresentados, existem diferentes diagramas, que vamos estudar nos próximos tópicos.

2.3 Modelagem de sistemas

Transformar os requisitos em um sistema funcional é uma tarefa que exige técnicas e metodologias. Para isso, a UML (*Unified Modeling Language* ou Linguagem de Modelagem Unificada) pode nos ajudar, pois é uma linguagem de modelagem de sistema.

Os diagramas da UML podem ser utilizados para modelagem de sistemas estruturais e orientados a objetos. Para auxílio na criação dos diagramas, é necessária uma ferramenta CASE (do inglês *Computer-Aided Software Engineering*), pois já possui os estereótipos adequados para cada diagrama, proporcionando produtividade na elaboração.

Primeiro, é criado um projeto conceitual, que deve ser aprovado pelo cliente, que mostra exatamente o que o sistema fará. Uma vez aprovado, é produzido um detalhamento do projeto, denominado projeto técnico, no qual podem ser incluídos diagramas da UML, de forma a facilitar o entendimento dos desenvolvedores, afim de produzir uma solução que atenda às necessidades do cliente.

VOCÊ SABIA?

O trio de cientistas da computação, Ivar Jacobson, Booch Grandy e James Rumbaugh, uniram suas concepções para criar a UML, que, atualmente é uma linguagem de modelagem difundida para modelar sistemas de informação.

A UML é uma linguagem unificada, ou seja, em qualquer local do mundo são incorporados os mesmo padrões e diagramas, apenas sendo adaptados ao idioma local.

Mas, existem algumas dúvidas comuns no início do desenvolvimento do sistema, vamos listar algumas.

- Quais são as funcionalidades e suas ligações?
- Será utilizada programação estruturada ou orientada a objetos?
- Como será a estrutura de arquivos do projeto?
- Como funciona o negócio do cliente?
- Qual a estrutura física necessária para o projeto?
- Como será entrega das funcionalidades e módulos do sistema?
- Será utilizado algum componente ou classe de terceiro no projeto?

Esses questionamentos são importantes, pois vão nos guiar na execução do projeto. Com os diagramas da UML, fica mais fácil especificar e esclarecer as dúvidas que surgem.

Veja na figura a seguir, a utilização dos diagramas da UML e suas relações.

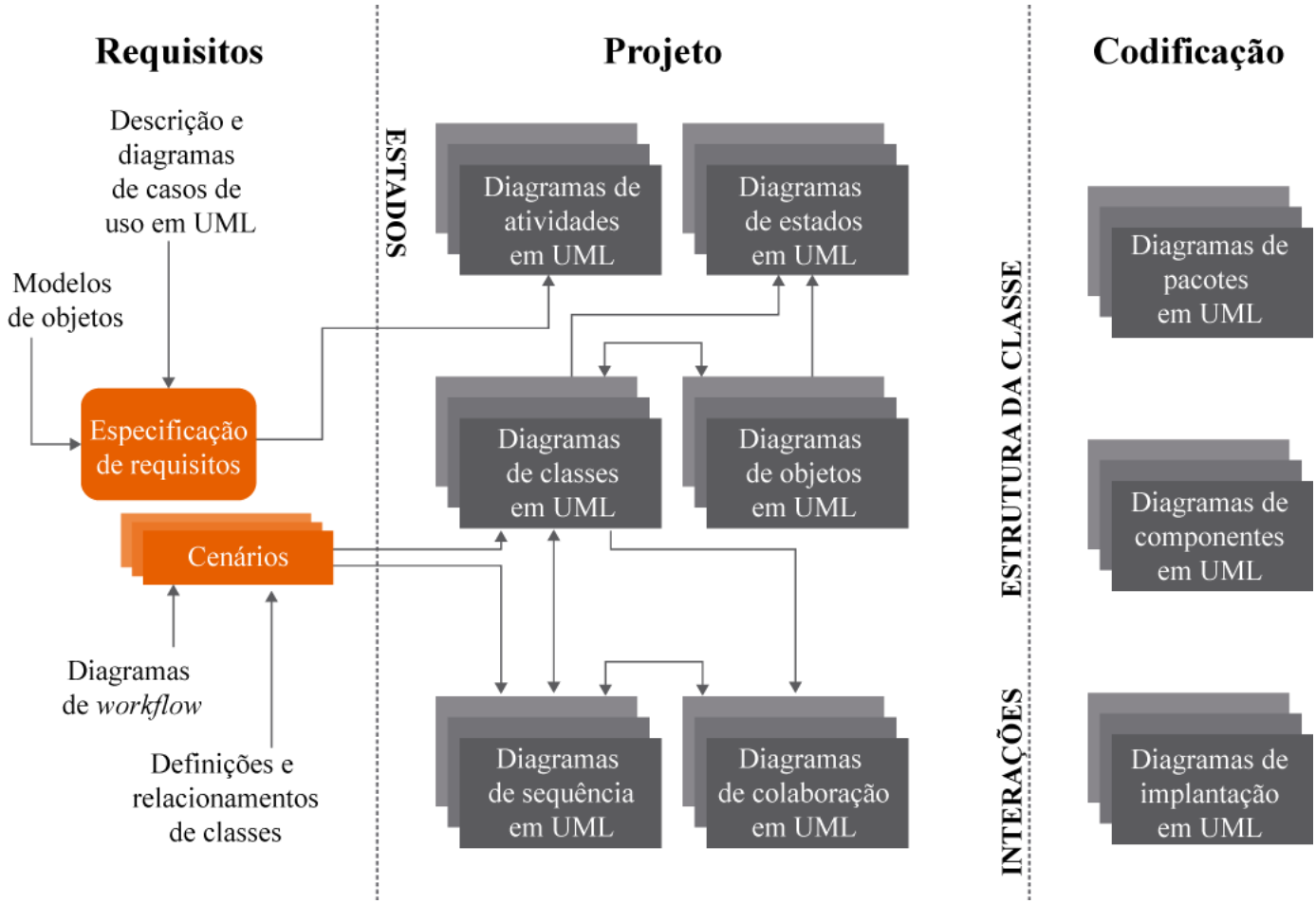


Figura 1 - Diagramas da UML, feitos com base na especificação de requisitos e suas relações dentro do projeto, com outros diagramas. Fonte: PFLEEGER (2004, p. 220).

Quando é feito o projeto de um sistema, podem haver diagramas conceituais e outros mais técnicos. Os diagramas conceituais podem ajudar na definição do sistema e o que ele deve fazer. Já os diagramas técnicos mostram, para os desenvolvedores, como deve ser feito.

O diagrama de Caso de Uso é um exemplo de diagrama conceitual, que pode ser apresentado para o cliente, usuários e desenvolvedores. O diagrama de classes é técnico, serve de orientação no desenvolvimento do sistema.

Os diagramas e desenhos, normalmente, são construídos no início de uma nova iteração. Cabe avaliar a viabilidade de construções. Mas, para todo o sistema que desenvolver é necessário criar todos os diagramas?

Não. A criação do diagrama só é necessária quando se deseja obter mais informações sobre um determinado assunto e, se esse diagrama deixará claro o projeto para todos os envolvidos, de forma a eliminar dúvidas e evitar retrabalho.

Uma das primeiras atividades dentro do projeto é a especificação de requisitos. Com base nos requisitos, se geram os primeiros diagramas da UML (diagrama de atividade, Caso de Uso e de classes).

2.3.1 Diagrama de Caso de Uso

O Caso de Uso descreve, de forma visual, um conjunto de funcionalidades presente no sistema ou que deve ser desenvolvido, com objetivo de apresentar uma parte do sistema, ou todo seu funcionamento.

José Carlos Cordeiro Martins (2007, p. 2) define Caso de Uso da seguinte forma: “um Caso de Uso (*Use Case*) é um tipo de classificador, representada por uma elipse, que representa uma unidade funcional coerente do sistema ou subsistema”. Podemos ver isso, na figura a seguir.

Figura 2 - Exemplo de Casos de Uso para um sistema de instituição de ensino (Caso de Uso “Efetuar Matrícula”) e para um sistema bancário (Caso de Uso “Abrir Conta”). Fonte: Elaborada pelo autor, 2018.

Veja alguns exemplos de possíveis casos de uso para um sistema de vendas:

- gerenciar fornecedor;
- emitir nota fiscal;
- baixar cobrança;
- cadastrar cliente.

Casos de Uso são tipicamente relacionados a "atores". Um ator pode ser um humano, um periférico ou sistema legado, veja alguns exemplos, na figura a seguir.

Figura 3 - Exemplos de Atores de Caso de Uso do tipo pessoa (Gerente), Sistema legado (Sistema Financeiro) e Periférico (Leitora de Código de Barra). Fonte: Elaborada pelo autor, 2018.

Como exemplo de outros possíveis atores, podemos citar: Analista Financeiro; Sistema e Pagamentos; Sistema ERP; Sistema de Estoque; colaborador; impressora de nota fiscal; catraca eletrônica.

O autor Craig Larman (2007, p. 4) define o conceito de ator como: “Um ator é algo com comportamento, tal como uma pessoa (identificada por seu papel), um sistema de computador ou uma organização; por exemplo, um caixa”.

No relacionamento de associação, o Ator gera um estímulo no Caso de Uso.

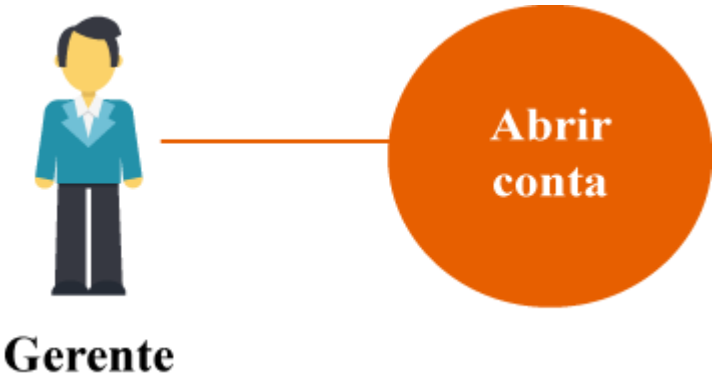


Figura 4 - Exemplo de relacionamento de Ator com Caso de Uso. Fonte: Elaborada pelo autor, 2018.

O diagrama de Caso de Uso é a junção destes elementos apresentados e de outros, como podemos ver no exemplo de um sistema financeiro que deseja controlar as contas a pagar e receber de uma empresa:

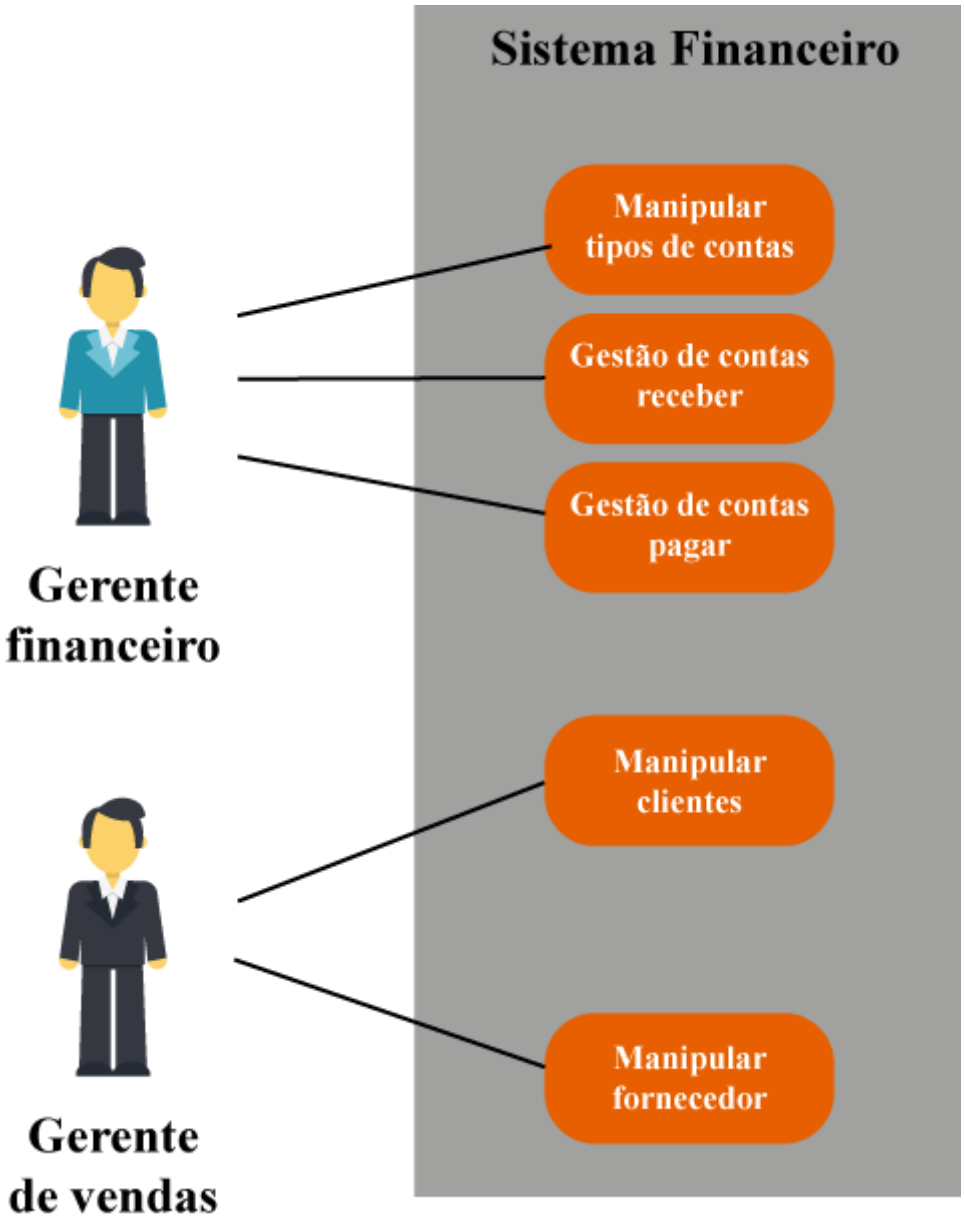


Figura 5 - Exemplo de Diagrama de Caso de Uso, para um sistema gerencial financeiro básico. Fonte: Elaborada pelo autor, 2018.

Como vemos na figura acima, os Casos de Usos são agrupados dentro de um retângulo, que representa a fronteira do sistema, ou seja, os Casos de Usos que estão dentro da fronteira, fazem parte do escopo do sistema, caso ele esteja fora da fronteira, faz parte do sistema, mas não seria responsabilidade de sua equipe fazer o desenvolvimento dessa funcionalidade.

O cientista da computação, Craig Larman (2007, p. 4) defende que o diagrama de Caso de Uso “é uma excelente imagem do contexto do sistema; ele é um bom diagrama de contexto, ou seja, mostra a fronteira de um sistema, o que está fora dele e como o sistema é usado. Serve como ferramenta de comunicação que resume o comportamento do sistema e seus atores”.

O diagrama de Caso de Uso pode ser utilizado como base, para aplicação de métricas de *software*.

VOCÊ QUER LER?

Na dissertação de Yara Freire (2008) "TUCP-M – Pontos de Casos de Uso Técnicos para Manutenção de *Software*", podemos entender como é feita a implementação da métrica de *software* Pontos por Caso de Uso. Acesse e leia: <http://fattocs.com/files/pt/livro-apf/citacao/YaraMariaAlmeidaFreire-2008.pdf (http://fattocs.com/files/pt/livro-apf/citacao/YaraMariaAlmeidaFreire-2008.pdf)>.

Em muitas vezes, o cliente não tem tempo para ler e validar os requisitos de um sistema, e, se caso seja descoberto no futuro, uma alteração nos requisitos, pode ser um risco para o projeto. A representação visual, no formato de diagramas, em muitas vezes, é mais atrativa e de rápido entendimento para o cliente, possibilitando a validação e continuidade do projeto, sem surpresas para o futuro.

2.3.2 Diagrama de classes

O diagrama de classes é um dos mais utilizados e importantes da UML, servindo de apoio para a maioria dos outros diagramas. Ele é construído sempre que o projeto utiliza o paradigma da orientação a objetos em sua construção.

Para Pfleeger (2004, p. 210) "a orientação a objetos é uma abordagem para desenvolvimento de *software* que organiza os problemas e suas soluções como um conjunto de objetos distintos". O diagrama define a estrutura das classes utilizadas pelo sistema, mensurando os atributos e métodos, e apresenta a estrutura estática ou fixa das classes, nas quais ela representa abstrações do mundo real.

Uma classe é composta pelo seu nome, atributos e métodos. O atributo define a característica de um registro ou objeto, e o método define o seu comportamento. Vamos tomar como exemplo a classe "*Cliente*", quais são seus possíveis atributos? Nome, telefone, e-mail, data de nascimento e sexo. Considerando a mesma classe, quais são os possíveis métodos? Cadastrar cliente, excluir cliente, buscar todos os clientes ativos e outros métodos que definem comportamentos e procedimentos para este registro.

O diagrama de classe demonstra as estruturas de programações, arquivos, classes e suas relações. Para Martin Fowler (2005, p. 52), “um diagrama de classes descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos existentes entre eles”. As relações são apresentadas de diversas maneiras:

- associação (conexão simples entre as classes);
- dependência (uma classe depende, ou usa, outra classe), ou seja, para um perfeito funcionamento ou existência, a classe precisa de uma outra classe;
- especialização (uma classe com uma especificidade maior);
- ou em pacotes (classes agrupadas por assuntos de mesma natureza).

O diagrama de classes é considerado estático, pois define sua estrutura. Veja na figura a seguir, um exemplo de diagrama de classes de um sistema de vendas.

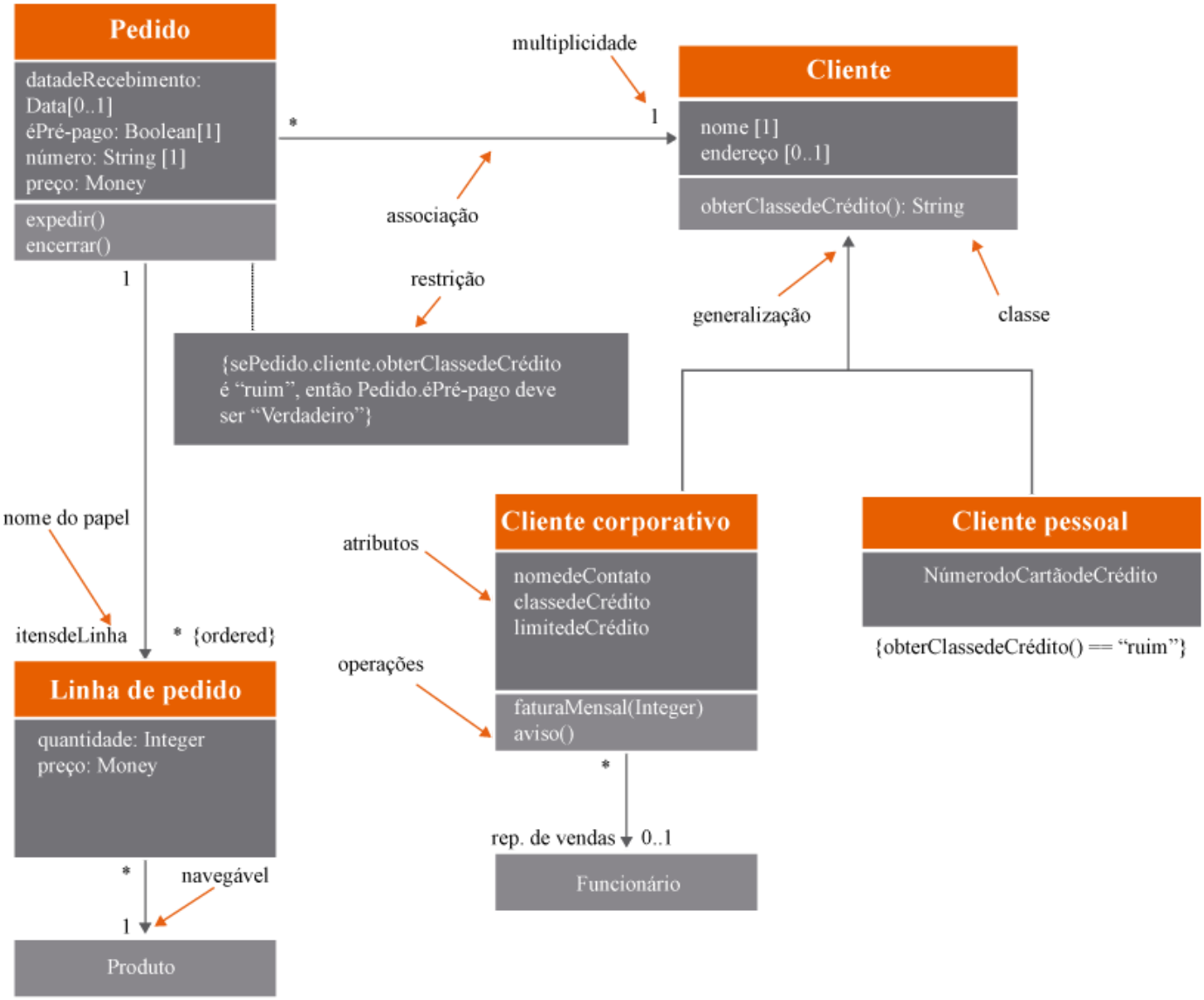


Figura 6 - Exemplo de diagrama de classes, exemplificando a utilização de artefatos para construção.

Fonte: FOWLER (2005, p. 53).

Uma classe do diagrama pode ser implementada utilizando uma linguagem de programação orientada a objetos, que tenha os recursos deste paradigma. Para uma melhor definição dos métodos e atributos de uma classe, se necessário, pode ser criado o diagrama de objetos que representa o momento de utilização da classe pelo sistema.

2.3.3 Diagrama de objetos

Os diagramas de classes e objetos devem sempre ser construídos juntos, pois um completa o outro, em relação, entendimento e construção da solução. O diagrama de objetos possui representações do momento de criação dos objetos das classes,

utilizando a mesma estrutura, acrescentando exemplos de dados em sua utilização. Martin Fowler (2005, p. 94) define que “um diagrama de objetos é um instantâneo dos objetos em um sistema em um determinado ponto no tempo”.

O diagrama de objetos não possui a mesma importância do diagrama de classes, mas deve ser criado ou utilizado para exemplificar a utilização de uma classe, ajudando na sua compreensão e definição.

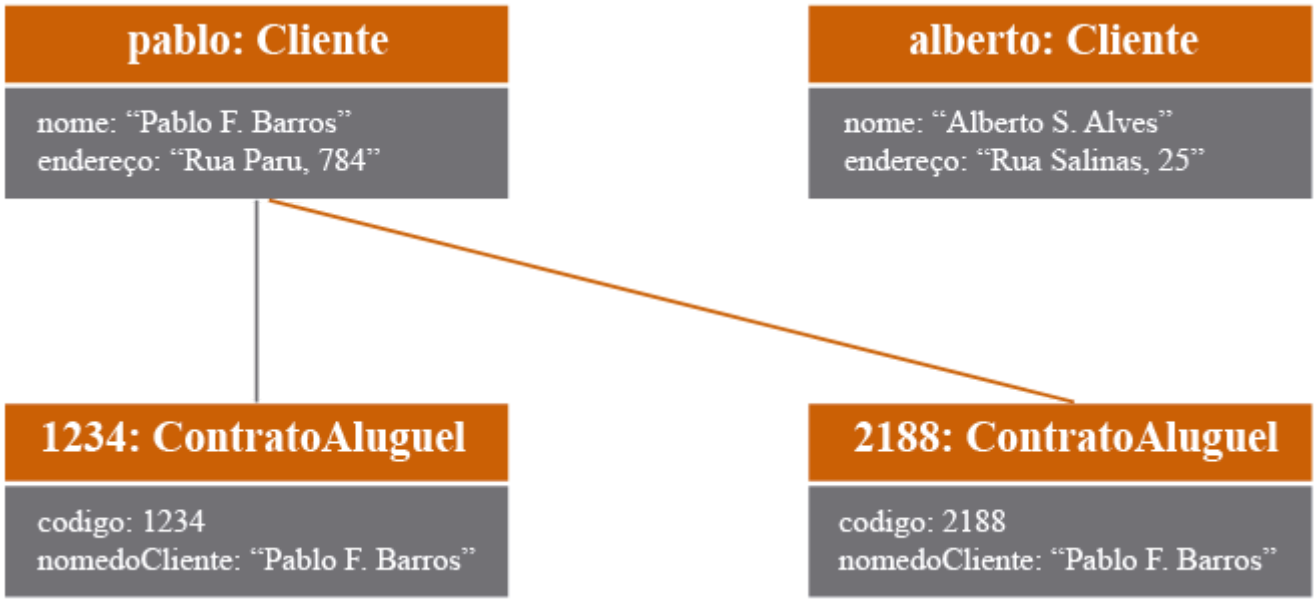


Figura 7 - Exemplo de Diagrama de Objetos, mostrando as instâncias das classes de Cliente e ContratoAluguel. Fonte: Elaborada pelo autor, 2018.

Observe na figura acima que ocorrem duas instâncias da classe *Cliente* e duas instâncias da classe *ContratoAluguel*. Instância se refere ao momento de utilização (processamento) da estrutura da classe. Para a classe de *Cliente*, por exemplo, em um determinado momento de utilização do sistema, o objeto seria representado pelos dados do cliente Alberto e, em outro momento, o cliente Pablo, que por sua vez, possui, ou está relacionado, a dois contratos de aluguéis. O contrato de aluguel pode possuir vários possíveis *status*, como: cancelado, em vigor, vencido, aguardando documentação, entre outros. O fluxo de troca de *status* e regras pode ser definido pelo diagrama de estado.

2.3.4 Diagrama de estado

O diagrama de estado define os possíveis estados de um objeto e suas transições, complementando a abstração do problema definidos pelo diagrama de classes e objeto. O autor Martin Fowler (2005, p. 48) reforça que “um diagrama de estados, o

qual pode ser útil, caso um conceito tenha um ciclo de vida interessante, com vários estados e eventos que mudam esses estados". Para Pfleeger (2004, p. 230), "um diagrama de estado mostra possíveis estados que um objeto pode assumir".

Não é necessário criar os diagramas de estado para todas as classes, somente aquelas que possuem quantidade significativa de estados possíveis (mais que quatro), ou que seja necessário explicar as regras de transição entre os estados.

Veja o exemplo da figura a seguir, de um diagrama de estados, para os pedidos de uma loja virtual. Um pedido é iniciado com o *status* de "*Aberto*", e ele pode sofrer mudança de *status* ao longo das operações de logística da loja, até o objetivo final, que é a entrega do produto para o cliente.

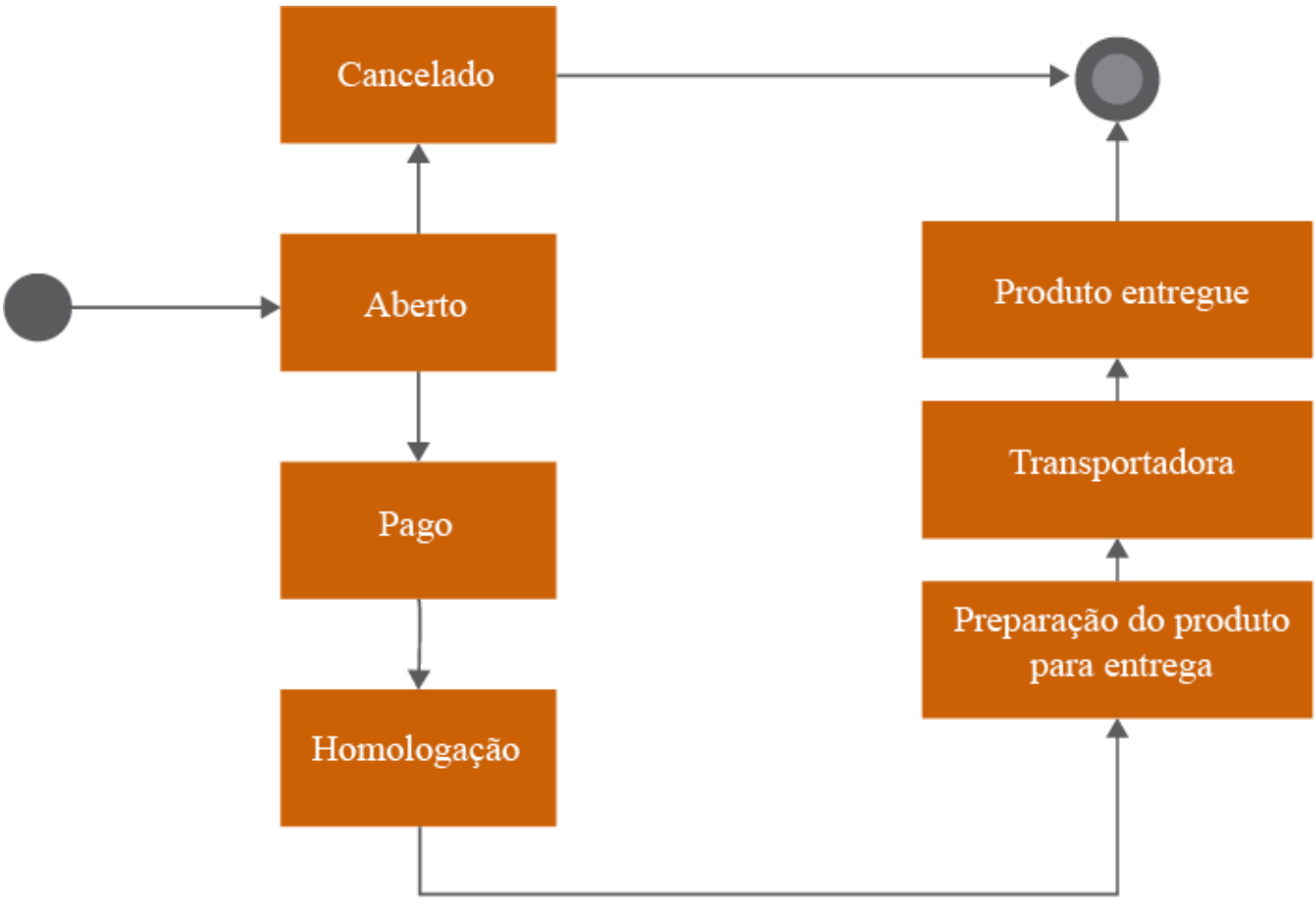


Figura 8 - Exemplo de diagrama de estados demonstrando as mudanças do status de um pedido dentro do sistema. Fonte: Elaborada pelo autor, 2018.

Embora utilize alguns estereótipos iguais, não confunda o diagrama de estados com o próximo diagrama, o de atividades.

2.3.5 Diagrama de atividades

Pelo diagrama de atividades, conseguimos definir um fluxo de um determinado processo. Para Martins Fowler (2005, p. 118), “os diagramas de atividades são uma técnica para descrever lógica de procedimento, processo de negócio e fluxo de trabalho”.

Este diagrama pode ser utilizado para entendimento e definição de um processo de um negócio, objetivando os processos que podem ser sistematizados e os naturais. Segundo Martin Fowler (2005, p. 47) “um diagrama de atividades, o qual pode exibir o fluxo de trabalho da organização, mostrando como o *software* e as atividades humanas interagem”.

Veja na figura a seguir, um exemplo de diagrama de atividades, demonstrando as etapas do processo de compra e os departamentos envolvidos (Cliente, Vendas e Estoque). Observe que, para cada departamento, existem atividades de sua responsabilidade dentro do processo de venda.

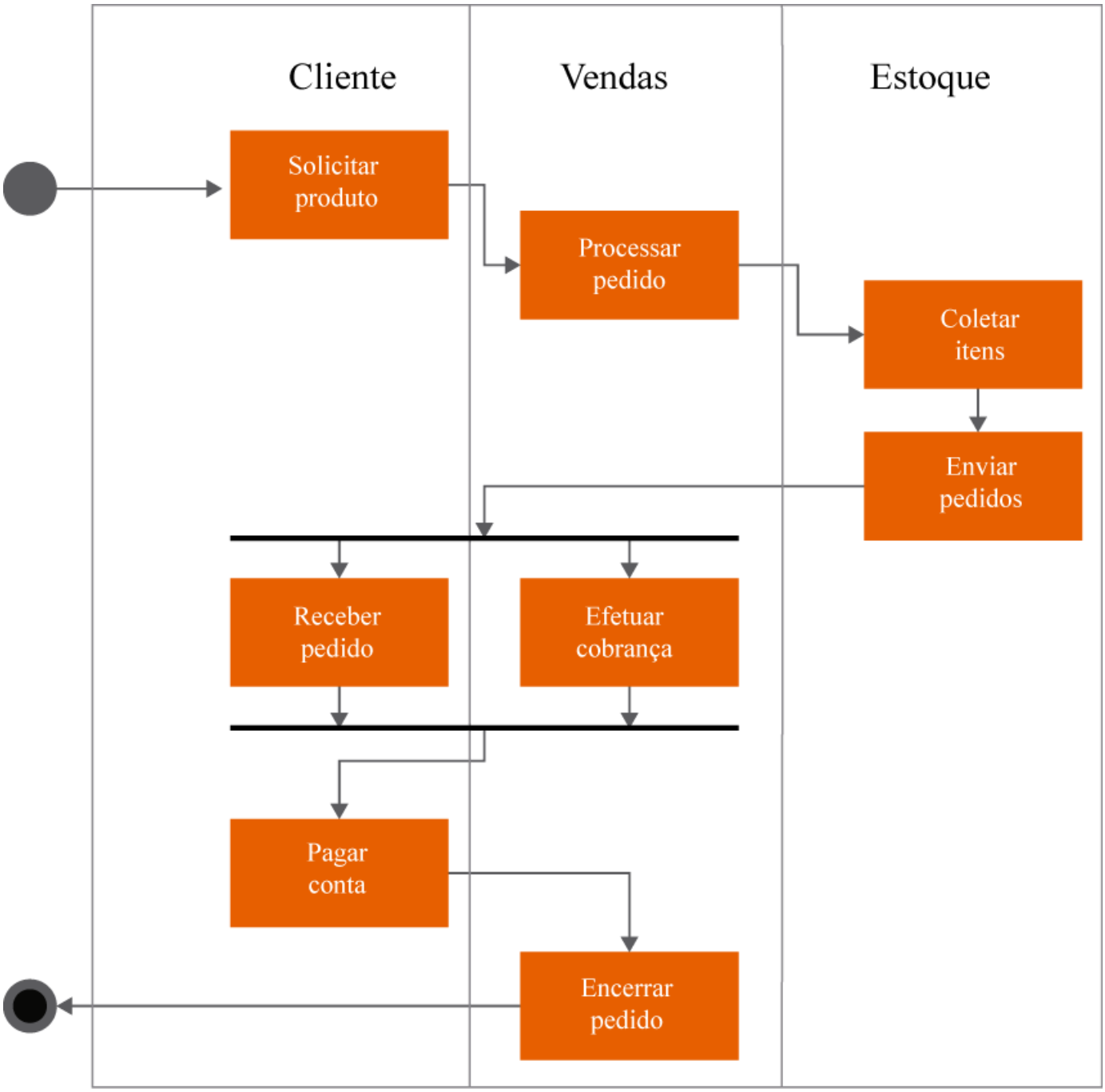


Figura 9 - Exemplo de diagrama de atividades demonstrando as etapas do processo de compra. Fonte: Elaborada pelo autor, 2018.

O diagrama de atividades é utilizado para o detalhamento de um processo, ou procedimento que você não conhece, ou não domina. Em muitas vezes, mesmo sem entender do assunto, somos alocados para um projeto que não dominamos. Quando isso ocorre, certamente o primeiro diagrama a ser construído, antes mesmo do levantamento de requisitos, é o diagrama de atividades. Com ele você vai entender todas as etapas do processo e suas regras, sendo possível determinar atividades que serão executadas em paralelo e condicionais de execução.

;O

ɥue você não tenha carteira de motorista e não domina as etapas necessárias pelas quais um candidato r, para ser habilitado como condutor de veículos de passeio. Com base no texto a seguir, será marcado ıado as possíveis atividades que podem ser utilizadas para a criação do diagrama de atividades, veja:

n deseja ser condutor de veículo de carro de passeio deve procurar uma autoescola (Centro de Formação ores – CFC) que lhe dará todo o apoio e informações necessários nessa caminhada. O candidato deve s exames: psicológico e de aptidão física e mental, realizado em clínicas credenciadas. O candidato deve curso teórico, para aprender as regras previstas no código de trânsito, e deve ter aprovação no exame fim, o candidato deve realizar aulas práticas em simuladores de direção, aulas práticas com o veículo e exame prático, para que seja aprovado no processo de habilitação.

Depois de ler o estudo de caso acima, podemos listar as atividades encontradas:

- procurar uma autoescola;
- exame psicológico;
- exame de aptidão física e mental;
- curso teórico;
- exame teórico;
- aulas práticas;
- simuladores de direção;
- aulas práticas com o veículo;
- exame prático.

Com base na listagem de atividades, é criado o diagrama de atividades, para definir a sequência lógica de ocorrência das atividades e suas respectivas regras.

O diagrama de atividade contempla informações que podem influenciar ou contribuir no levantamento de requisitos de um sistema, ou simplesmente, definir os procedimentos de um processo para um setor ou departamento.

2.3.6 Diagrama de pacotes

Um procedimento natural dentro da Engenharia de *Software* é a divisão do sistema em módulos funcionais. Para descrever e organizar os módulos e suas respectivas dependências, utilizamos o diagrama de pacotes, com a arquitetura de um *software* e o agrupamento de suas funcionalidades em assuntos de mesma natureza. Um pacote representa um grupo de classes, funcionalidades (ou outros elementos), que se relacionam com outros pacotes, em uma relação de dependência (ligação pontilhada e seta aberta).

Um diagrama de pacotes pode ser utilizado em qualquer etapa do processo de desenvolvimento. A divisão do sistema em pacotes, ou módulos, diminui a possibilidade de riscos (PFLEEGER, 2004, p. 93). Veja na figura a seguir um exemplo de divisão de um sistema de faculdade em pacotes ou módulos.

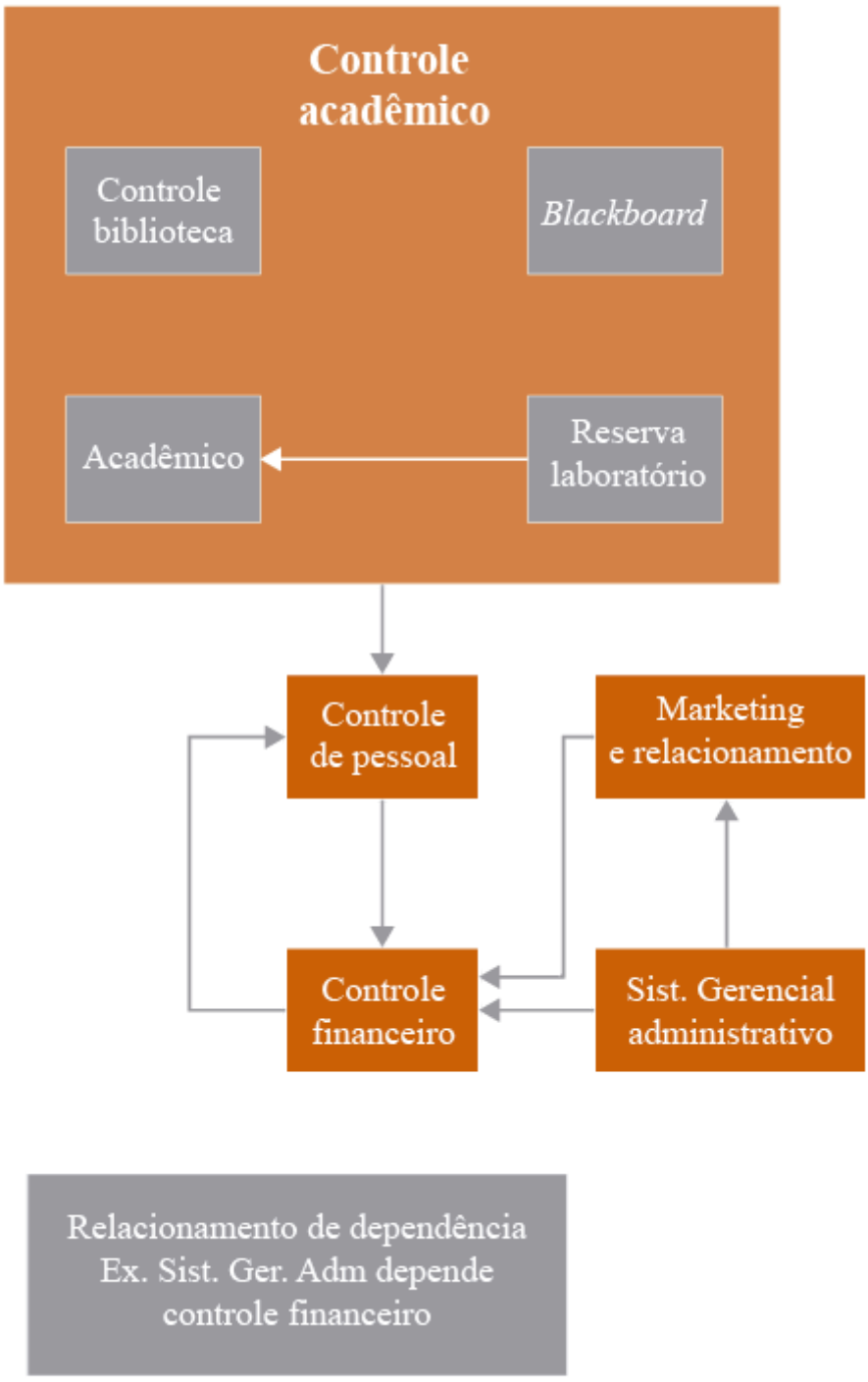


Figura 10 - Exemplo de Diagrama de Pacotes, mostrando os módulos do sistema e suas dependências e relações. Fonte: Elaborada pelo autor, 2018.

Repare na figura acima, o tamanho do sistema utilizado em faculdade. Quando deparamos com a necessidade de desenvolvimento de um sistema como este, de grande porte, a primeira estratégia a ser adotada é a divisão em módulos, para que facilite o desenvolvimento e a entrega de módulos funcionais para o cliente.

A seta pontilhada no diagrama, define que existe um relacionamento e a dependência que um módulo possui em relação ao outro. Como exemplo, observe que o módulo “Reserva Laboratório” possui dependência de informações, que são

controladas pelo módulo “Acadêmico”, neste caso, provavelmente o módulo “Acadêmico” deve ser desenvolvido primeiro.

Vamos praticar um pouco: considere que tenha que criar um sistema para uma grande empresa, com o objetivo de controlar todos os departamentos. Quais os módulos possíveis que você consegue imaginar que deve ter este sistema? Faça uma listagem no papel dos possíveis módulos para este sistema.

VOCÊ O CONHECE?

James Rumbaugh é um cientista da computação e metodologista do modelo de desenvolvimento de *software* orientado a objetos. Ele criou a *Object Modeling Technique* (OMT), e um dos criadores da UML. Reconhecido internacionalmente por trabalhos inovadores relacionados à arquitetura de *software* e a ambientes de desenvolvimento corporativos.

O diagrama de pacotes mostra uma visão ampla e geral do sistema. Para cada módulo, é importante avaliar a necessidade de construção de novos diagramas da UML, como: Caso de Uso, Classe, Atividade, Componentes e outros.

2.3.7 Diagrama de componentes

Um componente pode ser entendido como um encapsulamento de funcionalidades com o objetivo de resolver um determinado problema. Isso, porque, alguns códigos já foram desenvolvidos por uma outra pessoa, ou em um projeto anterior, não sendo necessária a construção do zero (FOWLER, 2005).

Por exemplo, provavelmente no sistema anterior já teria um módulo de cadastro de usuário e autenticação, por isso, não é necessário desenvolver novamente, cabe apenas adaptá-lo ao novo sistema.

Outros recursos também estão disponíveis na internet, geralmente sem custo, que podem ser utilizados sem ter a necessidade de se construir do zero. Por exemplo: classes de *e-mails*, geração de gráficos, boletos, relatórios e integrações com outros sistemas. Estes códigos funcionais prontos, testados e aprovados em outros projetos, podem ser considerados e transformados em componentes de *software*. Basta entender o funcionamento e usar dentro do seu projeto,

minimizando custos e tempo de desenvolvimento. “Use diagramas de componentes quando você estiver dividindo seu sistema em componentes e quiser mostrar seus relacionamentos por intermédio de interfaces ou a decomposição de componentes em uma estrutura de nível mais baixo” (FOWLER, 2005, p. 134).

Um diagrama de componentes da UML ilustra como as classes devem ser organizadas pela noção de componentes de trabalho. O componente é a notação utilizada para representar código reutilizado de uma aplicação. Para Ian Sommerville (2011, p. 317), “os componentes são abstrações de nível mais alto do que objetos e são definidos por suas interfaces”.

Em um projeto, devemos mensurar códigos, documentos e artefatos, que podemos reutilizar de outros projetos, a fim de otimizar o desenvolvimento e gerar benefícios estratégicos na entrega do produto.

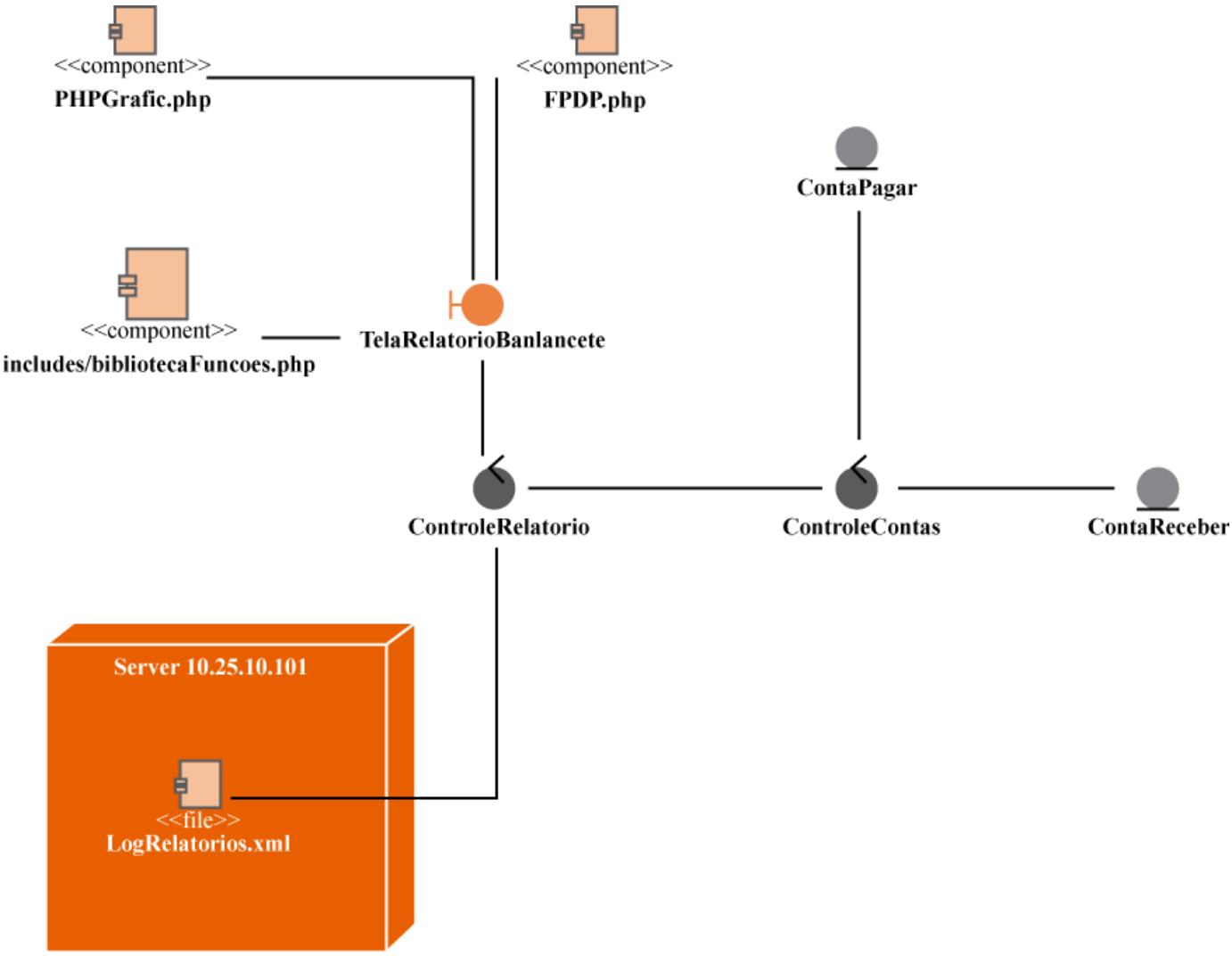


Figura 11 - Exemplo de diagrama de componentes, utilizando bibliotecas de terceiros dentro e fora do servidor e modelagem de classes MVC. Fonte: Elaborada pelo autor, 2018.

Pelo diagrama apresentado na figura acima, percebemos a utilização de bibliotecas/classes de terceiros dentro do projeto, representadas por componentes, sendo: *PHPGrafic.php*, *FPDF.pdf*, *bibliotecafuncoes.js* e *LogRelatórios.xml*.

As classes *PHPGrafic* e *FPDF* foram desenvolvidas por um grupo de pessoas e disponibilizadas para utilização em qualquer projeto. O projeto também conta com bibliotecas já prontas de outros projetos.

É especificado que, para todo relatório gerado, é necessário gravar um *log* dentro do arquivo *LogRelatorios.xml*, dentro do servidor 10.25.10.101. Veja a importância do diagrama na apresentação do funcionamento dos componentes e servindo de apoio para o projeto de arquitetura.

Assim, pudemos compreender os principais diagramas da UML e suas aplicações, mas dentro da metodologia, há outros diagramas com visões e abordagens diferentes dos que foram apresentados.

2.4 Projeto de arquitetura

O processo de projetar a arquitetura do projeto é fundamental para o sucesso. Ele consiste de identificar os subsistemas, componentes, classes de terceiros, comunicações com sistemas legados, estabelecer padrões de projetos e *framework*. Importante estabelecer, no projeto, as informações referentes à arquitetura do *software* na fase inicial do processo de concepção do sistema. Para Pfleeger (2004, p. 25), “a arquitetura completa de um sistema é importante não somente para facilitar a implementação e o teste, mas para a rapidez e eficiência com que poderão ser realizadas alterações e manutenções”.

VOCÊ QUER VER?

O documentário *Revolution OS*, do diretor J. T. S. Moore (2001), pode te ajudar a entender um pouco mais sobre a infraestrutura necessária para a produção de sistemas de informação. O documentário aborda temas como o *software* livre GNUX/Linux e Unix, e tipos de servidores para alguns tipos de aplicações.

Algumas definições de arquitetura podem ser consideradas requisitos não funcionais de *software*. Os requisitos não funcionais definem as características do sistema.

A definição da arquitetura é feita em duas escalas: parcial e total. A escala parcial visa levantar as necessidades de arquitetura para um módulo, ou parte do sistema, já a total determina as características-base para toda a aplicação, ou pode ser chamado, também, de arquitetura de pequena escala e grande escala. “A arquitetura em pequena escala está preocupada com a arquitetura de programas individuais” e “a arquitetura em grande escala preocupa-se com a arquitetura de sistemas corporativos complexos que incluem outros sistemas, programas e componentes de programas” (SOMMERVILLE, 2011, p. 105).

2.4.1 Padrões de arquitetura

A ideia principal da adoção de uma arquitetura de *software* é a possibilidade de reuso das melhores práticas de desenvolvimento de sistemas, possibilitando a produção em escala. Diversos tipos de padrões são adotados, entre eles: Cliente-Servidor, modelo em camadas, arquitetura de duto e filtro, modelo MVC e orientado a eventos.

O padrão MVC (modelo-visão-controlador, do inglês *Model-View-Controller*) é largamente utilizado na produção de sistema. Este padrão consiste na atribuição de responsabilidades para as classes e suas interações.

As classes de Fronteira ou Visão (*View*) tem como responsabilidade a interação com o usuário do sistema, se ela acionada via *mouse*, teclado ou outro tipo de periférico.

As classes de Fronteira têm como objetivo coletar as informações fornecidas pelo usuário, enviar para o processamento (responsabilidade das classes de controle e modelo) e fornecer para o usuário o resultado deste processamento. As classes de Controle ou Controlador (*Controller*) possui como responsabilidade a distribuição das solicitações recebidas pela classe de fronteira e aplicação das regras de negócio. Por fim, as classes de Entidade ou Modelo (*Model*) faz a interação com o banco de dados, aplicando as regras necessárias para a persistência da informação. Essas definições podem ser observadas de forma visual pela figura a seguir.

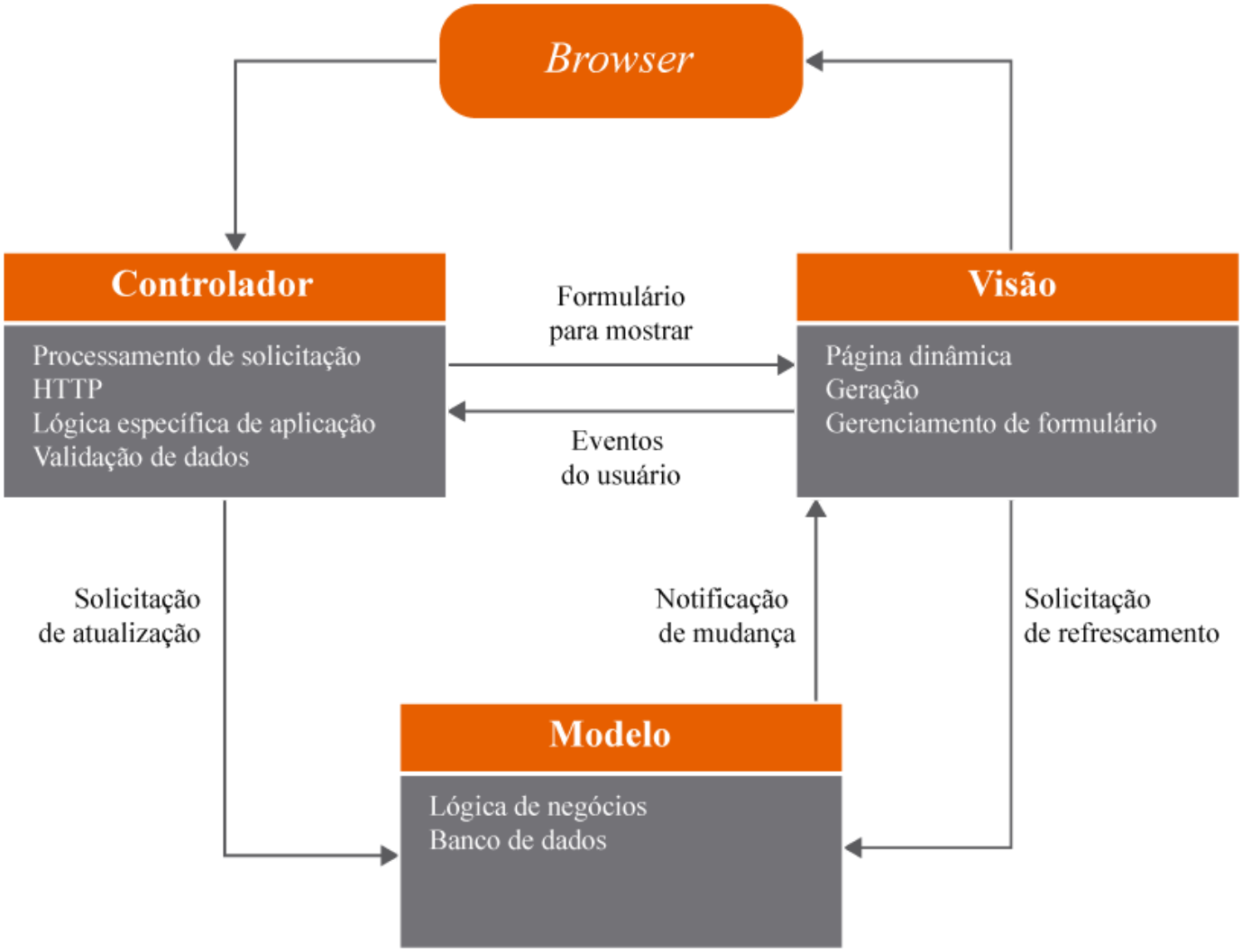


Figura 12 - Modelo MVC, com a atribuição de responsabilidades para as classes e suas interações. Fonte: SOMMERVILLE (2011, p. 110).

Essa abordagem pode ser aplicada no desenvolvimento incremental, ou seja, na medida em que as classes forem construídas, é feita a interação entre elas, e com isso, é possível reutilizar os métodos existentes ou criar novos. Este padrão é reconhecido por adotar três níveis de responsabilidades, como mencionado. Mas também existe a possibilidade de um quarto nível, uma camada entre a classe de modelo e o sistema operacional/banco de dados. Essa quarta camada pode apoiar a utilização de recursos do sistema operação ou gerenciamento do fluxo de transações com os bancos de dados.

A arquitetura cliente-servidor também é um modelo comum de utilização. Neste modelo várias estações de trabalho (cliente) fazem requisições nas quais o processamento da informação é feita no servidor e devolvida para o requisitante. Sommerville (2011, p. 113) afirma que “um sistema que segue o padrão cliente-servidor é organizado como um conjunto de serviços e servidores associados e

clientes que acessam e usam os serviços". A arquitetura pode ser definida com um ou mais servidores, cada um com sua responsabilidade de processamento de um ou mais serviços. Como exemplo de servidores, temos: servidores que oferecem serviço de impressão, gerenciamento de arquivos, servidor de aplicação, servidor de banco de dados e outros que podemos ver na figura a seguir.

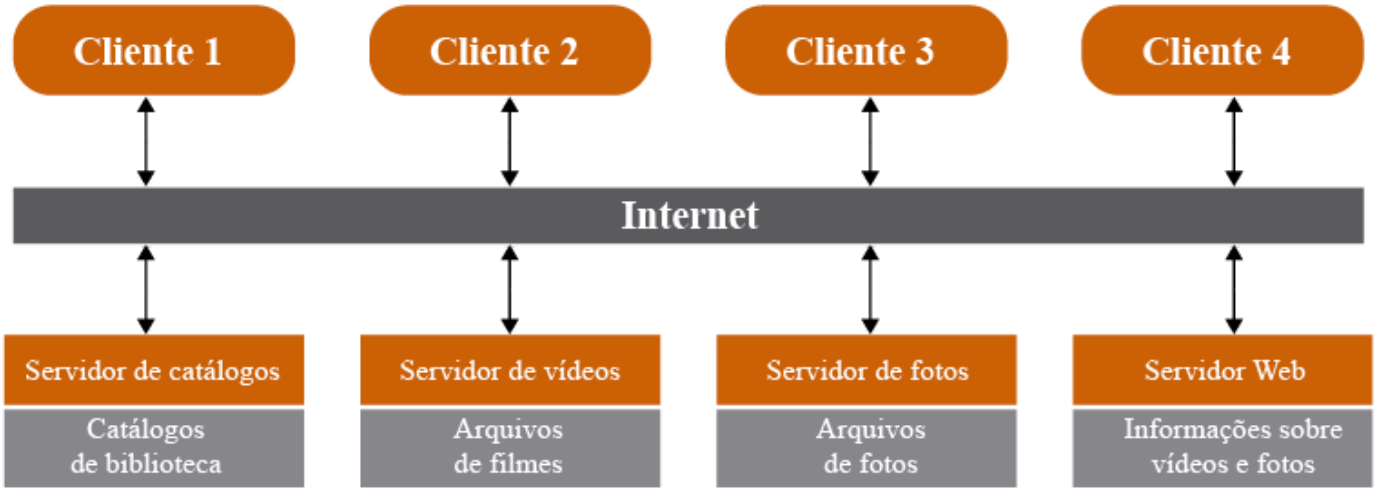


Figura 13 - Exemplo de arquitetura de servidor e consumo de serviços para um sistema de biblioteca de filmes. Fonte: SOMMERVILLE, (2011, p. 110).

Um conjunto de clientes pode chamar os serviços dos servidores, por meio de uma rede, possibilitando a execução de uma ou mais instâncias deste serviço, conforme regras de utilizações aplicadas.

Entre os diferentes tipos de arquitetura, vale mencionar, também, o padrão de dutos e filtros. Modelo de organização de processamento em tempo de execução, no qual são definidas as entradas, ocorre o processamento e se disponibiliza a saída. Internamente no sistema, os dados são processados em formato de sequência, com a mutação das informações, conforme as regras incluídas na programação, produzindo uma saída de informação. Você pode ver essa transformação ocorrendo na figura a seguir, que representa um sistema de pagamentos.

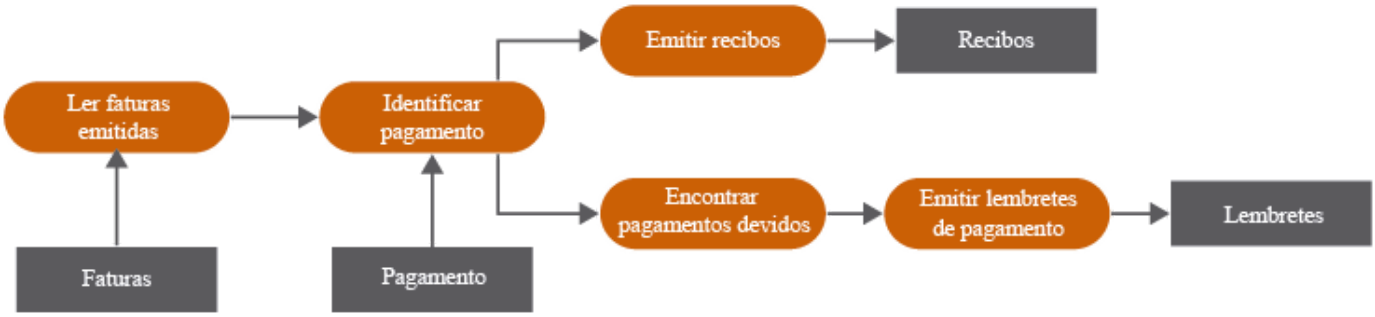


Figura 14 - Exemplo de processamento de informações e suas mutações para um sistema de pagamento.

Fonte: SOMMERVILLE (2011, p. 115).

Perceba, na figura acima, que ocorre a entrada de informação pelas funcionalidades de Faturas e Pagamentos, nas quais as informações são processadas e verificadas, dentro do sistema, a fim de emitir Recibos ou Lembretes de pagamento.

Cada tipo de arquitetura apresentada possui suas vantagens e desvantagens, cabe fazer um estudo do tipo de aplicação a ser desenvolvida para definição da arquitetura que será utilizada.

2.4.2 Vantagens da arquitetura de sistemas

A grande vantagem da definição da arquitetura, é facilitar a comunicação do projeto, auxiliando nas estratégias e discussões de fatores importantes, envolvimento dos analistas de sistemas e equipe, em relação ao cumprimento dos requisitos não funcionais e identificar as componentes reutilizáveis do projeto.

Sommerville (2011, p. 105) lista três vantagens de projetar e documentar a arquitetura de *software*.

> **1. Comunicação de *stakeholders***. A arquitetura é uma apresentação de alto nível do sistema e pode ser usada como um foco de discussão por uma série de diferentes *stakeholders*.
>
> **2. Análise de sistema**. Tornar explícita a arquitetura do sistema, em um estágio inicial de seu desenvolvimento, requer alguma análise. [...]
>
> **3. Reúso em larga escala.** Um modelo de uma arquitetura de sistema é uma descrição compacta e gerenciável de como um sistema é organizado e como os componentes interoperam.

O projeto de arquitetura é um processo criativo, assim, o processo difere, de acordo com o tipo de sistema que está sendo desenvolvido. No entanto, uma série de decisões comuns abrangem todos os processos de projeto e essas decisões afetam as características não funcionais do sistema.

2.4.3 Decisões de projeto de arquitetura

Uma série de decisões deve ser tomada para a execução do projeto de desenvolvimento de *software*. Vamos, novamente, levantar alguns questionamentos importantes para a definição da arquitetura de sistemas.

a) Existe alguma arquitetura existente e pronta que pode ser incorporada a este projeto?

b) Como é a distribuição de processamento dos servidores da aplicação e banco de dados?

c) Quais estilos de arquitetura são apropriados?

d) Que abordagem será usada para estruturar o sistema?

e) Como o sistema pode ser dividido em módulos funcionais?

f) Quais estratégias de pontos de controle serão utilizadas?

g) Como o projeto será avaliado?

h) Como o projeto e a arquitetura serão documentados?

Esse levantamento contempla informações importantes para a elicitação de requisitos, proporcionando habilidades de análise e validação, e priorizando-os, quando necessário.

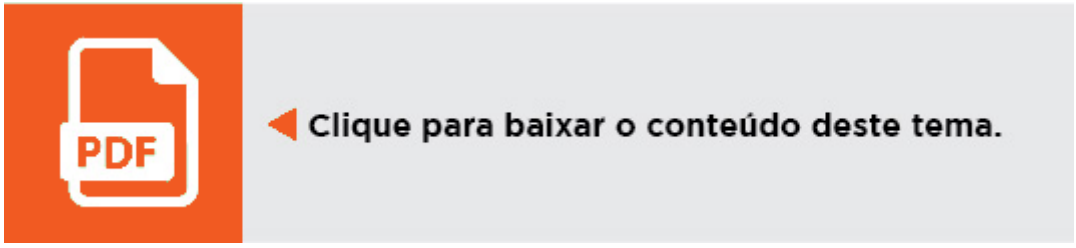
Os diagramas, que vimos neste capítulo, podem facilitar a abstração dos requisitos em um projeto, com a compreensão das diferenças entre eles e suas características. Vimos alguns dos mais importantes recursos da UML e o conhecimento técnico para a elaboração de diagramas, o que possibilita uma visão geral do processo de desenvolvimento e a percepção da importância da elaboração da análise de sistemas e da arquitetura do projeto.

Síntese

Chegamos ao final deste capítulo! Estudamos aqui como funciona a elicitação de requisitos, a modelagem de sistemas utilizando UML e o projeto de arquitetura. Também vimos alguns tipos de diagramas utilizados com mais frequência e aprendemos algumas técnicas para abstrair o máximo de informações, a fim de atingir os objetivos de um projeto. Com o conhecimento adquirido neste capítulo, é possível entender e participar de uma equipe de desenvolvimento de sistemas.

Neste capítulo, você teve a oportunidade de:

- aprender sobre as etapas e atividades para a elicitação de requisitos de sistemas;
- aprender a gerenciar e aplicar os conceitos de gerenciamento de requisitos;
- diferenciar e aplicar as metodologias de modelagem de sistemas;
- conhecer os diagramas da UML e suas aplicações;
- analisar os modelos de diagramas de contexto e interação;
- entender os modelos estruturais e comportamentais;
- entender e planejar a arquitetura necessária para o desenvolvimento e funcionamento de um sistema de informação.



Bibliografia

BOOCH, G. **UML**: guia do usuário. Grady Booch, James Rumbaugh, Ivan Jacobson. Rio de Janeiro: Elsevier, 2005.

ESTEVES, R. **O *Brainstorm* Eficaz, como gerar ideias com mais eficiência**. São Paulo: Dash, 2017.

FOWLER, M. **UML essencial**: um breve guia para a linguagem-padrão de modelagem de objetos. Tradução: João Tortello. 3. ed. Porto Alegre: Bookman, 2005.

FREIRE, Y. **TUCP-M – Pontos de Casos de Uso técnicos para manutenção de *software***. Dissertação de Mestrado. Fundação Edson Queiroz. Fortaleza: Universidade de Fortaleza (Unifor), 2008. Disponível em: <http://fattocs.com/files/pt/livro-apf/citacao/YaraMariaAlmeidaFreire-2008.pdf (http://fattocs.com/files/pt/livro-apf/citacao/YaraMariaAlmeidaFreire-2008.pdf)>. Acesso em: 15/08/2018.

GALLOTTI, G. M. A. **Qualidade de *Software***. São Paulo: Pearson. 2017.

LARMAN, C. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. Porto Alegre: Bookman, 2007.

MARTINS, J. C. C. **Gerenciamento de Projetos de Desenvolvimento de *Software* com PMI, RUP e UML**. 4. ed. Rio de Janeiro: Brasport, 2007.

MELO, A. C. **Desenvolvendo aplicações com UML 2.0: do conceitual à implementação**. 2. ed. Rio de Janeiro: Brasport, 2004.

MOORE, J. T. S. **Revolution OS**. Direção e Produção: J. T. S. Moore. Estados Unidos, 2001.

PFLEEGER, S. L. **Engenharia de *Software* - Teoria e Prática**. 2. ed. São Paulo: Pearson Addison Wesley, 2004.

SOMMERVILLE, I. **Engenharia de *Software***. 9. ed. São Paulo: Pearson Addison Wesley, 2011.