Capítulo

3

Requisitos

O levantamento e a análise de requisitos compõem uma parte significativa da fase de concepção dentro do UP. O analista pode e deve utilizar todas as informações disponíveis para identificar as fontes de requisitos (departamentos, pessoas, clientes, interfaces, sistemas etc.) e, para cada fonte, identificar as funções que o sistema deverá disponibilizar.

No caso da existência de diagramas de atividades ou de estados para entidades-chave do sistema, o levantamento de requisitos deve identificar quais as funções necessárias para realizar as atividades previstas ou as mudanças de estado.

A etapa de *levantamento de requisitos* corresponde a buscar todas as informações possíveis sobre as funções que o sistema deve executar e as restrições sobre as quais o sistema deve operar. O produto dessa etapa será o *documento de requisitos*, principal componente do anteprojeto de software.

A etapa de *análise de requisitos* serve para estruturar e detalhar os requisitos de forma que eles possam ser abordados na fase de elaboração para o desenvolvimento de outros elementos como casos de uso, classes e interfaces.

3.1. Levantamento de Requisitos

O levantamento de requisitos é o processo de descobrir quais são as *funções* que o sistema deve realizar e quais são as *restrições* que existem sobre essas funções. No caso do sistema Livir, por exemplo, o levantamento de requisitos vai permitir descobrir que o sistema deve controlar a compra e venda de livros, calcular automaticamente os pagamentos, permitir o registro de danos aos livros, gerar relatórios de vendas, verificar a disponibilidade de livros em estoque etc. Essas operações e muitas outras virão a constituir a *funcionalidade* do sistema, e por isso são chamadas também de *requisitos funcionais*.

As *restrições* sobre essas funções têm a ver com a questão: de que forma essas operações se realizam? Ou, ainda, quando, como, onde, para quem, por quem, por quanto tempo etc. essas operações se realizam? Essas restrições são também conhecidas como *requisitos não funcionais*.

3.1.1. Levantar Requisitos não é Projeto!

Um sistema a ser analisado é como uma floresta. Para explorar uma floresta desconhecida não é possível, em um primeiro momento, conhecer cada planta e inseto. Apenas no final da implantação do sistema é que a equipe poderá dizer que adquiriu conhecimento sobre cada uma das suas diminutas partes. Porém, no primeiro momento do processo de análise, não se pode entrar na floresta para estudar planta por planta porque assim não se irá adquirir uma visão do todo. Há um ditado que diz que *os detalhistas não conseguem enxergar a floresta devido ao excesso de árvores*.

Então, a fase de concepção deve fornecer a visão do todo, para que se possa ver o que é mais importante, e depois dividir o todo em partes nas quais os detalhes possam ser analisados e finalmente uma solução possa ser projetada. A organização de *ciclos iterativos* nas fases de elaboração e construção corresponde a subdividir a floresta em setores para olhar um setor de cada vez e, dessa forma, poder trabalhar com a complexidade inerente. Então, um dos objetivos finais da fase de concepção é justamente organizar a divisão do trabalho em casos de uso, que serão associados aos diferentes ciclos iterativos.

Na fase de concepção, o levantamento de requisitos é rápido e genérico. Ele é feito em extensão e não em profundidade. O analista deve entender a extensão do que o sistema deve fazer, mas sem detalhar *como* ele vai fazer. Somente na fase de elaboração, a análise dos requisitos será aprofundada.

A etapa de levantamento de requisitos deve ser de *descoberta* e não de invenção. Nela, a equipe de analistas, juntamente com o cliente e usuários, vai procurar listar o maior número possível de capacidades e restrições, mas sem se preocupar demasiadamente em ter uma lista completa por enquanto, o que normalmente é impossível. Os requisitos não descobertos nessa etapa deverão ser convenientemente acomodados ao longo do resto do processo de desenvolvimento.

Deve ficar claro para o analista que requisitos são coisas que o cliente ou usuário *solicita*, e não coisas que ele, como analista, *planeja*. Alguns analistas confundem o registro dos requisitos, que são a memória das solicitações do cliente, com o início do projeto do sistema. Um exemplo desse tipo de confusão consiste em colocar projetos de tabelas do banco de dados no documento de requisitos. Como justificar que um determinado projeto de tabelas relacionais seja uma *solicitação* do cliente? Isso eventualmente pode ser possível com clientes mais sofisticados ou que tenham sistemas legados, mas não é a regra geral. As tabelas de banco de dados são parte do *domínio da solução* (projeto), e não da análise. O analista deve buscar os requisitos que correspondem às informações que o cliente precisa que sejam gerenciadas. Depois, ele pode decidir se armazena essa informação em um banco de dados ou em alguma outra estrutura.

3.1.2. Desafios dos Requisitos

O documento ou diagrama de requisitos deve registrar as capacidades do sistema e as condições às quais ele deve se conformar. Os desafios ligados a essas informações são os seguintes:

a) como *descobrir* os requisitos;
b) como *comunicar* os requisitos para as outras fases ou equipes do projeto;
c) como *lembrar* dos requisitos durante o desenvolvimento e verificar se foram todos atendidos;
d) como *gerenciar* a mudança dos requisitos.

Não adiantaria escrever um belo documento de requisitos e depois não saber se os requisitos foram ou não atendidos pelo projeto. É importante que se tenham mecanismos para fazer sistematicamente essa verificação. Por isso, é importante manter relações de *rastreabilidade* entre os requisitos e outras partes do projeto do software.

Deve-se ter em mente também que os requisitos inevitavelmente mudam durante o desenvolvimento do projeto. Deve-se esperar, então, poder gerenciar a mudança dos requisitos nas demais fases de desenvolvimento.

Outras vezes, os requisitos mudam depois do desenvolvimento. Podem mudar as condições de contexto ou a forma de trabalho ou as políticas da empresa. Embora essas mudanças não possam ser previstas pelo analista, podem ser criados mecanismos que as acomodem ao sistema quando surgirem. Existem padrões de projeto específicos para tratar essas instabilidades do sistema (como, por exemplo, o padrão *Estratégia*).

Essas mudanças são totalmente imprevisíveis. Se o sistema não estiver estruturado para acomodar mudanças nos requisitos, haverá excesso de trabalho para implementá-las.

Esse tipo de situação faz com que os processos de análise e projeto dirigidos por requisitos (Alford, 1991) sejam inadequados para muitos sistemas. Fundamentar a arquitetura de um sistema em seus requisitos é como construir um prédio sobre areia movediça. Quando os requisitos mudam, a estrutura do sistema muda. O UP, porém, propõe que a arquitetura do sistema seja fundamentada em elementos muito mais estáveis: as *classes* (e *componentes*) que encapsulam informação e comportamento.

Essas classes, mais adiante, vão implementar as funcionalidades que, combinadas, permitem a realização dos requisitos. Mudando-se os requisitos, mudam-se as combinações, mas não a estrutura básica. Essa estrutura usualmente segue o *princípio aberto-fechado* (Meyer, 1988), no sentido de que está sempre pronta para funcionar (fechada), mas aberta para incorporar novas funcionalidades.

3.1.3. Requisitos Funcionais

Os requisitos funcionais devem conter basicamente os seguintes elementos:

a) a *descrição* de uma função a ser executada pelo sistema (usualmente entrada, saída ou transformação da informação);
b) a *origem* do requisito (quem solicitou) e/ou quem vai executar a função em questão (usuário);
c) quais as *informações* que são passadas do sistema para o usuário e vice-versa quando a função for executada;
d) quais *restrições lógicas* (regras de negócio) ou *tecnológicas* se aplicam à função.

Cada requisito funcional deve conter, portanto, uma *função*, que pode ser uma entrada (exemplo, "cadastrar comprador") ou saída (exemplo, "emitir relatório de vendas no mês").

É importante identificar a *origem* ou *usuário* do requisito, pois sempre é necessário validar os requisitos com essas fontes, verificando se estão bem escritos, completos e consistentes.

Algumas vezes também acontece de pessoas ou departamentos diferentes apresentarem o mesmo requisito de forma discrepante. Nesse caso, é necessário realizar um acordo ou identificar quem teria autoridade para determinar a forma aceitável do requisito.

As *informações de entrada e saída* são importantíssimas para que a análise de requisitos ocorra de forma sistemática. Sem essas informações, os requisitos ficam muito vagos e pouco é aprendido sobre o sistema nessa fase. Dizer simplesmente que "o sistema deve permitir o cadastro de compradores" é muito vago como requisito. O analista deve sempre procurar saber quais movimentos de informação essas funções envolvem. Por exemplo, o cadastro do comprador envolve apenas nome, endereço e telefone? No caso do sistema Livir, não. Deve incluir com certeza dados de um ou mais cartões de crédito, pois serão necessários para efetuar os pagamentos. Essa verificação de informações que entram e saem do sistema é que vai permitir ao analista descobrir a maioria dos conceitos e funções, realizando a pesquisa em extensão no espaço de requisitos para ter uma visão abrangente do todo. No exemplo visto, fica patente, após o detalhamento das informações que entram e saem, a necessidade de definir um requisito funcional para cadastrar cartões de crédito adicionais para o comprador.

3.1.4. Requisitos Não Funcionais

Os requisitos não funcionais aparecem sempre ligados a requisitos funcionais e podem ser basicamente de dois tipos: lógicos ou tecnológicos.

As *restrições lógicas* são as regras de negócio relacionadas à função em questão. Por exemplo, no registro de uma venda, uma série de restrições lógicas poderia ser considerada, como por exemplo: não efetuar a venda caso a operadora de cartão não autorize o pagamento ou não efetuar a venda caso a venda anterior tenha sido cancelada devido a um endereço inválido que ainda não foi corrigido.

As *restrições tecnológicas* dizem respeito à tecnologia para a realização da função, como, por exemplo, a interface (Web, por exemplo), o tipo de protocolo de comunicação, restrições de segurança ou tolerância a falhas etc.

Por exemplo, registrar a venda de um conjunto de livros é um requisito funcional. Estabelecer que o tipo de interface para efetuar uma venda deve seguir um padrão de interface de fluxo sequencial de telas é uma restrição tecnológica (de interface) sobre a forma como essa função é realizada.

Outro exemplo de requisito não funcional seria "a autorização de débito no cartão de crédito não deve levar mais do que 5 segundos". Isso seria uma restrição tecnológica de desempenho e afetaria a forma como o projetista iria pensar no mecanismo de acesso ao sistema da operadora de cartões. Nesse caso, o projeto do sistema teria de considerar seriamente o uso de conexões fixas com as operadoras de cartão em vez de linhas discadas.

3.1.5. Requisitos Suplementares

Os *requisitos suplementares* são todo tipo de restrição tecnológica ou lógica que se aplica ao sistema como um todo e não apenas a funções individuais. Por exemplo, um requisito suplementar pode estabelecer que o sistema deva ser compatível com um banco de dados legado ou ser implementado em uma determinada linguagem de programação, ou ainda, seguir um determinado padrão de interface ou *look and feel.*

Deve-se tomar certo cuidado no enunciado dos requisitos suplementares. Um requisito como "o sistema deve ser fácil de usar" é muito pouco esclarecedor para que possa ser útil. Melhor seria dizer: "o sistema terá ajuda *on-line* em todas as telas e para todas as funções". Isso dá uma ideia mais precisa sobre o que realmente deve ser realizado pelo sistema.

3.1.6. Documento de Requisitos

O *documento de requisitos* registra todos os tópicos relativos ao que o sistema deve fazer e sob quais condições. Esse documento ainda não precisa ser totalmente estruturado, e assume-se que não será completo em profundidade, embora se possa esperar que esteja razoavelmente completo em extensão. Eventuais lacunas desse documento serão preenchidas durante a fase de elaboração.

O documento de requisitos pode ser organizado de forma textual ou na forma de um diagrama (o diagrama de requisitos existente em algumas ferramentas *CASE*, porém, não pertence à especificação da UML).

Os requisitos funcionais podem ainda ser subdivididos em subsistemas, especialmente se a quantidade de funções for muito grande. Essa é uma primeira forma de organização do sistema.

Sugere-se que o documento de requisitos tenha um índice consistindo no nome da função ou requisito suplementar e que o corpo do documento contenha os detalhes mencionados na Seção 3.1.3.

A Figura 3.1 apresenta um exemplo de documento de requisitos para o sistema Livir. Se houvesse subsistemas, eles seriam o primeiro nível de divisão dentro de "Requisitos funcionais", contendo cada divisão um conjunto de requisitos numerados.

Sistema Livir – Documento de Requisitos

Requisitos funcionais

1. Registrar novos títulos a partir do catálogo das editoras.
2. Registrar vendas de livros.
3. Realizar encomendas de livros.
4. Registrar e autorizar pagamentos com cartão de crédito.
5. Registrar e aplicar promoções.
6. Calcular custos de entrega.
7. Emitir relatório de livros mais vendidos.
8. Emitir relatório de compradores mais assíduos.
9. Emitir sugestões de compra para compradores baseadas em compras anteriores.
10. ...

Requisitos suplementares

1. O sistema deve operar via interface Web.
2. Todos os controles de interface devem ter um campo de ajuda associado.
3. ...

Figura 3.1: O índice de um documento de requisitos para o sistema Livir.

O detalhamento de cada requisito (especialmente os funcionais) deve aparecer no corpo do documento. Esse detalhamento deve conter as partes mencionadas na Seção 3.1.3. A Figura 3.2 apresenta um exemplo.

1. Registrar novos títulos a partir do catálogo das editoras

Descrição:

O gerente seleciona as editoras para as quais pretende fazer a atualização. O processo é automático. O sistema consulta os ISBN disponibilizados e os compara com os existentes na base. Havendo novos ISBN, o sistema atualiza a base com as novas informações.

Fontes:

Sr. Fulano de Tal (gerente) e manual técnico da interface de catálogo das editoras.

Usuário:

O próprio gerente.

Informações de entrada:

O gerente informa quais são as editoras para as quais pretende fazer a atualização a partir de uma lista fornecida pelo sistema.

Informações de saída:

- A lista de editoras (nome).
- O relatório de atualizações efetuadas (uma lista contendo: nome da editora, ISBN, título e preço de compra).

Restrições lógicas:

Não há.

Restrições tecnológicas:

1. Deve ser usado o sistema de interface com as editoras, de acordo com o manual XYZ.1234.
2. A seleção feita pelo gerente se dá através de uma lista de seleção múltipla, a qual deve ser ordenada de forma alfabética.
3. ...

Figura 3.2: Detalhamento de um requisito.

Observa-se que o detalhamento das informações que entram e saem do sistema é fundamental para a compreensão mais detalhada dessa função. Através desse detalhamento é que a verdadeira dimensão do sistema vai sendo descoberta.

Sem esse detalhamento, o sistema pode parecer mais simples do que realmente é, o que explica por que, em muitos casos, os analistas esperam desenvolver um sistema em determinado tempo mas levam muito mais tempo, estourando prazos e orçamentos.

3.2. Análise de Requisitos

Na análise de requisitos, o analista vai procurar caracterizar certas propriedades dos requisitos já levantados, conforme as subseções seguintes.

3.2.1. Permanência e Transitoriedade

Uma primeira caracterização considerada importante na análise de requisitos é decidir se determinado requisito não funcional ou suplementar é *permanente* ou *transitório*.

Requisitos não funcionais podem ser considerados permanentes (não vão mudar) ou transitórios (podem mudar) de acordo com uma decisão tomada pelo analista em conjunto com o cliente. O requisito não tem a propriedade de permanência ou transitoriedade por si: ela é decidida de acordo com a conveniência. Um mesmo requisito pode ser considerado permanente ou transitório dependendo do que se queira em relação ao tempo de desenvolvimento ou custo da manutenção do software.

Por exemplo, um requisito suplementar poderia estabelecer que o sistema Livir trabalhe com uma única moeda: o real. Se esse requisito suplementar for considerado *permanente*, todo o sistema será construído de forma que o real seja a única moeda. Se o requisito for considerado *transitório*, o sistema deverá ser construído de forma a poder acomodar futuramente outras moedas ou, ainda, mais de uma moeda de cada vez.

As consequências de decidir que um requisito é permanente são as seguintes:

a) fica mais barato e rápido desenvolver o sistema em si;
b) fica mais caro e demorado mudar o sistema caso o requisito, por algum motivo, mude no futuro (com a implantação de uma nova moeda, por exemplo).

Por outro lado, decidir que um requisito é transitório tem como consequências:

a) fica mais caro e complexo desenvolver o sistema (ele deverá acomodar funcionalidades para a mudança da moeda);

b) fica mais barato e rápido fazer a manutenção no sistema (caso a moeda mude, o sistema já está preparado para acomodar esse fato com uma simples reconfiguração).

Então, a natureza dos requisitos não funcionais não vai decidir se eles são permanentes ou transitórios. O analista é que tem de tomar essa decisão (com o aval do cliente). O ideal seria elencar aqueles requisitos de maior importância (que se espera que possam *mesmo* mudar num futuro próximo e cuja mudança tenha maior *impacto* no sistema) e considerá-los transitórios, deixando os demais como permanentes.

3.2.2. Requisitos Evidentes e Ocultos

Os requisitos funcionais podem ser opcionalmente classificados em evidentes ou ocultos:

a) os *requisitos funcionais evidentes* são funções efetuadas com conhecimento do usuário. Esses requisitos usualmente correspondem a trocas de informação, como consultas e entrada de dados, que ocorrem com o meio exterior através da interface do sistema;

b) os *requisitos funcionais ocultos* são funções efetuadas pelo sistema sem o conhecimento explícito do usuário. Usualmente, são cálculos ou atualizações feitas pelo sistema sem a solicitação explícita do usuário, mas como consequência de outras funções solicitadas por ele.

É importante classificar os requisitos dessa forma porque, posteriormente, eles serão associados aos casos de uso através de *relações de rastreabilidade*. Apenas os requisitos evidentes corresponderão aos passos do caso de uso expandido porque são executados com o conhecimento explícito do usuário. Os requisitos ocultos são executados internamente pelo sistema. Então, embora não apareçam explicitamente nos passos de um caso de uso expandido, esses precisam ser adequadamente associados a aqueles para ser lembrados no momento de projetar e implementar as operações do caso de uso.

Um exemplo de requisito evidente é emitir um relatório de livros mais vendidos por requisição do gerente. Um exemplo de requisito oculto seria aplicar uma política de desconto, se ela existir. Nesse caso, nenhum usuário solicita explicitamente ao sistema para fazer essa aplicação. É uma atividade que o sistema executa independentemente dos usuários e, portanto, de forma oculta.

3.2.3. Requisitos Obrigatórios e Desejados

Os requisitos ainda podem ser classificados em *obrigatórios* e *desejados*, ou seja, aqueles que devem ser obtidos de qualquer maneira e aqueles que podem ser obtidos caso isso não cause maiores transtornos no processo de desenvolvimento.

No caso dos requisitos funcionais, essa classificação indica uma priorização de desenvolvimento. Um bom analista tomaria as funções como base para calcular o tempo de desenvolvimento. Assim, não faria muito sentido falar em funções obrigatórias e desejáveis, mas sim em quanto tempo levaria para desenvolver esse ou aquele conjunto de funções.

Entretanto, nem sempre a coisa funciona assim. No caso de alguns sistemas pode-se querer trabalhar com prioridades, desenvolvendo inicialmente determinadas funções consideradas obrigatórias e, posteriormente (se sobrar tempo), outras funções consideradas desejadas.

Mas os requisitos não funcionais e suplementares são bem mais imprevisíveis do que os funcionais para efeito de estimativa de esforço. Assim, em alguns casos, pode ser necessário efetivamente classificar esses requisitos por graus de prioridade.

Definem-se algumas restrições que devem ser obtidas a qualquer custo e outras que seria desejável obter, desde que isso não extrapole o tempo ou recursos disponibilizados para o projeto.

Por exemplo, no caso do sistema Livir, o requisito de que a interface seja Web poderia ser considerado obrigatório. Nesse caso, não se aceita outra coisa. Porém, o acesso através de telefone celular poderia ser um requisito desejável, já que não é absolutamente necessário para o efetivo funcionamento do sistema.

Com a formalização dos contratos de desenvolvimento de software, porém, cada vez menos flexibilidade se tem em relação a requisitos desejáveis. O desenvolvedor deve dizer claramente quais requisitos vai implementar, quanto tempo vai levar e quanto vai custar. Qualquer desvio dessas previsões pode implicar multas ou cancelamento de contratos.

3.2.4. Classificação de Requisitos Não Funcionais e Suplementares

Os requisitos não funcionais e suplementares podem ser classificados por atributo, ou seja, se são requisitos de interface, de implementação, de efi-

ciência, de tolerância a falhas etc. A finalidade principal das classificações de requisitos em categorias é a organização. Algumas sugestões de possíveis categorias são:

a) *usabilidade*: quais fatores humanos estão envolvidos no sistema? Que tipo de ajuda o sistema vai prover? Quais as formas de documentação ou manuais estarão disponíveis? Como esses manuais vão ser produzidos? Que tipo de informação eles vão conter? Seria interessante definir esses tópicos na fase de concepção, visto que o contrato com o cliente deve especificar muitas dessas questões;

b) *confiabilidade*: que tipo de tratamento de falhas o sistema vai ter? O analista não é obrigado a produzir um sistema totalmente tolerante a falhas, mas deve estabelecer que tipo de falhas o sistema será capaz de gerenciar: falta de energia, falha de comunicação, falha na mídia de gravação etc. Não se deve confundir *falha* com erro de programação, visto que erros de programação são elementos que nenhum software deveria conter. As falhas são situações anormais que podem ocorrer mesmo para um software implementado sem nenhum erro de programação;

c) *desempenho*: que tipo de eficiência e precisão o sistema será capaz de apresentar? Pode-se estabelecer, por exemplo, como requisito de eficiência, que nenhuma consulta à base de dados de compradores vai demorar mais de cinco segundos. Na fase de concepção não se define *como* o sistema fará para cumprir o requisito, apenas se diz que de alguma forma ele terá de ser cumprido no projeto. Cabe ao projetista e programador garantir que o requisito seja satisfeito. Se o analista por algum motivo conclui que o requisito dificilmente poderá ser implementado, o requisito passa a ser um *risco* do sistema e eventualmente necessitará de um estudo mais aprofundado ainda na fase de concepção, para verificar a possibilidade de sua realização;

d) *configurabilidade*: o que pode ser configurado no sistema? Deve-se definir os elementos que poderão ser configurados pelo usuário sem que seja necessário recompilar o sistema. Exemplos de itens configuráveis são: o tipo de impressoras, a moeda do país, políticas da empresa, fontes e cores da interface, idioma etc.;

e) *segurança*: quais são os tipos de usuários e que funções cada um pode executar? Sugere-se a implantação de um sistema de permissões, de forma que possam ser criados dinamicamente perfis de usuários com diferentes conjuntos de permissões;

f) *implementação*: qual linguagem deve ser usada? Por que motivo? Que bibliotecas estarão disponíveis? Quais bancos de dados serão acessíveis? Há necessidade de comunicação com sistemas legados?;

g) *interface*: como deve ser a interface? Vai ser seguida alguma norma ergonômica?;

h) *empacotamento*: de que forma o software deve ser entregue ao usuário final?;

i) *legais*: muitas vezes, uma equipe de desenvolvimento deve contar com uma assessoria jurídica para saber se está infringindo direitos autorais ou normas específicas da área para a qual o software está sendo desenvolvido.

Embora essa lista seja extensa, o analista deve ter em mente que se trata apenas de uma forma de classificação para que ele possa mais facilmente avaliar quais requisitos são relevantes para cada um dos tipos listados. Não há necessidade de procurar requisitos que não existem, por exemplo, estabelecendo complicados requisitos de empacotamento para um cliente para o qual não faz a menor diferença a forma como o software será entregue.

Também não se deve perder tempo discutindo se um requisito é desse ou daquele tipo. Mais importante do que classificar é reconhecer que o requisito existe. Esse tipo de discussão, que não acrescenta conhecimento ao estudo do problema, deixa a análise travada, ou seja, não se anda mais para frente.