Programador Web

Análise de requisitos

Conceitos

Via de regra, requisito é uma condição necessária para satisfazer um objetivo. Antigamente, no contexto de desenvolvimento, dizia-se que requisitos eram sinônimos de funções, ou seja, de tudo que o *software* deveria fazer funcionalmente. Todavia, percebeu-se que, além de funções, requisitos são objetivos, propriedades, restrições que o sistema deve ter para satisfazer contratos, padrões ou especificações de acordo com o(s) usuário(s).

Portanto, um requisito pode ser um aspecto que o sistema proposto deve fazer ou uma restrição no desenvolvimento do sistema. Em ambos os casos, o foco deve ser contribuir para resolver os problemas do cliente, e não atender aos desejos do programador ou do arquiteto.

O conjunto dos requisitos como um todo representa um acordo negociado entre todas as partes interessadas no sistema. Sendo assim, existem dois tipos de requisitos:

Requisitos funcionais

Os requisitos funcionais referem-se aos aspectos de funcionamento do sistema (o que ele deve fazer, as suas funções e a informação) de modo funcional, ou seja, quais são as funcionalidades e os serviços.

Exemplos:

- [RF001] O Sistema deve cadastrar médicos profissionais (entrada);
- [RF002] O Sistema deve emitir um relatório de clientes (saída);
- [RF003] O Sistema deve passar um cliente da situação "em consulta" para "consultado" quando o cliente terminar de ser atendido (mudança de estado);
- [RF004] O cliente pode consultar seus dados no sistema.

Requisitos não funcionais

Os requisitos não funcionais referem-se aos critérios que qualificam os requisitos funcionais. Logo, aqui são encontradas funcionalidades que darão sustentação para o sistema e para os requisitos funcionais.

Tais critérios podem ser de qualidade para o *software* (requisitos de *performance*, usabilidade, confiabilidade, robustez etc.) ou, ainda, de qualidade para o processo de *software* (requisitos de entrega, implementação etc.).

Os requisitos não funcionais definem propriedades e restrições do sistema, como tempo, espaço, linguagens de programação, versões do compilador, sistemas de gestão de base de dados (SGBD), sistema operacional, método de desenvolvimento etc.

Geralmente, os requisitos não funcionais são mensuráveis. Por isso, preferencialmente uma medida ou uma referência para cada requisito não funcional deve ser associada.

Exemplos:

- [RNF001] O sistema deve imprimir o relatório em até 5 segundos;
  - [RNF002] Todos os relatórios devem seguir o padrão de relatórios especificado pelo setor XYZ;
- [RNF003] O sistema deve ser implementado em Java.

Tipos de requisitos não funcionais

Os requisitos não funcionais podem ser divididos nestes três tipos:

- Requisitos do produto final
- Requisitos organizacionais
- Requisitos externos

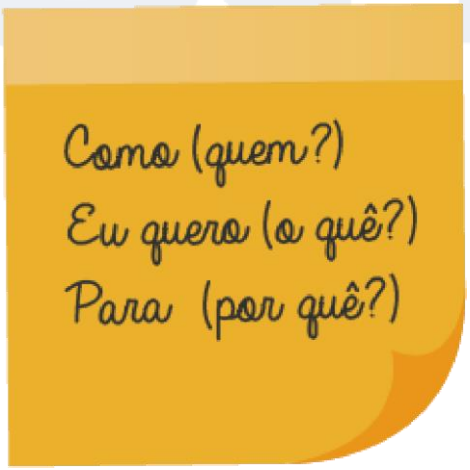
Na figura a seguir, é demonstrada a divisão dos requisitos não funcionais, facilitando a identificação deles. Assim, o analista pode, ao olhar para o gráfico (figura 1), visualizar funcionalidades necessárias para corresponder aos tipos de requisitos não funcionais.

Figura 1 – Fluxograma de requisitos não funcionais

Os requisitos são fundamentais para o desenvolvimento de projeto. Logo, um levantamento de requisitos bem elaborado ajuda na construção sólida de um sistema. Ao contrário, um levantamento malfeito gera retrabalho, gastos, atrasos de cronograma e insatisfação do usuário, o qual percebe que a sua solicitação não foi compreendida pelo time.

Os requisitos de sistema estão presentes em todas as fases do ciclo de desenvolvimento, bem como servem de apoio ao desenvolvimento e ajudam nos testes, na avaliação do *software* e na apresentação do sistema. Porém, tudo isso inicia com o levantamento de requisitos, que pode ser realizado de diferentes formas.

Histórias de usuário (user stories)



As *user stories* são utilizadas para auxiliar na identificação de requisitos do sistema a partir de uma linguagem mais voltada para o negócio e não tão técnica. Quanto mais “quebrada” a história, melhor especificado será o sistema, isto é, a captura de “quem”, “o que” e “por que”.

Portanto, uma *user story* deve relatar o modo como algo (o que) deve funcionar para o usuário do sistema (quem) e o motivo de o funcionamento ser aquele (por que), tudo voltado para a parte de negócio, e não para a forma técnica. A parte técnica será definida pelos desenvolvedores, que terão acesso às histórias e saberão como o sistema deve se comportar.

Observe o exemplo:

Quem? O atendente de caixa do supermercado.

O quê? Quer lançar os produtos de compra do cliente.

Por quê? Para que o sistema crie a lista de compras com a soma dos valores.

O exemplo citado é a forma mais simples de montar uma *user story* para iniciantes. Assim, ao colocar “quem”, “o que” e “por que”, fica fácil listar as respostas. Analistas experientes tendem a montar tudo de uma vez sem as perguntas, mas isso depende do entendimento de cada um, sendo as duas formas válidas. Apenas devem estar claros quem é o usuário da ação, o que ele quer fazer e qual é o motivo, para que todos consigam entender o processo e levantar os requisitos necessários.

Mais alguns exemplos podem ser vistos a seguir:

- **Eu, como sistema**, quero **segurança no acesso** ao sistema para que **nenhuma pessoa não autorizada o acesse**.
- **Eu, como médico da clínica**, desejo **ter os meus pacientes cadastrados** para que **possa atendê-los** no sistema.
- **Eu, como médico e administrador**, quero **ter os profissionais da clínica cadastrados no sistema** para que **possa controlar atendimentos**.

Após elaborar as histórias de usuário, o analista deve listar as regras do sistema para que fiquem claros os limites deste, facilitando a validação com o usuário e fazendo com que o time de desenvolvimento entenda como o sistema deve ser montado, pois apenas saber a história não ajuda na construção. É importante estar clara a forma como o sistema responderá à funcionalidade, o que também facilita a elaboração da lista de requisitos funcionais e não funcionais. Exemplo 1

Regras:

1. O cadastro de paciente deve ser feito por código, incluindo nome, nome do responsável (se houver), documento do paciente ou do responsável (se houver), data de nascimento, campo para observações e *check* indicando se o paciente está ativo.

2. No cadastro, deve ser guardado o histórico de atendimentos, permitindo que o médico veja as informações de cada atendimento.

3. Um paciente não pode ser excluído, apenas desativado.

4. A inclusão de foto do paciente é permitida, mas não obrigatória.

5. A busca de paciente por nome, código ou pacientes ativos é permitida.

6. A lista inicial deve exibir pacientes com mais atendimentos feitos nos últimos 60 dias.

Exemplo 2 **Regras:**

1. O sistema deve solicitar um usuário ao ser acessado, com caracteres normais, minúsculos ou maiúsculos, ou, ainda, caracteres especiais e números (no máximo dez caracteres).
2. O sistema deve solicitar uma senha, de quatro a 12 caracteres (a senha deve ser criptografada no banco de dados).
3. Caso o usuário erre o *login* ou a senha, deve ser exibida esta mensagem: "*Login* ou senha incorreto".
4. Na tela, deve aparecer a opção de *login* para alterar a senha do usuário. Caso essa opção seja acessada, o sistema deve abrir uma tela solicitando *login*, senha atual, senha nova e confirmação da senha nova, validando as informações antigas para liberar a alteração. Se alguma informação estiver inválida, o sistema deve exibir a mesma mensagem da regra 3.

Após elaborar as regras, inicia-se a parte de montar os diagramas, os quais terão as regras em sua base. Por isso, é muito importante saber elaborar correta e claramente as regras, para facilitar a diagramação.

Entrevista com cliente e questionário para levantamento de requisitos

A **entrevista** normalmente é a primeira técnica utilizada, sendo um método tradicional mais simples e que produz bons resultados na fase inicial de obtenção

de dados. Nela, analistas entrevistam clientes para definir os objetivos gerais e as restrições que o *software* deverá ter.

A entrevista deve ser feita de forma objetiva, visando a obter o máximo de informações do cliente. Diversas seções de entrevistas podem ser marcadas.

As dicas seguintes podem auxiliar na direção de entrevistas bem sucedidas com o usuário:

- Desenvolver um plano geral de entrevistas
- Certificar-se da autorização para falar com os usuários
- Planejar a entrevista, usando, assim, o tempo eficientemente

A diferença entre o plano geral e o planejamento da entrevista está relacionado ao fato de que, no plano geral, o analista montará um questionário juntamente com respostas esperadas e um fluxo da entrevista. Já no planejamento da entrevista, com base no plano geral, o analista destacará importâncias e dividirá o tempo das perguntas, para que o tempo correto, de acordo com a complexidade e a importância, seja aplicado a cada uma delas.

- Utilizar ferramentas automatizadas que sejam adequadas e um estilo apropriado ao entrevistar

Quando se fala em ferramentas automatizadas, podem-se utilizar ferramentas como o questionário do Google, que serve para elaborar questionários.

Escopo

É importante determinar um escopo relativamente limitado, focando em uma pequena parte do sistema, para que a reunião não se estenda por mais de uma hora. O usuário tem dificuldade de concentração em reuniões muito longas, por isso é importante focar, durante a reunião, no escopo definido.

Planejamento

Para planejar a entrevista, é necessário que antes dela sejam coletados e estudados todos os dados pertinentes à discussão, como formulários, relatórios, documentos e outros. Dessa forma, o analista estará bem contextualizado e terá mais produtividade nos assuntos a serem discutidos na entrevista.

Um plano de entrevista é necessário para que não haja dispersão do assunto principal e para que a entrevista não fique longa, deixando o entrevistado cansado e não produzindo bons resultados.

Perguntas

Para elaborar perguntas detalhadas, é necessário solicitar algumas ações ao usuário. São elas:

- Explicar o relacionamento entre o que está em discussão e as demais partes do sistema
- Descrever o ponto de vista de outros usuários com relação ao item que esteja sendo discutido
- Descrever informalmente a narrativa do item em que o analista deseja obter informações
- Indicar se o item em discussão depende, para a própria existência, de alguma outra coisa, para assim poder juntar os requisitos comuns do sistema, formando um escopo conciso

Condução

A atitude do analista durante a entrevista é determinante para o fracasso ou o sucesso dele. Deve-se evitar conduzir a entrevista como uma tentativa de persuadir o entrevistado. O uso excessivo de termos técnicos também não é adequado. O entrevistador deve cuidar o tom de voz e a postura ao mencionar informações de diferentes entrevistados.

O analista deve ter em mente que o entrevistado é o perito no assunto em questão e é quem fornecerá as informações necessárias para o desenvolvimento do sistema. Portanto, é importante não questionar a credibilidade do entrevistado.

Ao conduzir a entrevista, convém que o entrevistador dê margem ao entrevistado para que este possa expor as suas ideias. Para tanto, pode-se utilizar a confirmação, momento em que o entrevistador reformula, com as próprias palavras, o que o entrevistado disse, verificando se este está compreendendo o que está sendo dito.

Depois da entrevista, é preciso validar se o que foi documentado pelo analista está de acordo com a necessidade do usuário, se o usuário não mudou de opinião e se o usuário entende a notação ou a representação gráfica das informações fornecidas.

Questionário

O uso de **questionário** é indicado, por exemplo, quando há diversos grupos de usuários que podem estar em locais diferentes. Nesse caso, elaboram-se pesquisas específicas de acompanhamento com usuários selecionados, os quais têm um potencial de contribuição aparentemente mais importante.

Existem vários tipos de questionários que podem ser utilizados. Entre eles, podem-se mencionar a **múltipla escolha**, a **lista de verificação** e as **questões com espaços em branco** (caixa de texto livre para digitação do entrevistado). O questionário deve ser desenvolvido para minimizar o tempo gasto em respostas.

Na fase de preparação do questionário, deve ser indicado o tipo de informação a ser obtida. Assim que os requisitos forem definidos, o analista deve elaborar o questionário com questões simples, claras e concisas, deixar espaço suficiente para as respostas que forem descritivas e agrupar as questões de tópicos específicos em um conjunto com um título especial.

O questionário deve vir acompanhado de uma carta explicativa, redigida por um alto executivo, para que a resposta do usuário tenha o endosso de seu superior, pois, às vezes, há apenas a figura de um usuário representando o desejo de uma instituição, e para que ao final do projeto não seja desfeito algo porque a visão da corporação está diferente da visão do usuário entrevistado. O ideal é que o

alto escalão esteja ao menos ciente das respostas, para enfatizar a importância da pesquisa para a organização.

Ademais, deve ser desenvolvido um controle que identifique todas as pessoas que receberão o questionário. A distribuição deve ocorrer junto com instruções detalhadas sobre como preenchê-lo, e o prazo para devolução do questionário deve ser indicado claramente.

Ao analisar as respostas dos participantes, é feita uma consolidação das informações fornecidas no questionário, documentando as principais descobertas e enviando uma cópia com tais informações para o participante, em respeito ao tempo dedicado à pesquisa.

Mencionaram-se aqui apenas as duas formas mais conhecidas de levantamento de requisitos, mas muitas outras podem ser utilizadas. O *brainstorming*, por exemplo, é uma técnica em que todos os envolvidos lançam ideias e o moderador vai registrando para posterior discussão e refinamento das ideias pelo grupo, nascendo, assim, os requisitos do sistema.

Outra forma de levantar requisitos é a prototipagem, que consiste em o analista criar um protótipo do sistema, apresentar para o usuário e iniciar um debate em que sugestões são lançadas. Vários outros métodos podem ser utilizados, tais como *joint application development* (JAD), *workshops*, etnografia e levantamento orientado a pontos de vista.

Para saber mais detalhes sobre outras formas de levantar requisitos, pesquise sites e fóruns, como DevMedia, iMasters etc., que compartilham o trabalho de desenvolvedores e analistas. Além disso, verifique as referências deste material, que indicam livros utilizados como base para entendimento.

Sistema de classificação de requisitos

FURPS+

FURPS+ é o acrônimo para representar um modelo para classificação de atributos de qualidade de *software* (requisitos funcionais e não funcionais). Sendo assim, fica mais fácil para o analista identificar se a análise tem toda a cobertura possível.

Funcionalidade (*functionality*): é todo o aspecto funcional do sistema que está sendo desenvolvido.

Usabilidade (*usability*): indica o tempo de treinamento necessário para um usuário se tornar produtivo, o tempo de duração desejado para determinada operação no sistema e na ajuda *on-line*, a documentação do usuário e o material de treinamento.

Confiabilidade (*reliability*): refere-se à disponibilidade, ao tempo de correção, ao tempo permitido para indisponibilidade quando ocorre uma falha, à precisão, ao número máximo de defeitos (*bugs*/*kilo lines of code* – KLOC), às categorias de bugs (categorizados por nível de impacto).

Desempenho (*performance*): indica o tempo de resposta para uma transação, a *throughput* (transações por segundos), a capacidade (transações concorrentes), a operação parcial (situação do sistema aceitável quando ele estiver prejudicado de alguma forma) e o uso de recursos (memória, espaço em disco, comunicação etc.).

Suportabilidade (*supportability*): indica o padrão de codificação, a convenção de nomenclatura, as bibliotecas de classes e os utilitários de manutenção. O **"+"(plus)** indica outros requisitos, como *design*, implementação, interface, aparelhos eletrônicos que devem estar conectados para o bom funcionamento do sistema etc.

Usabilidade (*usability*): indica o tempo de treinamento necessário para um usuário se tornar produtivo, o tempo de duração desejado para determinada operação no sistema e na ajuda *on-line*, a documentação do usuário e o material de treinamento.

Confiabilidade (*reliability*): refere-se à disponibilidade, ao tempo de correção, ao tempo permitido para indisponibilidade quando ocorre uma falha, à precisão, ao número máximo de defeitos (*bugs*/*kilo lines of code* – KLOC), às categorias de bugs (categorizados por nível de impacto).

Desempenho (*performance*): indica o tempo de resposta para uma transação, a *throughput* (transações por segundos), a capacidade (transações concorrentes), a operação parcial (situação do sistema aceitável quando ele estiver prejudicado de alguma forma) e o uso de recursos (memória, espaço em disco, comunicação etc.).

Suportabilidade (*supportability*): indica o padrão de codificação, a convenção de nomenclatura, as bibliotecas de classes e os utilitários de manutenção. O **"+"(plus)** indica outros requisitos, como *design*, implementação, interface, aparelhos eletrônicos que devem estar conectados para o bom funcionamento do sistema etc.

Diagramas e suas aplicações
Diagrama de atividade

Um diagrama de atividade **é essencialmente um gráfico de fluxo**, envolvendo a modelagem das etapas sequenciais em um processo computacional. Uma atividade é uma execução não atômica em andamento em uma máquina de estados e que acaba resultando em alguma ação. Logo, a atividade representa uma ação/execução que poderá sofrer alterações e gerar novas atividades.

O **diagrama de atividade** é um diagrama definido pela linguagem de modelagem unificada (*unified modeling language* – UML).

Os diagramas de atividade não são importantes somente para a modelagem de aspectos dinâmicos de um sistema ou um fluxograma, mas também para a construção de sistemas executáveis por meio de engenharia de produção reversa.
Os diagramas de atividade não são importantes somente para a modelagem de aspectos dinâmicos de um sistema ou um fluxograma, mas também para a construção de sistemas executáveis por meio de engenharia de produção reversa.

Alguns conceitos importantes estão dentro do diagrama de atividade. São eles:

- Atividades
- Subatividade
- Transição
- Ação
- Decisão
- Raia
- Bifurcação (*fork*)
- Sincronização (*join*)
- Objeto
- Envio de sinal
- Recepção de sinal
- Região
- Exceção

A **atividade** nada mais é do que o comportamento a ser realizado, utilizado quando se cita uma atividade no diagrama. Já a **subatividade** é a execução de uma sequência não atômica de atividades.

A **transição** é o fluxo de uma atividade para outra, o que acaba gerando uma **ação**. Essa ação é uma transformação dentro do sistema. Dentro da transição, pode-se colocar o conceito de **decisão**, que consiste, na dependência de uma condição, em mostrar as diferentes transições. A transição representa uma decisão que pode desviar o fluxo ilustrado no diagrama.

Semelhante a esse conceito, há a *fork* e a *join*. A *fork* (bifurcação) separa a transição em duas ou mais transições que devem ser executadas ao mesmo tempo, servindo para ilustrar quando o sistema deve executar mais atividades em paralelo. Já a *join* (sincronização) faz o oposto: une duas ou mais transições em uma – no caso, quando o sistema executa duas atividades em paralelo e o seu resultado parte para uma atividade ou transição.

A **raia** ilustra fronteiras entre módulos, funcionalidades, sistemas ou subsistemas, conforme o nível de detalhe e o foco do diagrama.

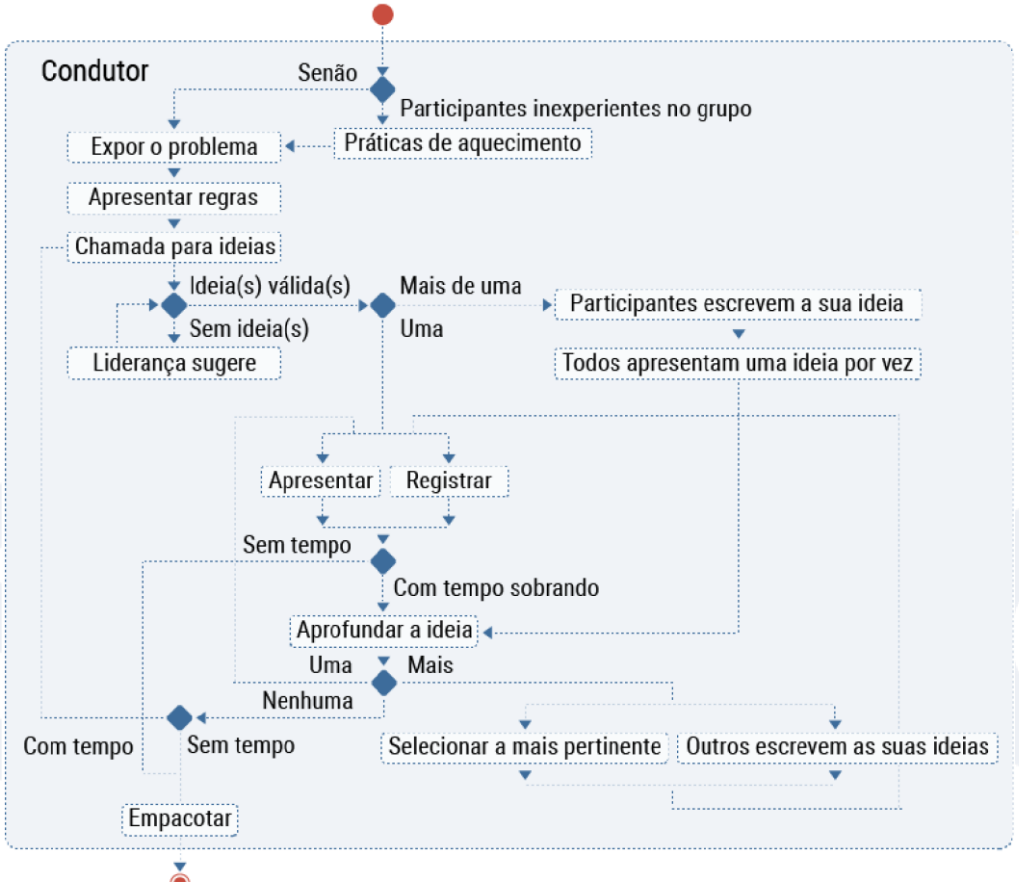


Figura 2 – Diagrama de atividade para elaboração de ideia

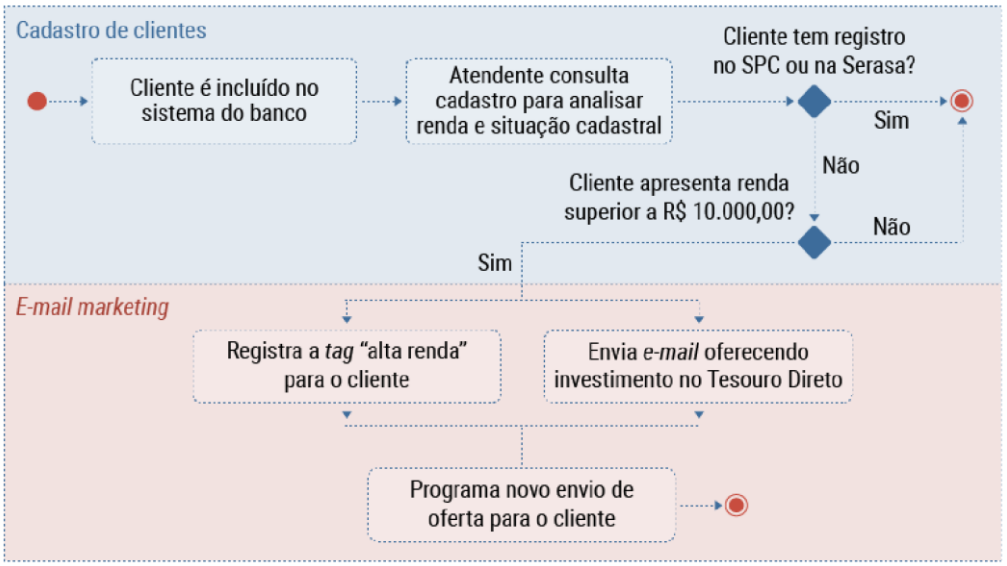


Figura 3 – Diagrama de atividade para relacionamento com o cliente Fonte: <https://www.ateomomento.com.br/uml-diagrama-deatividades/>.

O propósito desse tipo de diagrama é documentar o aspecto funcional (não estrutural) do *software*; mostrar os **aspectos específicos** de alguma rotina do negócio que será automatizada pelo *software*; demonstrar como as

funcionalidades realizarão requisitos funcionais (funções executadas pelas funcionalidades); e documentar de forma **macro** como o sistema funcionará, mas orientado ao *software*, e não ao processo de negócio. É recomendável não utilizar o diagrama de atividade quando da elicitação de requisitos, pois o foco do diagrama de atividades é o **comportamento**, **o aspecto "funcional"** do *software*.

Quanto à modelagem de processos de negócio, **modelar processos não é modelar *software*.**

Como visto na figura 3, além de uma atividade depender da outra, ainda existem decisões que impactam diretamente o fluxo, como a renda do cliente. No diagrama, caso a renda seja superior a R$ 10.000,00, o fluxo segue para o envio de *e-mail marketing*. Contudo, caso a renda não seja superior a R$ 10.000,00, o fluxo segue para o encerramento do diagrama.

Sendo assim, quando o desenvolvedor for criar o diagrama, ele deve saber exatamente como o *software* se comporta nesse tipo de situação.

Diagrama de caso de uso

Os casos de uso, individualmente, **descrevem requisitos funcionais**. A especificação dos requisitos visa a descrever o que o sistema deve fazer para satisfazer as metas dos usuários (requisitos funcionais) e quais são as outras propriedades desejáveis que o sistema contém (requisitos não funcionais). O diagrama de caso de uso, portanto, facilita a visualização dos requisitos.

Observe alguns conceitos em diagramas de caso de uso:

- Casos de uso
- Atores
- Relacionamento de dependência
- Relacionamento de generalização
- Relacionamento de associação

Figura 4 – Caso de uso de agendamento e consulta de atendimentos

Um caso de uso, de fato, é uma especificação de um conjunto de ações executadas por um sistema e contém um resultado observável. Além disso, é representado por uma elipse, com o nome do caso de uso dentro ou abaixo dela. Se houver limites do sistema no diagrama, o caso de uso deve ficar dentro da elipse.

Já os **atores** são a especificação de um papel executado por um usuário ou outro sistema que interage com o assunto (sistema). O ator deve ser externo ao sistema, e este deve ter associações exclusivamente para casos de uso, componentes ou classes, com exceção do fato de um ator poder herdar o papel de outro. O ator é representado por um boneco palito (*stick man*).

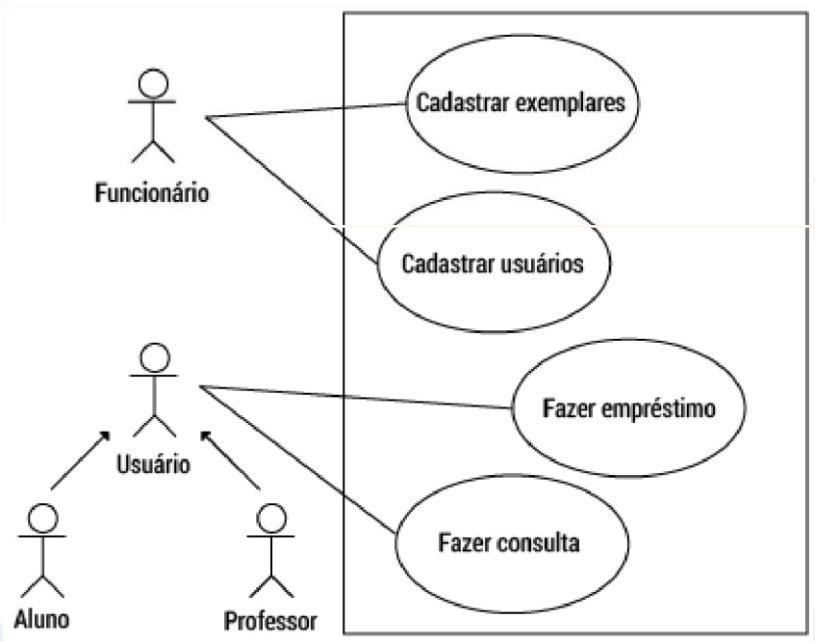


Figura 5 – Caso de uso para relacionamento de biblioteca

A **dependência** (*include*) nada mais é do que uma das formas de interação, isto é, um dado caso de uso pode incluir outro. Incluir é uma relação direta entre dois casos de uso, implicando que o comportamento do caso de uso incluído seja inserido no comportamento do caso de uso inclusor. Essa relação indica uma obrigatoriedade de o caso de uso incluir a funcionalidade do caso de uso incluído.

Já a **associação** (*extend*) é uma outra forma de interação, na qual um caso de uso pode estender outro. Essa relação indica que o comportamento do caso de uso extensor pode ou não ser inserido no caso de uso estendido. Notas ou restrições podem ser associadas ao relacionamento para ilustrar as condições em que esse comportamento será executado.

Portanto, para modelar, estabeleça o contexto do sistema, identificando os atores (usuários e sistemas externos) associados a ele. Para cada ator, considere o comportamento que eles esperam ou requerem do sistema.

Agora, para atingir as suas metas, nomeie cada um desses comportamentos esperados como casos de uso, analise-os tentando identificar comportamentos comuns a mais de um deles e defina essa parte comum como caso de uso

genérico, estabelecendo um **relacionamento de generalização**. Após, tente identificar outros relacionamentos, como a dependência entre casos de uso.

Por fim, modele esses casos de uso, os seus atores e os seus relacionamentos em diagramas de casos de uso e complemente-os com notas que descrevam requisitos não funcionais.

Algumas dicas podem ser muito úteis. São elas:

- Dar nomes que comuniquem o seu propósito
- Quando existirem relacionamentos, distribuir os casos de uso de maneira a minimizar os cruzamentos de linhas
- Organizar os elementos espacialmente de maneira que aqueles que descrevem comportamento associados fiquem mais próximos
- Usar notas para chamar a atenção de características importantes - as notas não são apenas para os requisitos não funcionais
- Não complicar o diagrama, colocando apenas o que é essencial para descrever os requisitos

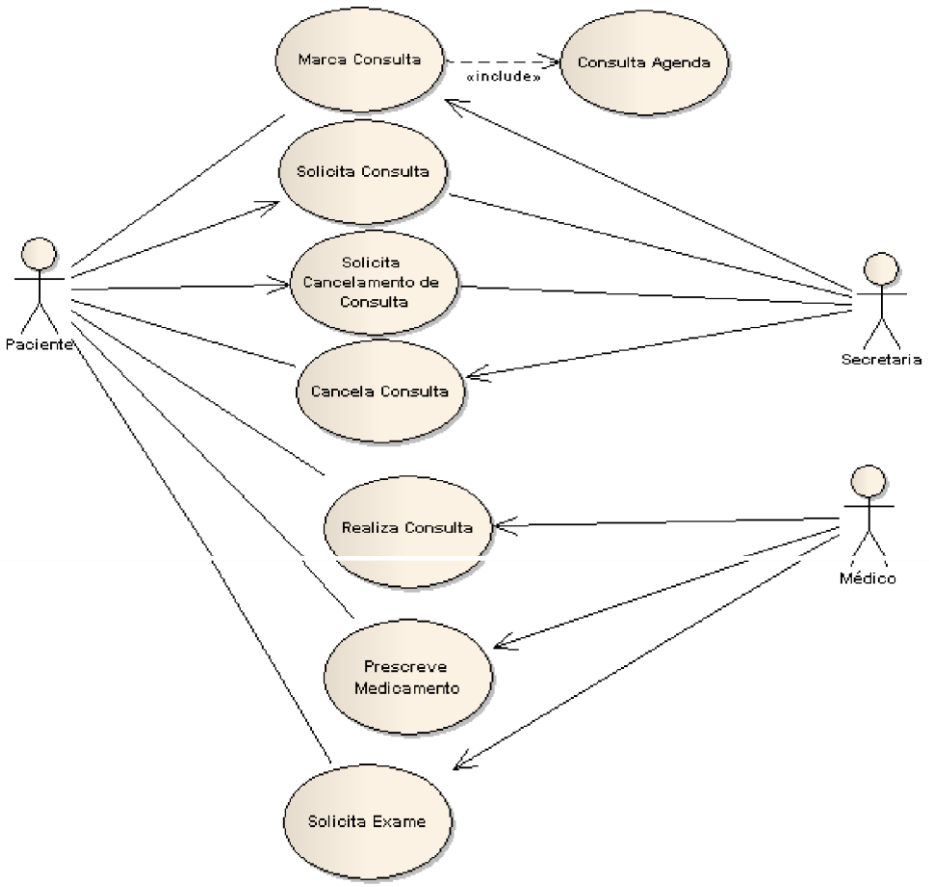


Figura 6 – Caso de uso de agendamento e consulta de atendimentos com relacionamento de inclusão

Portanto, como visto, os analistas têm à disposição uma série de ferramentas e métodos para extrair informação dos usuários. O entendimento dessas ferramentas é indispensável para um analista, pois a construção de material impacta diretamente a construção de um sistema.

Hoje em dia, em um mundo globalizado e com alta concorrência, os detalhes são diferenciais para se destacar no mercado, e, para tanto, deve-se entregar algo que gere impacto ao usuário e que seja um facilitador.

Muitas vezes, há muitos sistemas para um mesmo processo ou uma mesma maneira de trabalhar. Contudo, eles são rasos e não se adequam à realidade do usuário, ressaltando a importância de aplicar métodos como entrevistas e questionários para entender o que o usuário deseja no sistema.

Os profissionais de informática/desenvolvimento devem traduzir o processo manual do usuário em um sistema utilizável e prático.

O levantamento de requisitos e as histórias de usuários ajudam na comunicação com o público, fazendo com que seja possível validar se faz sentido ou não o que se pensou como solução sistêmica para o cliente. Porém, na outra ponta, estão os desenvolvedores, que precisam de uma linguagem mais técnica e funcional para traduzir em código-fonte. Em razão disso, os diagramas vêm para facilitar o entendimento, fazendo um alinhamento entre analista, usuário e desenvolvimento e gerando uma cadeia de qualidade e de entrega assertiva.