

Ferocu, Plano de curso – DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA**0000 – ENGENHARIA DE SOFTWARE I – PRESENCIAL - 80 Aulas**

| Competências Profissionais desenvolvidas neste componente |
|--|
| <ul style="list-style-type: none">Especificar os requisitos, projetar e documentar soluções de software baseadas no conhecimento apropriado de teorias, modelos e técnicas, observando as necessidades dos projetos.Modelar e implantar processos de negócio, propor soluções de TI a fim de aumentar a competitividade das organizações. |

Objetivos de Aprendizagem:

- Identificar as características de Sistemas de Informação, seus tipos, viabilidade técnica, características de custo, valor e qualidade da informação.
- Explicar as características de um sistema, seus componentes e relacionamentos.
- Compreender o ciclo de vida utilizando concepções do modelo cascata.
- Utilizar conceitos da UML na análise de requisitos e na elaboração de diagramas focando na modelagem de sistemas.

Ementa: Introdução à Análise de Sistemas. Modelos de Ciclo de Vida de Software. Modelos de Processos de Desenvolvimento de Software (Modelo em Cascata, Espiral e Prototipagem). Definição e classificação de Requisitos de Software (funcionais e não funcionais). Técnicas de Levantamento de Requisitos. Modelo de Negócios aplicado ao levantamento de Requisitos (Canvas). Estudo de Viabilidade. Técnicas de documentação. Metodologias para desenvolvimento de sistemas.

Metodologia proposta: Aulas Expositivas. Aprendizagem Baseada em Projetos/Problemas. Sala de Aula Invertida. Estudo de Caso Real. Nesta disciplina o professor é responsável por desenvolver um projeto Interdisciplinar integrando as disciplinas de Desenvolvimento Web I e Design Digital, seguindo o Manual de Projetos Interdisciplinares expedido pela CESU.

Instrumentos de avaliação:

Avaliação Formativa: Exercícios para prática. Análise e Resolução de Problemas acompanhado de rubrica de avaliação. Análise da documentação do projeto interdisciplinar.

Avaliação Somativa: Provas. Projetos. Avaliação em pares e Trabalhos Interdisciplinares. Validação do projeto para inclusão no Portfólio Digital do aluno.

Bibliografia Básica:

BEZERRA, Eduardo. **Princípios de Análise e Projeto de Sistemas com UML**. 3 ed. Rio de Janeiro: Elsevier, 2015.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**. 8 ed. São Paulo: McGraw Hill Brasil, 2016.

SOMMERVILLE, Ian. **Engenharia De Software**. 10 ed. São Paulo: Pearson Brasil, 2019.

Bibliografia Complementar:

LARMAN, Craig. **Utilizando UML e padrões**. 3 ed. Porto Alegre: Bookman, 2007.

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação**. 3 ed. Rio de Janeiro: Brasport, 2005.

WASLAWICK Raul. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. 2 ed. Rio de Janeiro: Elsevier, 2010.

Observações a considerar:**1-) Comunicação de alunos com alunos e professores:**

- Um e-mail para a sala é de grande valia para divulgação de material, notícias e etc.
- Criação de grupo nas redes sociais também é interessante.
- Grupo em WhatsApp também é muito interessante.

2-) Uso de celulares:

Para o bom andamento das aulas, recomendo que utilizem os celulares em *vibracall*, para não atrapalhar o andamento da aula.

3-) Material das aulas:

A disciplina trabalha com NOTAS DE AULA que são disponibilizadas ao final de cada aula.

4-) Prazos de trabalhos e atividades:

Toda atividade solicitada terá uma data limite de entrega, de forma alguma tal data será postergada ou seja, se não for entregue até a data limite a mesma receberá nota 0.

5-) Qualidade do material de atividades:

- Impressas ou manuscritas:
Muita atenção na qualidade do que será entregue, atividades sem grampear, faltando nome e número de componentes, rasgadas, amassadas, com rebarba de folha de caderno e etc. serão desconsiderados por mim.
- Digitais:
Ao enviarem atividades para o e-mail da disciplina, SEMPRE no assunto deverá ter o nome da atividade que está sendo enviada, e no corpo do e-mail deverá ter o(s) nome(s) do(s) integrante(s) da atividade, sem estar desta forma a atividade será DESCONSIDERADA.

6-) Critérios de avaliação:

Cada trimestres teremos as seguintes formas de avaliação:

- 1 avaliação teórica;
- 1 avaliação prática (a partir do 2º trimestre, e as turmas do 2º módulo em diante);
- Seminários;
- Trabalhos teóricos e/ou práticos;
- Assiduidade;
- Outras que se fizerem necessário.

Caso o aluno tenha alguma menção I em algum dos trimestres será aplicada uma recuperação, que poderá ser em forma de trabalho prático ou teórico.

1. Introdução a Engenharia de sistemas

Engenharia de sistemas é um campo interdisciplinar da engenharia que foca no desenvolvimento e organização de sistemas artificiais complexos.

Uma definição de Engenharia de Sistemas provém do INCOSE (*International Council of Systems Engineering*) e conceitua que "a Engenharia de Sistemas é uma abordagem interdisciplinar que torna possível a concretização de 'Sistemas' de elevada complexidade. O seu foco encontra-se em definir, de maneira precoce no ciclo de desenvolvimento de um sistema, as necessidades do usuário, bem como as funcionalidades requeridas, realizando a documentação sistemática dos requisitos, e abordando a síntese de projeto e a etapa de validação de forma a considerar o problema completo"

A Análise do problema é o processo de compreensão do problema do mundo real e das necessidades do usuário, juntamente com a proposição de soluções que enderecem essas necessidades.

É comum existirem várias soluções e o trabalho do desenvolvedor consiste exatamente na exploração de todas elas e na identificação daquela que melhor se ajuste ao problema real.

Observação: É importante notar que nem sempre é necessário que exista um problema no sentido estrito, pois, muitas vezes, o desenvolvimento de sistemas é motivado pela vontade de aproveitar oportunidades determinadas pela tecnologia.

Isto coloca o binômio (automatização X inovação) em foco. Em outras palavras, problema e oportunidades são dois lados da mesma moeda. Para fugir desta “neurose”, os desenvolvedores podem se concentrar na análise do problema.

O objetivo da análise do problema é ganhar uma melhor compreensão do problema a ser resolvido, antes de o desenvolvimento se iniciar.

Um problema pode ser definido como a discrepância entre como as coisas são e como se espera que elas sejam.

Nesta visão, existe uma série de maneiras de endereçar o problema além daquela óbvia, de desenvolver um (novo) sistema:

- proporcionar aperfeiçoamentos de sistemas já existentes;
- proporcionar soluções alternativas que não impliquem o desenvolvimento de um novo sistema;
- proporcionar treinamento adicional sobre soluções cobertas, mas não dominadas pelo público-alvo.

O objetivo do exercício de resolução de problemas consiste em ganhar uma melhor compreensão do problema a ser resolvido, antes de o desenvolvimento realmente começar. Os passos específicos para alcançar esta meta são os seguintes:

- 1) ganhar concordância mútua em relação à definição do problema;
- 2) compreender a raiz das causas, ou “o problema por trás do problema”;
- 3) Identificar os stakeholders e os perfis de usuário;
- 4) definir a fronteira da solução do sistema;
- 5) identificar as restrições impostas sobre a solução.

2. Definição do problema

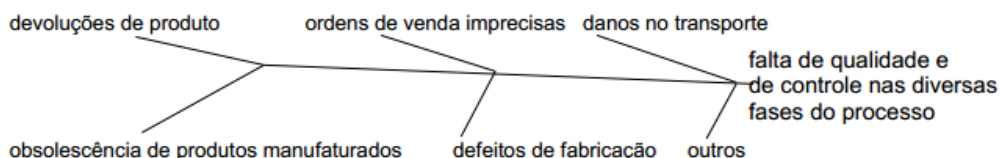
O problema pode ser definido por meio da utilização do formulário de definição de problemas, descrito a seguir.

| Elemento | Descrição |
|---|---|
| <i>O problema de...</i> | Descrição do problema |
| <i>afeta...</i> | Identificação dos <i>stakeholders</i> |
| <i>e resulta em...</i> | Descrição do impacto na atividade, no mundo real, dos <i>stakeholders</i> |
| <i>Poderia ser resolvido por meio de...</i> | Descrição de uma possível solução |
| <i>com os benefícios...</i> | Descrição dos benefícios esperados |

2.1 Identificação da raiz do problema

Com o intuito de descobrir “o problema por trás do problema”, pode ser utilizada uma técnica do “Diagrama de espinha de peixe”.

Exemplo: Imagine uma companhia que realize vendas pela Internet de itens de vários tipos de utilidade doméstica. Os problemas levantados numa primeira conversa informal podem levar à identificação do problema principal. Isto pode ser visto no seguinte Diagrama de espinha de peixe instanciado:



Às vezes não é tão fácil identificar a raiz do problema e, nesses casos, é necessário investigar profundamente cada um dos fatores e atribuir a cada um deles fatores de peso, até se chegar à causa principal.

2.2 Identificação do problema

Uma vez identificado o problema real, cada uma das causas que contribuem para gerar o problema deve ser examinada e, aquelas que tivessem solução computacional devem ter a solução desenvolvida.

Cabe chamar a atenção para o fato de que uma solução computacional pode corresponder, por exemplo, à reelaboração de um formulário de vendas de um sistema já existente.

No exemplo recém descrito, poderia se chegar a esta solução a partir da aplicação do formulário de definição de problemas, por exemplo, à causa “ordens de venda imprecisas”.

2.3 Identificação dos stakeholders

O termo stakeholders se refere a todos os potenciais interessados pelo sistema a ser desenvolvido. Em outras palavras, qualquer um que possa vir a ser materialmente afetado pela implementação de um novo sistema ou aplicação.

Os stakeholders usuários potenciais do sistema são fáceis de serem identificados. Os restantes, usuários indiretos, devem ser procurados na fronteira do ambiente onde a atividade à que o sistema dará apoio é realizada.

Os stakeholders podem ser classificados em 4 categorias, associadas ao sistema e não à atividade-fim da organização:

contribuintes: atores - são os que de fato executam a ideia principal e produzem os resultados; responsáveis – aqueles que concebem a ideia principal são os que mais contribuem com o sistema e mais se beneficiam dele;

fontes: clientes – aqueles que recebem os produtos indiretos do sistema; fornecedores – responsáveis por proporcionar as condições necessárias para o funcionamento do sistema;

mercado: parceiros – colaboram compartilhando recursos e juntando forças para resolver o problema de forma colaborativa; competidores – representam um desafio na medida em que eles concorrem ou entram em conflito com o sistema a ser desenvolvido;

comunidade: legisladores – aqueles responsáveis pelo estabelecimento das regras, sejam elas oficiais ou protocolos sociais; espectadores - compreende a comunidade que receberá os ganhos e ou as perdas decorrentes da utilização do sistema a ser desenvolvido e implantado.

Existem algumas perguntas que podem facilitar a tarefa de identificação dos stakeholders:

Quem (são e ou) serão os usuários do sistema?
Quem são os clientes (comprador) do sistema?
Quem mais será afetado pelas saídas que o sistema vai produzir?
Quem vai avaliar e aprovar o sistema quando ele estiver implementado e implantado?
Há qualquer outro tipo de usuário interno ou externo do sistema cujas necessidades devam ser consideradas?
Quem irá fazer a manutenção do sistema?
Tem mais alguém que possa vir a se importar com o que está sendo definido?

Exemplos de Perfis de usuário e stakeholders para o problema recém discutido.

Perfis de usuário direto: funcionários que entram os pedidos de compra, supervisor de pedidos de compra, pessoal do controle da produção, empregado que elabora a nota de compra.

Outros stakeholders: diretor e equipe da TI da organização, diretor financeiro, gerente de produção.

2.4 Exercícios de fixação do conteúdo.

1-) Utilizando o formulário de definição do problema, faça o levantamento do seguinte problema: Um almoxarifado de pequenas peças (cerca de 500.000 peças já existem em estoque) onde todo o processo é manual, e que 2 almoxarifados, e 1 líder interagem nesta área, fornecendo peças para 20 líderes de uma produção.

2-) Elabore e descreva uma situação problema (implementação de um sistema), onde você possa identificar a raiz do problema através do diagrama de “Espinha de peixe”.

3-) Usando a situação problema do exercício anterior identifique os stakeholders.

Ao final, subam a tarefa em formato .doc na plataforma *Teams* em tarefas, o prazo será até a próxima aula dia 15/02/2022.

3. Definição das fronteiras do sistema

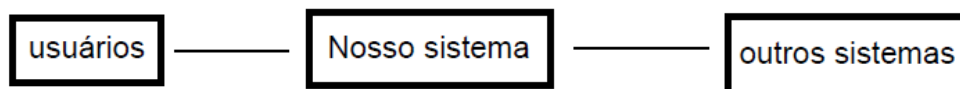
Após identificar todos os perfis envolvidos, devemos definir um sistema que possa endereçar o problema.

Ao considerarmos uma solução potencial, devemos determinar os seus contornos, ou seja, a fronteira entre a solução e o mundo real que a rodeia.

Neste intuito, dividimos o mundo em duas classes de coisas:

1. nosso sistema;
2. coisas que interagem com nosso sistema.

Numa primeira visão, a perspectiva do sistema poderia ser assim representada:



Esta divisão determina atores e papéis associados a eles em relação ao sistema.

Conceito: Um ator é qualquer pessoa externa ao sistema que interage com ele.

A identificação dos atores é uma atividade não trivial. Algumas perguntas ajudam a identificá-los.

Quem vai suprir, usar ou remover informação no ou do sistema?

Quem vai operar o sistema?

Quem vai fazer a sua manutenção?

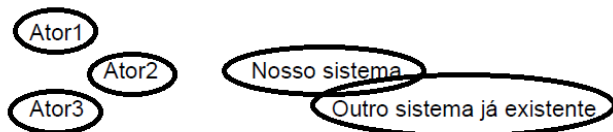
Onde o sistema vai ser usado?

Onde o sistema pega a informação? De onde o sistema retira a informação que necessita?

Que outros sistemas vão interagir com o sistema em desenvolvimento?

Após a obtenção das respostas às questões anteriores, uma visão mais apurada das fronteiras do sistema poderia ser:

O Nosso sistema pode ter diversos atores (ator1, ator2, ator3,...) interagindo com ele e pode, simultaneamente, ter uma interseção com um outro sistema computacional X.



Neste caso, a solução proposta seria composta de um novo sistema a ser desenvolvido e uma modificação a ser efetuada num sistema existente.

4. Determinação das restrições impostas ao sistema

Conceito: Uma restrição é uma limitação imposta ao grau de liberdade que o projetista tem no desenvolvimento da solução do problema.

Existem uma série de fontes potenciais de restrições que devem ser consideradas:

| Fonte | Questões típicas |
|-------------|--|
| econômica | <p>Quais as restrições financeiras aplicáveis?</p> <p>Há custos de produtos vendidos ou alguma consideração sobre a composição de preço?</p> <p>Há questões sobre licenciamento envolvidas?</p> |
| política | <p>Há aspectos políticos internos ou externos que devam ser considerados?</p> <p>Há algum problema ou aspecto crítico entre departamentos?</p> |
| tecnológica | <p>Estamos limitados na nossa escolha de tecnologias?</p> <p>Precisamos nos ater a trabalhar sobre plataformas ou tecnologias já utilizadas?</p> <p>Temos proibição de usar alguma nova tecnologia?</p> <p>Alguém espera que a gente utilize alguns pacotes de software específicos?</p> |
| de sistemas | <p>A solução a ser construída será parte do sistema existente?</p> <p>Devemos manter compatibilidade com as soluções existentes (internas ou externas)?</p> <p>Quais os sistemas operacionais e os ambientes aos que o nosso sistema deve dar suporte?</p> |
| ambiental | <p>Existem restrições ambientais ou regulatórias?</p> <p>Há restrições legais?</p> <p>Quais são os requisitos de segurança?</p> <p>Que outros padrões devemos seguir?</p> |

| | |
|-------------------------------|--|
| temporal e de demais recursos | <p><i>Tem um cronograma definido?</i></p> <p><i>Estamos limitados aos recursos existentes?</i></p> <p><i>Podemos utilizar mão de obra externa?</i></p> <p><i>Podemos estender os recursos? Em caso afirmativo, de forma transitória ou permanente?</i></p> |
|-------------------------------|--|

Uma tabela ajuda a registrar as restrições para posterior exame:

| Risco | Restrição | Rationale (motivo) |
|---------------------------|---|---|
| procedimentos | Uma cópia em papel do recibo deve ser mantida pelo prazo de um ano. | O risco de perda de dados eletrônicos no período de implantação é grande. |
| Desperdício (equipamento) | ... | |
| Desperdício (humano) | | |
| ... | | |

5. Modelagem do negócio

Há uma série de questões cujas respostas devem ficar claras antes de partir para o desenvolvimento de um sistema:

Por que construir um sistema?

Onde ele será instalado?

Como podemos determinar qual a funcionalidade máxima para cada nó particular do sistema?

Quando devemos usar passos de processamento “manual” (humano!) ou outros caminhos que não o de sistemas computacionais?

Quando devemos considerar a reestruturação da organização como passo prévio à resolução do problema?

A técnica de modelagem do negócio nos ajuda a resolver estas questões.

Os objetivos principais da modelagem do negócio são os seguintes:

- 1) entender a estrutura e a dinâmica da organização onde o sistema vai se inserir;
- 2) assegurar que os clientes, os usuários os desenvolvedores tenham consenso sobre o problema;
- 3) entender como empregar novos sistemas para potencializar a produtividade e quais os sistemas existentes podem vir a ser afetados pelo novo sistema.

6. Escolha da técnica certa

A indústria definiu como padrão o Unified Modeling Language (UML).

Conceito: A UML é uma linguagem para visualizar, especificar, construir e documentar os artefatos de um sistema substancialmente de software.

A UML proporciona um conjunto de elementos, notações, relacionamentos e regras de uso que podem ser utilizadas no processo de modelagem de sistemas computacionais.

7. Análise de Requisitos de Sistemas

Roger S. Pressman em “*Engenharia de Software*” explica:

“Uma compreensão completa dos requisitos de software é fundamental para um bem-sucedido desenvolvimento de software. Não importa quão bem projetado ou quão bem codificado seja, um programa mal analisado e especificado desapontará o usuário e trará aborrecimentos ao desenvolvedor.”

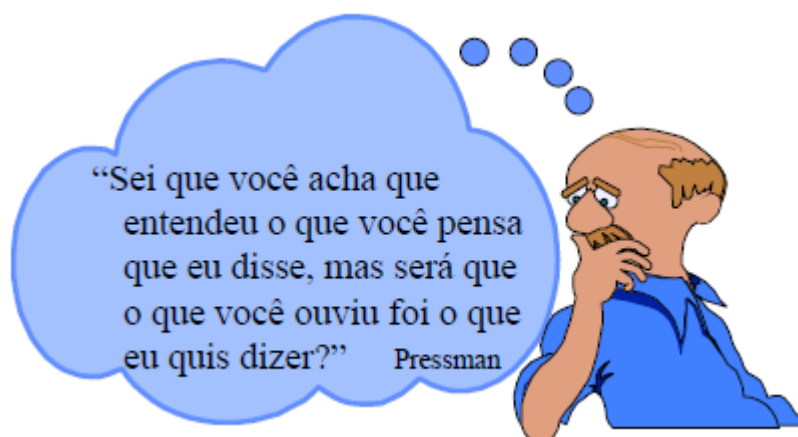
A tarefa de análise de requisitos é um processo de descoberta refinamento, modelagem e especificação. O escopo do software, inicialmente estabelecido pelo engenheiro de sistemas é refinado durante o planejamento do projeto de software, é aperfeiçoado em detalhes. Modelos do fluxo de informação e controle exigido, comportamento operacional e conteúdo de dados são criados. Soluções alternativas são analisadas e atribuídas a vários elementos de software. Tanto o desenvolvedor como o cliente desempenha um papel ativo na análise a especificação de requisitos. O cliente tenta reformular um conceito de função e desempenho de software, às vezes nebuloso, em detalhes concretos. O desenvolvedor age como indagador, consultor e solucionador de problemas.

A análise e especificação de requisitos podem parecer uma tarefa relativamente simples, mas as aparências enganam. O conteúdo de comunicação é muito elevado.

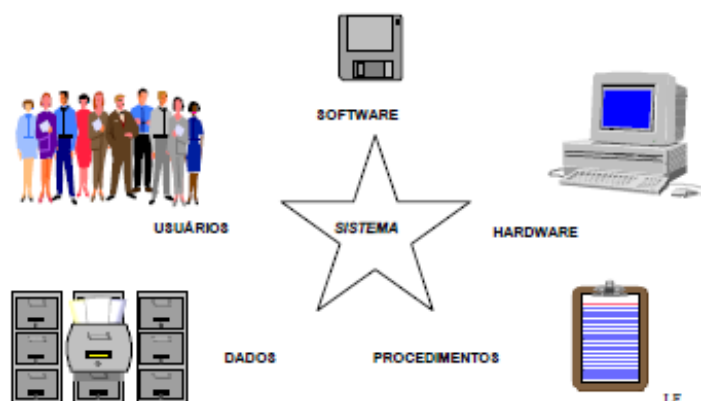
Abundam as chances de interpretações errôneas e informações falsas. A ambiguidade é provável. O dilema com o qual se defronta um engenheiro de software pode ser mais bem entendido repetindo-se a declaração de um cliente anônimo: "Sei que você acredita que entendeu o que acha que eu disse, mas não estou certo de que percebe que aquilo que ouviu não é o que eu pretendia dizer..."

A análise de requisitos é uma tarefa da engenharia de software que efetua a ligação entre a alocação de software em nível de sistema e o projeto de software (Figura a seguir). A análise de requisitos possibilita que o engenheiro de sistemas especifique a função e o desempenho do software, indique a interface do software com outros elementos do sistema e estabeleça quais são as restrições de projeto que o software deve enfrentar. A análise de requisitos permite que o engenheiro de software (muitas vezes chamado de analista nesse papel) aprimore a alocação de software e construa modelos do processo, dos dados e dos domínios comportamentais que serão tratados pelo software.

A análise de requisitos proporciona ao projetista de software uma representação da informação e da função que pode ser traduzida em projeto procedimental, arquitetônico e de dados. Finalmente, a especificação de requisitos proporciona ao desenvolvedor e ao cliente os critérios para avaliar a qualidade logo que o software for construído.



Componentes de um Sistema



Dentro da análise de requisitos temos as seguintes atividades principais:

- identificação das necessidades do usuário
- identificação dos requisitos do sistema
- análise de viabilidade
- análise de custo/benefício
- definição dos componentes do sistema
- alocação das funções nos componentes
- interfaces do sistema

- estruturação da informação

7.1 Exemplo: Processo de classificação de caixas

Em uma fábrica, os produtos são embalados em caixas e colocados sobre uma esteira rolante. No final da esteira, estas caixas são armazenadas em depósitos adequados, de acordo com o tipo do seu conteúdo.

Os tipos do produto são reconhecidos, através do código impresso em uma das faces da caixa.

7.1.1 Alternativa 1

Processo manual:

- um funcionário lê o código da caixa que vem sobre a esteira
- retira a caixa da esteira e armazena no depósito adequado

7.1.2 Alternativa 2

- a classificação é feita através de uma leitora de código de barras e um controlador
- a leitora lê o código quando a caixa passa na sua frente
- o controlador aciona um mecanismo que dirige a caixa para o depósito adequado

7.1.3 Alternativa 3

- a classificação é feita através de uma leitora de código de barras e um robô
- a leitora lê o código quando a caixa passa na sua frente
- o robô transporta a caixa da esteira para o depósito adequado

7.2 Critérios para seleção

7.2.1 critérios de projeto

- custo e prazo adequados
- risco associado a estimativas de custo e prazo

7.2.2 critérios comerciais

- rentabilidade da solução
- aceitação do mercado
- concorrência

7.2.3 critérios técnicos

- existência de tecnologia e recursos
- garantia da obtenção das funções e do desempenho
- manutenção do sistema
- risco associado à tecnologia

7.2.4 critérios para produção

- existência de equipamentos para produção
- disponibilidade de componentes
- adequação da garantia de qualidade

7.2.5 critérios para recursos humanos

- disponibilidade de pessoal treinado
- existência de fatores políticos
- entendimento do cliente em relação ao sistema

7.2.6 critérios para interfaces com ambiente

- adequação da interface do sistema com outros sistemas
- balanceamento da automação em relação ao Usuário

7.2.7 critérios legais

- existência de risco em relação à parte legal
- existência de infração potencial

8. Processo de Análise de Requisitos

A análise de Requisitos ou [Engenharia](#) de Requisitos é um aspecto importante no Gerenciamento de Projetos, é a responsável por coletar dados indispensáveis, necessários, exigências de que o usuário necessite para solucionar um problema e alcançar seus [objetivos](#). Assim como determinar as suas expectativas de um usuário para determinado produto.

Segundo a IEEE (1990) a análise de requisitos é um processo que envolve o estudo das necessidades do usuário para se encontrar uma definição correta ou completa do sistema ou requisito de [software](#).

Essa análise de requisitos é vital para o desenvolvimento do sistema, ela vai determinar o sucesso ou o fracasso do projeto. Os requisitos colhidos devem ser quantitativos, detalhados e relevantes para o projeto. Pois eles fornecerão a referência para validar o produto final, estabelecerão o acordo entre cliente e fornecedor sobre o que o software fará e consequentemente reduzirão os custos de desenvolvimento, pois requisitos mal definidos implicam num retrabalho.

Dentro deste contexto é importante a comunicação e o envolvimento constante com os usuários do software, pois eles influenciarão no resultado final do produto.

A Análise de Requisitos vai consistir em:

- Reconhecer o problema – nesta fase encontra-se a especificação do sistema, o planejamento, o contato do analista com o cliente com a intenção de entender a visão do cliente com relação ao problema.
- Avaliar o problema e a síntese da solução – tem-se o entendimento do problema, e faz-se a identificação das informações que serão necessárias ao usuário, identificação das informações que serão necessárias ao sistema e a seleção da melhor solução possível dentro das soluções propostas.
- Modelar (Modelagem) – é um recurso usado para o suporte da síntese da solução, o modelo vai apresentar ferramentas que facilitarão o entendimento do sistema, como as funcionalidades, informações e comportamento do sistema.
- Especificar os requisitos – consolida funções, interfaces, desempenho, o contexto e as restrições do sistema.
- Revisar (Revisão) – Juntos, cliente e analista, avaliarão o objetivo do projeto com o intuito de eliminar possíveis redundâncias, inconsistências e omissões do sistema, obtendo uma mesma visão.

8.1 Tipos de requisitos

Requisitos do projeto – requisitos do negócio, gerenciamento e entrega do produto.

Requisitos do produto – requisitos técnicos, de segurança, de desempenho, etc.

Requisitos funcionais: eles vão estabelecer como o sistema vai agir, e o que deve fazer, as funcionalidades e serviços do sistema, devendo ser descritos detalhadamente. Nesta fase, pode-se usar o MER, modelos de casos de uso, fluxogramas, para facilitar o entendimento das funções do sistema.

Requisitos não funcionais: definem as propriedades do sistema e suas restrições. Ex.: a confiabilidade do sistema, o tempo de resposta do [programa](#), o espaço em disco.

8.2 Técnicas de Análise de Requisitos

Entrevista – Consiste na investigação direta com os clientes e usuários, fazendo entrevistas para coletar suas expectativas.

Brainstorming – conhecida também como “Tempestade de idéias” essa técnica consiste em coletar idéias, não descartar ou desprezar qualquer tipo de idéia que surja no processo e selecionar a melhor ideia possível podendo ser uma combinação de idéias.

Questionários e pesquisas – podendo ser os questionários com perguntas fechadas no qual caiba apenas as respostas sim ou não, ou perguntas abertas, na qual possibilita a descrição segundo o usuário de suas atividades e possíveis problemas, levando em consideração as opiniões expressas do usuário.

Observação – o analista dispõe de tempo para observar as atividades do usuário, como utiliza o sistema e como se comporta diante de situações problemáticas.

Neste contexto há outras técnicas tais como workshops, mapas mentais, protótipos, etc.

A análise de requisitos vai ser o processo a determinar as necessidades e interesses dos stakeholders (pessoas ligadas diretamente ou indiretamente ao sistema) para atingir seus objetivos, para que nosso projeto não seja implementado desta forma:

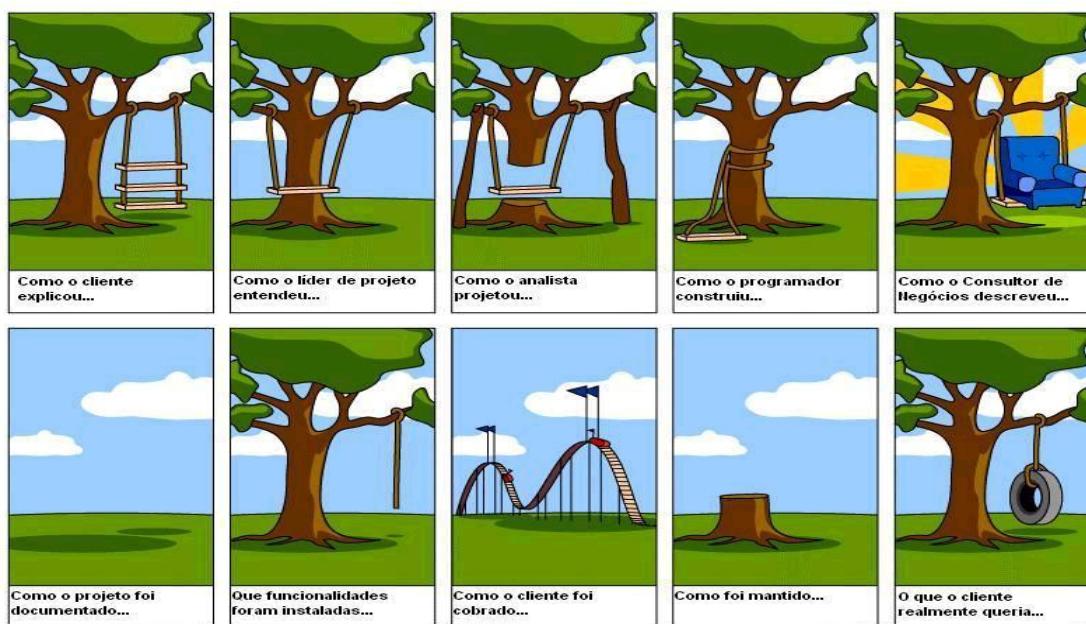


Figura: Visões diferentes de um mesmo projeto

8.3. Motivações e Etapas da Prototipação

A etapa de Análise de Requisitos engloba um conjunto de atividades de fundamental importância para o processo de desenvolvimento de um software e deve ser, portanto, realizada independentemente da abordagem e técnica utilizadas.

Em muitos casos, é possível aplicar-se técnicas de análise de modo a derivar uma representação em papel (ou em arquivo, no caso da utilização de uma ferramenta CASE) que sirva de referência para o projeto do software.

Em outras situações, a partir da coleta de requisitos um modelo do software — um protótipo — pode ser construído de modo a permitir uma melhor avaliação por parte do desenvolvedor e do cliente.

O paradigma da prototipação tem como ponto de partida uma Solicitação de Proposta encaminhada pelo cliente. A partir desta solicitação, os seguintes passos são realizados:

- análise da solicitação para identificar a necessidade da prototipação; normalmente, softwares interativos e/ou gráficos ou softwares que exijam a utilização de algoritmos combinatórios podem ser objeto de prototipação; no entanto, o fator complexidade deve permitir determinar a realização ou não do protótipo; outro ponto que deve ser pesado nesta decisão é se a equipe de desenvolvimento tem experiência e ferramentas adequadas à prototipação;
- especificação resumida dos requisitos, realizada pelo analista, de modo a representar o domínio da informação e os domínios funcionais e comportamentais do software, de modo que a atividade de particionamento do problema possa ser efetuada;
- revisada a especificação resumida dos requisitos, é realizado um projeto do protótipo, concentrando-se principalmente nos aspectos arquitetônicos e de dados e deixando de lado os aspectos procedimentais;
- desenvolvimento do protótipo, se possível a partir da utilização de blocos de construção de software preexistentes (o que na maioria dos casos são de difícil obtenção); por outro lado, a construção de um protótipo pode ser facilitada graças à existência de diversas ferramentas orientadas a esta atividade;
- apresentação do protótipo (testado) ao cliente, para que este possa efetuar sua avaliação; a partir desta avaliação, o cliente pode sugerir extensões ou reformular alguns requisitos de modo a que o software possa melhor corresponder às reais necessidades;
- repetição iterativa dos dois passos anteriores, até que todos os requisitos tenham sido formalizados ou até que o protótipo tenha evoluído na direção de um sistema de produção.

8.3.1 Métodos e Ferramentas de Prototipação

O desenvolvimento de um protótipo é uma atividade que deve ser realizada num tempo relativamente curto, mas de forma a representar fielmente os requisitos essenciais do software a

ser construído. Para conduzir de modo eficiente esta atividade, já se dispõe de três classes de métodos e ferramentas, as quais são apresentadas a seguir:

- as técnicas de quarta geração (4GT), as quais englobam conjuntos de linguagens para geração de relatórios e de consulta a bancos de dados e derivação de aplicações e programas; a área de atuação destas técnicas é atualmente limitada aos sistemas de informação comerciais, embora estejam aparecendo algumas ferramentas orientadas a aplicações de engenharia;
- os componentes de software reusáveis, os quais permitem a "montagem" do protótipo a partir dos "blocos de construção" (building blocks), dos quais não se conhece necessariamente o seu funcionamento interno, mas as suas funções e interfaces são dominadas pelo analista; o uso desta técnica só é possível a partir da construção (ou existência) de uma biblioteca de componentes reusáveis, catalogados para posterior recuperação; em alguns casos, um software existente pode ser adotado como "protótipo" para a construção de um novo produto, mais competitivo e eficiente, o que define uma forma de reusabilidade de software;
- as especificações formais e os ambientes de prototipação, que surgiram em substituição às técnicas baseadas em linguagem natural; as vantagens da utilização destas técnicas são, basicamente, a representação dos requisitos de software numa linguagem padrão, a geração de código executável a partir da especificação e a possibilidade de validação (da parte do cliente) de modo a refinar os requisitos do software.

8.3.2. Prototipação de Alta Fidelidade

Protótipos de alta fidelidade são semelhantes ao produto final. Este tipo de prototipação utiliza a mesma técnica que o sistema final e é indicado quando o objetivo é a venda do sistema ou o teste de problemas técnicos.

Há algumas vantagens em usar a prototipação de alta fidelidade, como: funcionalidades semelhantes as do sistema final, a definição completa do esquema navegacional, um elevado grau de interação com os usuários e a exploração de testes com muito realismo.

No entanto, há algumas desvantagens como, por exemplo: elevado custo e tempo de desenvolvimento, ineficiente para testes de opções de design e levantamento de requisitos.

A prototipação de alta fidelidade poderá ser implementada seguindo um dos métodos:

- ☐ Prototipação Throw-Away, na qual, seu objetivo é identificar e validar requisitos.
- ☐ Prototipação Evolutiva, que tem o objetivo de minimizar o tempo de desenvolvimento do sistema

Pontos positivos

- Equívocos entre usuários do sistema e desenvolvedores são expostos;
- Funcionalidades esquecidas ou confusas são identificadas;
- Maior produtividade do desenvolvedor, pois fica mais fácil saber exatamente o que vai ser desenvolvido;
- Os protótipos podem ser reavaliados por uma pessoa experiente em técnicas de usabilidade, assim criando interfaces gráficas mais produtivas e funcionais;
- Os protótipos podem ainda serem usados para compor documentações técnicas, manuais do usuário e serem utilizadas no treinamento do sistema;
- Maior aproximação do sistema ao objetivo final do projeto;
- Melhoria da qualidade do projeto e conseqüentemente na manutenção;
- Agiliza o processo de validação, aprovação e homologação do sistema.

Pontos negativos

• O tempo de desenvolvimento de protótipos está dependente da experiência das pessoas envolvidas. O tempo dos protótipos iniciais pode ser demorado, enquanto se adquire a experiência de como elaborar protótipos de forma rápida e eficiente; Um processo de coleta eficiente ajuda no repasse das informações para o prototipador do sistema, além de outros fatores, como padrão da interface gráfica do sistema, como por exemplo: padrão de telas de listas, cadastro, relatórios, e menu de navegação.

• Em algumas circunstâncias o desenvolvimento de protótipos atrasa o desenvolvimento e origina um aumento do custo do sistema final. O sistema obtido com base nos resultados da elaboração dos protótipos é melhor mas poderá não ser recompensador. Por esse motivo várias empresas optam pela prototipação de processos complexos, os processos simples, como um simples cadastro, utiliza-se outras técnicas de documentação mais simples e também eficientes;

• Alguns requisitos, como requisitos de "em tempo real" e requisitos não funcionais podem ser difíceis ou mesmo impossíveis de implementar em um protótipo.

Quando usar?

- ☐ Cliente com objetivos gerais sem detalhes;
- ☐ Desenvolvedor não tem certeza da eficiência de um algoritmo;
- ☐ Interação homem-máquina pode não ser aceita pelo cliente.

O que gerar como protótipo?

- ☐ Navegação de telas;
- ☐ Subconjunto de funcionalidade existente no sistema;
- ☐ Toda a funcionalidade existente que será melhorada em um novo esforço de desenvolvimento.

Desvantagens ?

- ☐ O cliente vê a versão em funcionamento e exige alguns acertos para colocar logo em uso.
- ☐ A codificação utilizada para apresentar o protótipo pode no final ser a definitiva.
- ☐ O descartar do protótipo pode ser visto com perda de tempo para o cliente.

9. Perfil do Analista

O analista de requisitos é o analista de sistemas responsável por extrair e especificar em documentos formais o que o sistema deve fazer. Ou seja, é a pessoa que entrevista os futuros usuários do sistema, entende qual o problema a ser automatizado, o que pode ser melhorado, descreve em documentos formais como o sistema deve se comportar de acordo com cada ação do usuário para, posteriormente, serem projetados e codificados pela equipe de desenvolvimento.

Logo de início, vemos que este profissional atenderá pelo menos duas frentes. A do usuário solicitando como o sistema deve funcionar e da equipe de desenvolvimento pedindo informações mais detalhadas para a elaboração do sistema.

Outra qualidade que esse profissional deve possuir é a capacidade de comunicação e de abstração. Digamos que ele deve ser quase um psicólogo para os usuários, escutá-los, entender o perfil do usuário, saber conversar, mas não deixar de ser objetivo. Conseguir extrair o desejo de um usuário muitas vezes frustrado com sistemas que não funcionam é bem complicado. Durante o levantamento de requisitos, o profissional, além de manter a atenção no que está sendo falado, deve tentar imaginar a solução que melhor se adequa ao negócio do cliente e que seja possível desenvolver sem problemas, de acordo com a tecnologia adotada. Quando o usuário está explicando o que o sistema deve fazer, o profissional deve conseguir enxergar possíveis problemas, impactos, modelos, protótipos, conexões com outros sistemas existentes, etc.

Sabemos que, o quanto antes os problemas forem detectados será mais barato e mais fácil resolvê-los. Ter um nível superficial de requisitos do sistema acarreta um risco maior de, quando formos detalhar os requisitos no desenvolvimento, apareçam problemas não antes detectados.

Isso também não quer dizer que um analista de requisitos deve entender sobre tudo e detectar todos os problemas com antecedência, um profissional que tiver interesse em ser excelente, deve se esforçar para possuir essas qualidades. Com certeza essa ação trará resultados bem surpreendentes.

Para tentar provocar ainda mais a discussão podemos ver que um analista de requisitos que possui essas qualidades citadas acima pode estar criando um caminho natural (mas não menos árduo) para, no futuro, galgar o perfil de gerente de projetos. Isso não é uma regra, mas veja que muitos atributos citados anteriormente como boa comunicação, relação interpessoal, capacidade de vislumbrar soluções, entender da tecnologia e procurar riscos e oportunidades é inerente ao perfil de gerente de projetos.

Assim podemos concluir e como sempre citamos em nossas aulas, um Analista não é somente um profissional com o conhecimento em uma determinada linguagem de programação, um analista deve também ter habilidades para:

- Captar conceitos abstratos e sintetizar soluções
- Extrair fatos pertinentes de fontes confusas ou conflitantes
- Compreender o ambiente do cliente
- Saber introduzir a automação no ambiente do cliente
- Comunicar-se bem de forma oral e por escrito.



10. Introdução a UML

Com o passar do tempo e dos erros, novas práticas e metodologias no desenvolvimento de softwares foram criadas a fim de solucionar tais problemas, mas nenhuma prática realmente vingou, pois a falta de um padrão e um método eficaz não correspondiam com a realidade atual.

Em 1995, na tentativa de unir as melhores técnicas de modelagem de software orientado a objeto atualmente existente, é lançado um esboço da versão 0.8 do Processo Unificado (Unified Process), dando então os primeiros passos para a UML (Unified Modeling Language).

Chegando versão 0.9, que incorpora o método OOSE (Object-oriented software engineering), para que o projeto dê um padrão, uma metodologia, não fosse apenas mais uma dentre outras criadas, é criado um pool de empresas formando um consórcio, produzindo assim a versão 1.0 da UML, que foi enviada ao Object Management Group (OMG) em 1997 a fim de torná-la um padrão internacional.

Depois, sucedeu-se versões que teve como alteração estrutural mais brusca a UML 2.0 comparada as atualizações sofridas até a versão 1.5.

O desenvolvimento de sistemas de software de grande porte são suportados por métodos de análise e projeto que modelam esse sistema de modo a fornecer para toda a equipe envolvida (cliente, analista, programador, etc.) uma compreensão única do projeto. A UML (Unified Modeling Language) é o sucessor de um conjunto de métodos de análise e projeto orientados a objeto (OOA&D). A UML está, atualmente, em processo de padronização pela OMG (Object Management Group). A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem é um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. O processo são os passos que devem ser seguidos para se construir o projeto. O modelo de linguagem é uma parte muito importante do método. Corresponde ao ponto principal da comunicação. Se uma pessoa quer conversar sobre o projeto, como outra pessoa, é através do modelo de linguagem que elas se entendem. Nessa hora, o processo não é utilizado. A UML define uma notação e um meta-modelo. A notação são todos os elementos de representação gráfica vistos no modelo (retângulo, setas, o texto, etc.), é a sintaxe do modelo de linguagem. A notação do diagrama de classe define a representação de itens e conceitos tais como: classe, associação e multiplicidade. Um meta-modelo é um diagrama de classe que define de maneira mais rigorosa a notação.

10.1 A Linguagem UML

A UML utiliza elementos gráficos para modelar sistemas, representando conceitos e metodologias do paradigma do desenvolvimento orientado a objetos. Estes elementos gráficos estão divididos em duas categorias: Os diagramas estruturais e os diagramas comportamentais. A linguagem possui treze diagramas que são eles:

Comportamental:

Diagrama de Casos de Uso
Diagrama de Atividades
Diagrama de Visão Geral de Interação
Diagrama de Máquina de Estados
Diagrama de Sequência

Diagrama de Comunicação
Diagrama de Temporização

Estrutural:

Diagrama de Classes
Diagrama de Objetos
Diagrama de Pacotes
Diagrama de Estrutura Composta
Diagrama de Componentes
Diagrama de Utilização

Cada diagrama será explicado de acordo com a listagem acima, tendo como foco à simplificação na explicação do assunto abordado, direcionado a explicação prática e objetiva da UML.

10.1.1 Diagrama de Casos de Uso

Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não nos aprofundamos em detalhes técnicos que dizem como o sistema faz.

Este artefato é comumente derivado da especificação de requisitos, que por sua vez não faz parte da UML. Pode ser utilizado também para criar o documento de requisitos.

Diagramas de Casos de Uso são compostos basicamente por quatro partes:

Cenário: Sequência de eventos que acontecem quando um usuário interage com o sistema.

Ator: Usuário do sistema, ou melhor, um tipo de usuário.

Use Case: É uma tarefa ou uma funcionalidade realizada pelo ator (usuário)

Comunicação: é o que liga um ator com um caso de uso

Vamos criar um cenário de exemplo para vermos a notação de um diagrama de caso de uso:

“A clínica médica Saúde Perfeita precisa de um sistema de agendamento de consultas e exames. Um paciente entra em contato com a clínica para marcar consultas visando realizar um check-up anual com seu médico de preferência. A recepcionista procura data e hora disponível mais próxima na agenda do médico e marca as consultas. Posteriormente o paciente realiza a consulta, e nela o médico pode prescrever medicações e exames, caso necessário”.

Com esse cenário simples podemos começar a criar nosso diagrama. Inicialmente vamos definir nossos atores:

- a) Paciente
- b) Secretária
- c) Médico



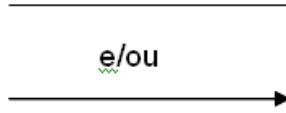
Agora vamos definir algumas ações de cada usuário:

- a) Paciente
 - Solicita Consulta
 - Solicita Cancelamento de Consulta
- b) Secretária
 - Consulta Agenda
 - Marca Consulta
 - Cancela Consulta
- c) Médico
 - Realiza Consulta
 - Prescreve Medicação
 - Solicita Realização de exames

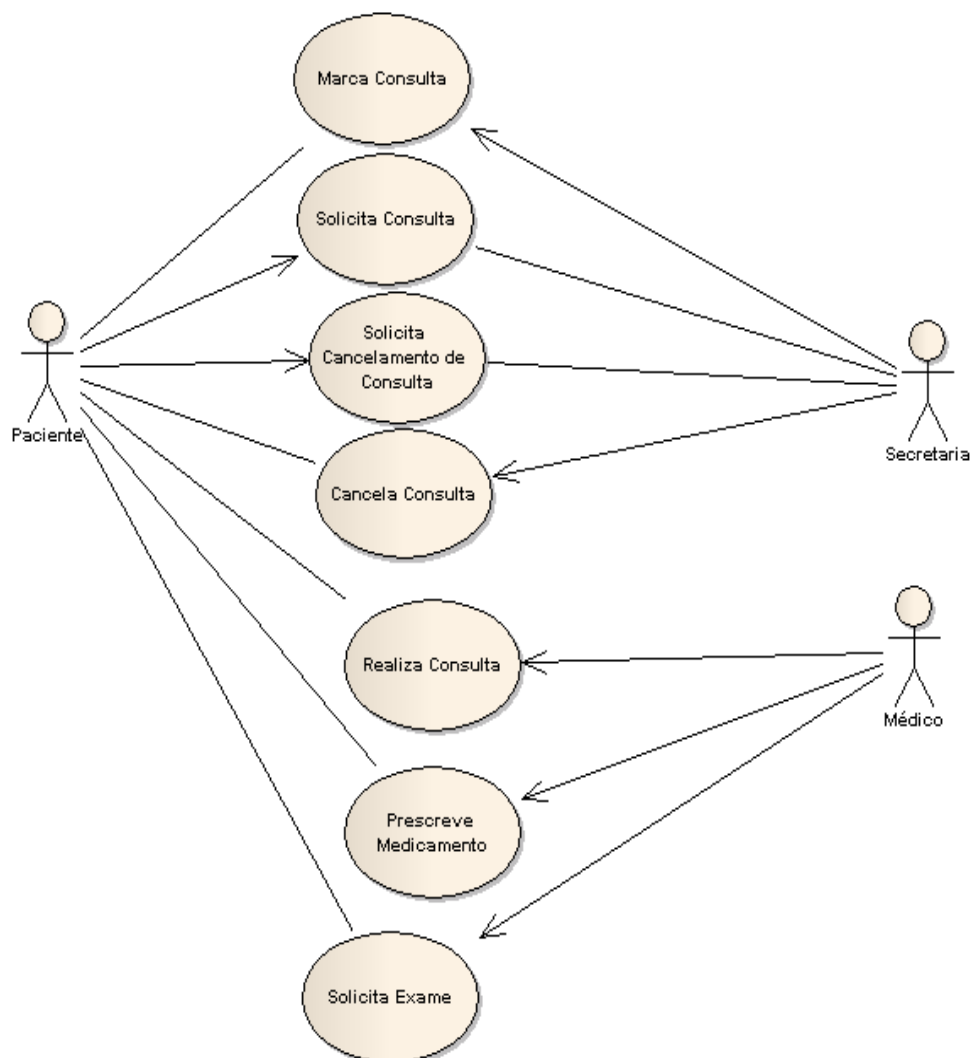
Bom, agora já temos uma relação de atores e ações relacionadas a esses atores. Poderíamos criar um documento textual (como foi feito acima), para registrar nossos atores e funcionalidades. Mas o leitor não concorda que uma imagem vale mais que mil palavras? Pois

bem, podemos expressar tudo o que definimos em um desenho simples utilizando os padrões da UML para documentação de casos de uso.

No quadro abaixo segue a definição de algumas figuras do diagrama:

| Ator | Caso de Uso | Comunicação |
|---|---|---|
|  |  |  |

Podemos agora construir o diagrama:



Como podemos observar esse diagrama composto por desenhos simples descrevem de maneira bem objetiva o que textualmente poderia ficar extenso. Nele vemos as funcionalidades do sistema e as interações dos usuários com elas.

Para melhorar um pouco mais esse diagrama vamos ver o conceito de include>>. Include e extend são relações entre os casos de uso.

Include: seria a relação de um caso de uso que para ter sua funcionalidade executada precisa chamar outro caso de uso.

Extend: Esta relação significa que o caso de uso extendido vai funcionar exatamente como o caso de uso base só que alguns passos novos inseridos no caso de uso extendido.

Tanto um como o outro, são notados como setas tracejadas com o texto `include>>` ou `extend>>`.

Sabendo disso podemos modificar o diagrama inserindo um novo caso de uso “Consultar Agenda”, que será utilizado no caso de uso “Marca Consulta”. Pois a secretária, antes de marcar precisa verificar a disponibilidade da agenda do médico certo?



Com ele podemos trabalhar em três áreas muito importantes nos projetos:

- 1) Definição de Requisitos: Novos casos de usos geralmente geram novos requisitos conforme o sistema vai sendo analisado e modelado;
- 2) Comunicação com os Clientes: Pela sua simplicidade, sua compreensão não exige conhecimentos técnicos, portanto o cliente pode entender muito bem esse diagrama, que auxilia o pessoal técnico na comunicação com clientes
- 3) Geração de Casos de Teste: A junção de todos os cenários para um caso de uso pode sugerir uma bateria de testes para cada cenário

Em maiores detalhes:

- **Atores**



Ator

Um ator é representado por um boneco e um rótulo com o nome do ator. Um ator é um usuário do sistema, que pode ser um usuário humano ou um outro sistema computacional.

- **Caso de uso**



Caso de Uso

Um *caso de uso* é representado por uma elipse e um rótulo com o nome do *caso de uso*. Um *caso de uso* define uma grande função do sistema. A implicação é que uma função pode ser estruturada em outras funções e, portanto, um *caso de uso* pode ser estruturado.

- **Relacionamentos**

- Ajudam a descrever *casos de uso*
- Entre um ator e um *caso de uso*
 - Associação

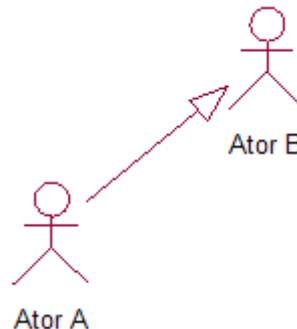


Ator

Caso de Uso

Define uma funcionalidade do sistema do ponto de vista do usuário.

- Entre atores
 - Generalização



- Entre *casos de uso*
 - *Include*

Um relacionamento *include* de um *caso de uso* A para um *caso de uso* B indica que B é essencial para o comportamento de A. Pode ser dito também que B *is_part_of* A.

- *Extend*

Um relacionamento *extend* de um *caso de uso* B para um *caso de uso* A indica que o *caso de uso* B pode ser acrescentado para descrever o comportamento de A (não é essencial). A extensão é inserida em um ponto de extensão do *caso de uso* A.

Ponto de extensão em um *caso de uso* é uma indicação de que outros *casos de uso* poderão ser adicionados a ele. Quando o caso de uso for invocado, ele verificará se suas extensões devem ou não serem invocadas.

Você entendeu?! Provavelmente, não. É que *extend* é unanimemente considerado um conceito obscuro.

Vamos a novas explicações.

Quando se especifica B *extends* A, a semântica é:

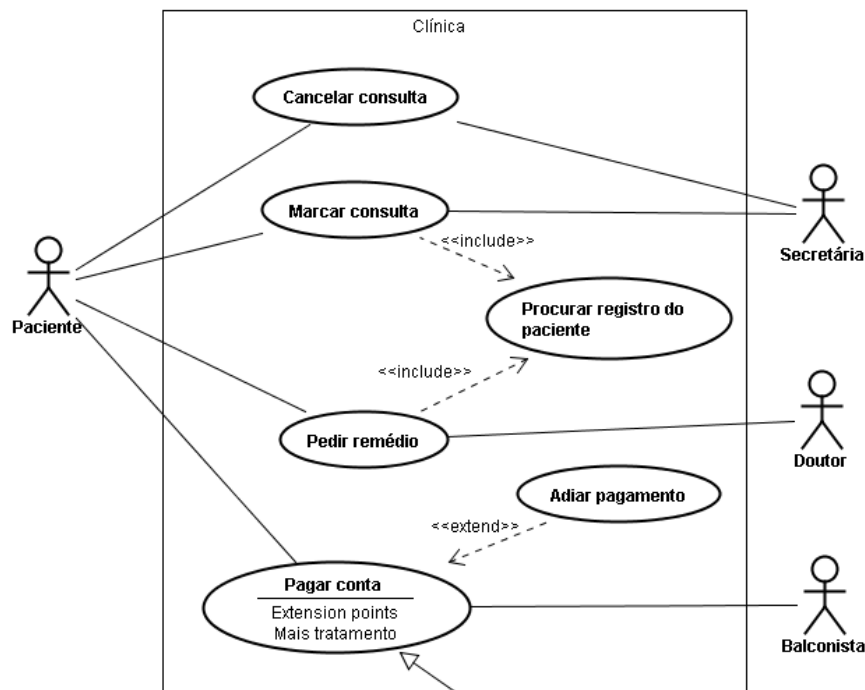
- Dois *casos de uso* são definidos: **A** e **A extended by B**;
 - B é uma variação de A. Contém eventos adicionais, para certas condições;
 - Tem que ser especificado onde B é inserido em A.
- Generalização ou Especialização (é_um) *caso de uso* B é_um *caso de uso* A (A é uma generalização de B, ou B é uma especialização de A).

Um relacionamento entre um caso de uso genérico para um mais específico, que herda todas as características de seu pai.

- **Sistema**

- Limites do sistema: representado por um retângulo envolvendo os *casos de uso* que compõem o sistema.
- Nome do sistema: Localizado dentro do retângulo.

Exemplo 1



Exemplo 2

