

CAPÍTULO 1

UML COMO FERRAMENTA DE ANÁLISE DE SISTEMAS

1	UML - UNIFIED MODELING LANGUAGE	2
2	VISÃO GERAL DOS DIAGRAMAS UML	6
3	DIAGRAMA DE CASOS DE USO.....	14
4	ESTÓRIAS DE USUÁRIOS	43
5	EXERCÍCIOS PRÁTICOS.....	48
6	REFERÊNCIAS	50

1 UML - *UNIFIED MODELING LANGUAGE*

UML - *Unified Modeling Language* é uma linguagem, principalmente gráfica, para a modelagem de sistemas independente de linguagens de programação e de paradigmas de linguagens de programação. Trata-se de uma linguagem para descrever os modelos que podem ser criados durante a análise de sistemas. Aqui, neste livro, a amável leitora há de considerar os modelos como sendo uma representação simplificada de um problema. Esta representação deve conter todas as funcionalidades essenciais daquilo que está sendo modelado.

A UML é o resultado do trabalho de **James Rumbaugh**, **Grady Booch** e **Ivar Jacobson** que no final da década de 1990 reuniram três linguagens diferentes de modelagem em uma gramática e sintaxe comum. Dando origem, tanto ao termo UML quanto a uma empresa, a Rational, para prestar serviços de modelagem de sistemas e de treinamento em UML. Nesta década, estavam e curso duas outras revoluções no desenvolvimento de sistemas: a programação orientada a objetos e o Java.

O paradigma de programação orientada a objetos tem sua origem ligada aos anos 1950, mas a criação de uma linguagem de programação capaz de incluir o conceito de objetos para representar estruturas do mundo real em um programa aparece apenas com a linguagem Simula em 1962. Desenvolvida por Kristen Nygaard e **Ole-Johan Dahl** e, fora uma implementação de 1967, realizada pela IBM a linguagem e o paradigma de programação orientada a objetos ficou praticamente limitada a academia. Completa esta história a linguagem Smalltalk, uma linguagem de programação que adota o paradigma de programação orientada a objetos, dos anos 1970 interpretada. Contudo, este paradigma aparece comercialmente no começo dos anos 1980. Em 1979 **Bjarne Stroustrup** trabalhando na At&T divulga a primeira versão do C++. Uma linguagem desenvolvida para trazer o paradigma de orientação objetos para o nível de desenvolvimento de sistemas ocupado de forma quase exclusiva pelo C, finalmente

a orientação a objetos aparece em revistas, livros e artigos devotados a produção de soluções comerciais. Outras linguagens aparecem para o uso deste paradigma mas nenhuma chama mais atenção que o Java, de 1991. **James Gosling**, trabalhando na Sun Microsystems procurava criar uma linguagem de programação para resolver o problema da compilação para plataformas diferentes e optou por criar uma linguagem, baseada no C++ que fosse puramente orientada a objetos e surge o Java.

A terceira coincidência ocorre no lançamento do SAT/R3 em 1992. A SAP, fornecedora do ERP, o sistema de gestão industrial e empresarial de maior sucesso entre as 500 maiores empresas do planeta. Adota, o Java como linguagem oficial para o desenvolvimento de módulos padronizados para o seu sistema de gestão. Isso provoca a adoção do Java por estas empresas e a disseminação da orientação a objetos no mundo empresarial e comercial. No meio deste turbilhão, surge a UML.

A leitora me perdoe, mas tenho que ressaltar que a UML ainda que tenha sido muito correlacionada ao Java, não está fundamentalmente atrelada a nenhuma linguagem de programação e a nenhum paradigma de programação. Talvez a exceção seja o Diagrama de Classes. Neste diagrama fazemos a modelagem dos objetos de negócio em forma de classes, importando para o mundo dos negócios os conceitos da programação orientada a objetos então parece lógico que no desenvolvimento dos sistemas, utilizemos este diagrama para modelar as classes em um ambiente orientado a objetos. A UML vai muito além de meros objetos.

O objetivo da UML, produzir um modelo do negócio e para tanto faz uso de um conjunto de diagramas que permitem a modelagem de todos os processos envolvidos na criação de uma solução computacional para um determinado problema. Atualmente a UML é mantida pelo *The Object Management Group* que, além de incentivar a adoção desta ferramenta de modelagem, fomenta a adoção do paradigma de programação orientada a objetos.

Linguagem. Eu me referi a UML como linguagem, mais de uma vez. Isso é importante. Por mais que a UML seja um conjunto de diagramas, cada um destes diagramas e símbolos possuem uma sintaxe bem definida. O objetivo de uma

linguagem é a comunicação logo, se você realmente deseja modelar um sistema, precisa considerar que seus diagramas só serão inteligíveis se você seguir as regras da gramática e da sintaxe desta linguagem. A UML não resolve todos os problemas possíveis e nem foi criada para o seu problema. É uma ferramenta genérica e permite um grau de liberdade na criação dos diagramas. A leitora não deve abusar desta liberdade sob a pena de criar um diagrama que só a leitora e Deus entendam.

1.1 Versão e objetivo

A versão 2.51 da UML foi lançada em dezembro de 2017¹ e na seção de escopo deste manual é possível ler o seguinte objetivo explícito:

O objetivo da UML é fornecer a arquitetos de sistemas, engenheiros de software e desenvolvedores de software ferramentas para análise, design e implementação de sistemas baseados em software e para a modelagem de processos de negócio e processos similares. OMG, 2017

O parágrafo acima foi traduzido de forma livre, mas permite o posicionar a UML além de uma linguagem de programação específica e diretamente no cenário da modelagem de processos e negócios. A UML é parte integrante dos processos de engenharia de software, há quem diga parte fundamental. Sendo constituída de 23 diagramas diferentes a UML abrange todos os processos de análise que serão importantes na geração do código para solucionar o problema analisado. Contudo, graças a complexidade do processo de desenvolvimento de software é raro encontrar profissionais especializados em todos estes diagramas. O que vemos com frequência são profissionais usando dois ou três destes diagramas por toda a sua vida profissional. Cada um dos diagramas da UML é voltado para uma parte do processo é necessário que o profissional responsável pela modelagem conheça a

¹ UML 2.51 – Norma disponível em: <https://www.omg.org/spec/UML/About-UML/#docs-normative-supporting>

área especificada no diagrama que irá utilizar na modelagem da sua parte do problema para que realmente entenda a linguagem e possa modelar o problema.

1.2 Natureza dos Diagramas

Didaticamente podemos classificar os diagramas da UM em quatro classes: estruturais, comportamentais, estáticos e dinâmicos. Não se trata de classes mutuamente exclusivas e não existe uma linha rígida que limite a fronteira entre estas classes. Esta classificação serve para a leitora ter uma noção da função do diagrama em relação ao ambiente de análise.

Diagramas Estruturais versus Diagramas Comportamentais: as características estruturais indicam como o negócio está estruturado enquanto os comportamentais indicam o fluxo de informação no sistema.

Diagramas Estáticos versus Diagramas Dinâmicos: diagramas que modelam condições não relacionadas a mudança do tempo são estáticos, todos os outros, são dinâmicos. Mesmo que indiquem um instante congelado no tempo.

Um diagrama estrutural pode ser estático ou dinâmico, mas não será comportamental. A Figura 1. Apresenta alguns diagramas da UML segundo sua natureza.

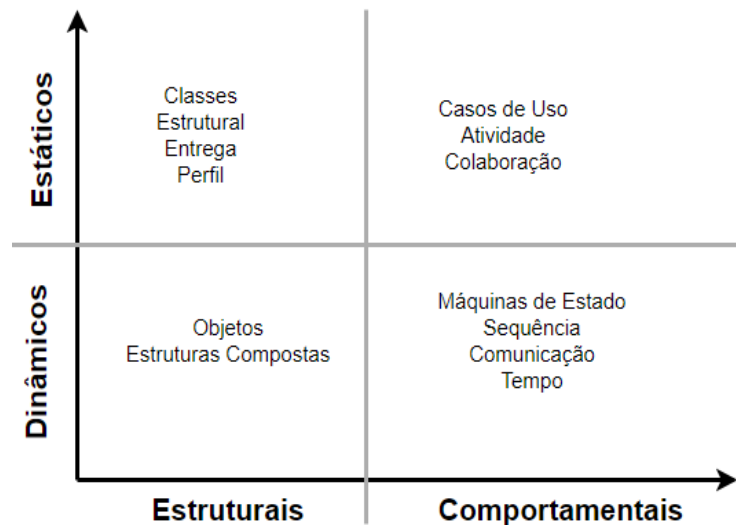


Figura 1 - Natureza dos Diagramas UML. Fonte: adaptado de (UNHELKAR,2010)

Se formos nos ater especificamente ao que o padrão diz veremos que a UML 2.51 utiliza apenas duas classes para definir a natureza dos diagramas: estruturais e comportamentais. Como disse antes, do ponto de vista da análise de sistemas essa classificação não tem relevância serve apenas para que a leitora possa criar o entendimento necessário. Além disso, o padrão não impede a mistura de diagramas, ou símbolos para a criação de um diagrama mais complexo que atenda às necessidades de um projeto específico e que não exista na UML. Use esta liberdade com parcimônia. Apesar do padrão 2.51 não impor limites rígidos ao uso dos símbolos e diagramas propostos, existem limitações que são impostas pelas ferramentas que usamos para criar os diagramas. Muitas destas ferramentas limitam os símbolos que podemos usar em cada diagrama. E acabam tornando rígida uma atividade que deveria ser flexível.

2 VISÃO GERAL DOS DIAGRAMAS UML

Na UML 2.51 estão definidos 23 diagramas diferentes. Alguns são mais comuns durante o processo de modelagem. Por comuns, entenda mais frequentemente utilizados. Isto ocorre porque percebemos que a rigidez dos projetos de engenharia não é adequada ao desenvolvimento de software. Parece estranho, mas esta percepção começou no final da década de 1990 e tomou o Século XXI. Hoje, o processo de análise e desenvolvimento de software está mais ligado as técnicas ágeis que ao formalismo da engenharia.

Parte do sucesso destas técnicas ágeis está na redução do tempo de desenvolvimento e entrega de produtos graças a simplificação da documentação. Um dos mantras que se ouve em ambientes de desenvolvimento nos anos 10 e 20 do Século XXI é *menos documentação, mais código*. Sem querer defender, ou criticar, qualquer modelo de desenvolvimento, é preciso lembrar a leitora que não existe almoço grátis. Quando alguém renuncia à documentação está trocando esta

por confiança. Confiança nos *stakeholders*² e nos desenvolvedores. E uma parte significativa da indústria de desenvolvimento está tendo problemas para a entrega dos produtos, muito parecidos com os que existiam antes da criação da UML. Hoje, procuramos um equilíbrio entre o que indispensável na documentação, ainda que tome tempo e o que pode ser feito de forma mais... como direi? Ágil. Neste cenário de mudanças contínuas, alguns diagramas se destacam.

2.1 Diagramas de Caso de Uso

Use Case Diagrams ou Diagrama de Caso de Uso representam um modelo da interação entre os atores (*stakeholders* e outros sistemas) com o sistema que está sendo modelado. Captura, em alto nível, os requerimentos do sistema. Não mostra um caso de uso, em vez disso, apresenta os atores, suas funções, suas relações com o sistema e as funcionalidades envolvidas.

Os diagramas de caso de uso são estáticos e comportamentais. Neste diagrama estão representadas as interações entre atores e funcionalidades do sistema e são ferramentas importantes para a definição e a compreensão dos requerimentos funcionais de um sistema. A parte comportamental do diagrama de casos de uso fica clara na indicação gráfica das ações que estão representadas. Por outro lado, este diagrama deve ser classificado como estático porque as relações entre atores e o sistema, entre atores e atores e mesmo entre diferentes diagramas de caso de uso, em relação a passagem do tempo, não estão explicitadas.

O conjunto de informações que serão trocadas entre atores, funcionalidades do sistema e sistemas externos que serão representados nos diagramas de casos de uso devem ser registrados em documentos textuais de suporte a criação deste diagrama. Uma parte importante desta documentação tem origem na elicitação de

² A palavra *stakeholder* tem origem nas regras do duelo britânico, mas hoje em dia representa o conjunto de pessoas que afetam, ou serão afetadas, pelo desenvolvimento de um sistema. O conceito é mais amplo, entretanto, para a análise de sistemas isso basta.

requisitos e requerimentos do sistema. A Figura 2 apresenta um exemplo deste diagrama.

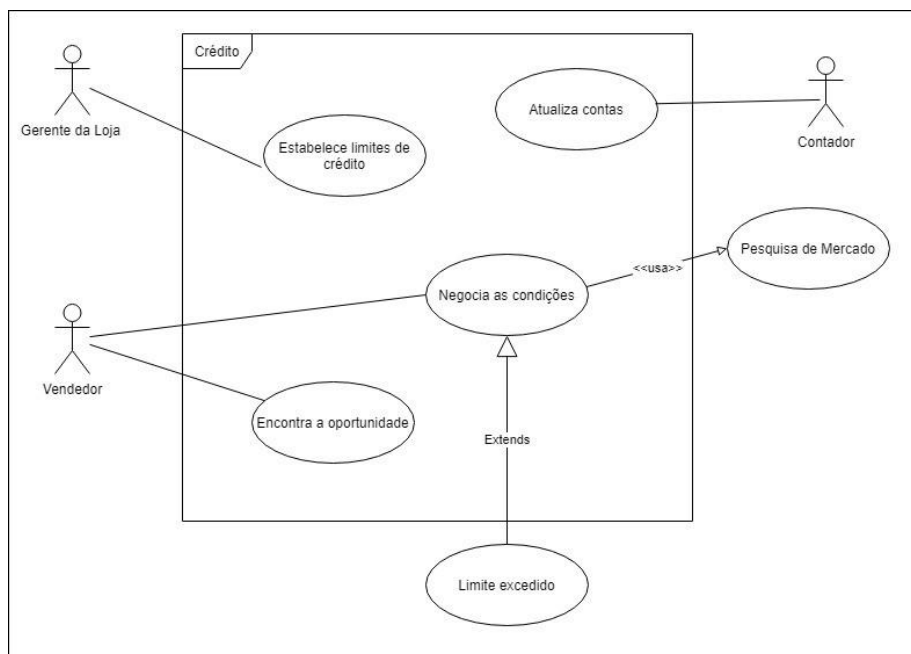


Figura 2 - Exemplo de diagrama de Casos de Uso

O diagrama apresentado na Figura 2 mostra três atores (gerente de loja, vendedor e contador), seus casos de uso (*estabelece limites de crédito, ...*) e a interação com estas funcionalidades. As linhas conectando os atores aos casos de uso indicam uma associação, sem representar nenhum tipo de dependência. No diagrama da Figura 2 existe um exemplo de extensão (*extends*) e outro de uso (*uses*), nestes casos estas relações são de dependência e determinam como alguns casos de uso se relacionam com outros casos de uso.

2.2 Diagramas de Atividade

Activity Diagrams ou Diagrama de atividades são como fluxogramas. São diagramas que modelam o fluxo de informação em um determinado processo de negócio. Estes diagramas têm origem nos fluxogramas e documentam o fluxo de informação entre atores, casos de uso e regras de negócio. São divididos em raias,

*swimlanes*³, e comumente, atribuímos estas raias a cada um dos atores envolvidos no processo que estamos modelando. Assim, podemos ver a responsabilidade de cada ator envolvido e a sequência das atividades que serão executadas pelo sistema. A Figura 3 apresenta um exemplo de Diagrama de Atividades.

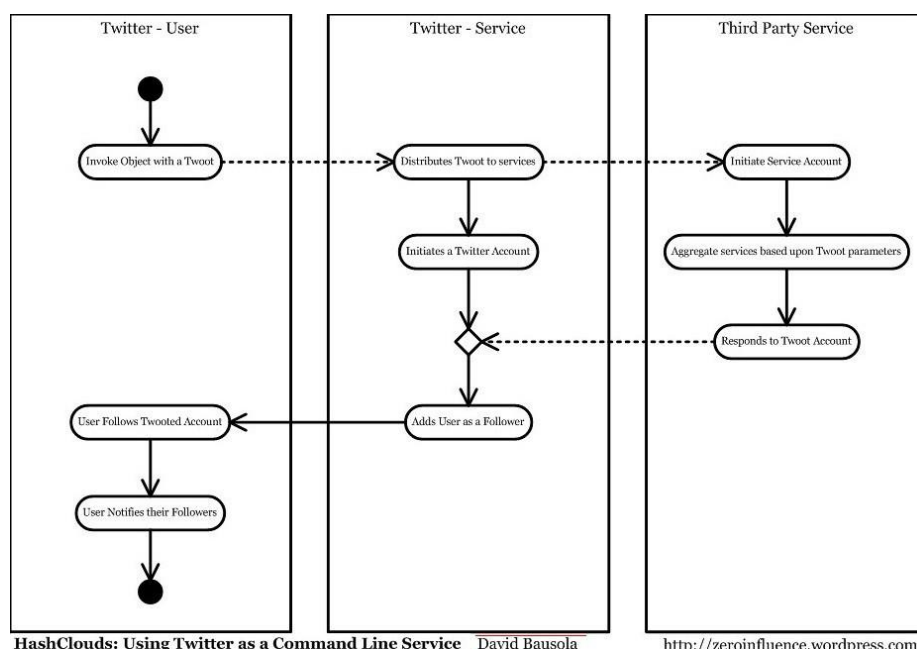


Figura 3 - Exemplo de Diagrama de Atividades. Fonte: adaptado de (BAUSOLA, 2008).

No diagrama mostrado na Figura 3 podemos ver três raias, a ação de cada ator, o fluxo em que estas ações ocorrem no sistema e qual ator é responsável pela ação em um dado momento. Esta divisão em raias, permite o entendimento das atividades que estão acontecendo ao mesmo tempo e, graças a isso, são ferramentas voltadas a modelagem de processos de negócio em ambientes complexos. Estas raias eram chamadas de *swimlanes*, contudo, o padrão mudou. Na UML 2.51 são chamadas de partições, do inglês *partitions*. Neste trabalho vamos usar os termos raias, partições, *swimlanes* e *partitions* de forma livre. Aproveitando da liberdade que a UML permite.

³ Swimlanes, em tradução livre, raias de piscinas para natação.

2.3 Diagramas de Classe

Classe Diagrams ou Diagrama de classe são diagramas estruturais e estáticos que representam os elementos mais importantes nos domínios técnico e de negócio. Podem representar tanto as entidades do negócio quanto as entidades necessárias para a criação do software. Neste caso, a melhor opção é partir para uma linguagem de programação que utilize o paradigma orientado a objetos. Destacam-se nesta classe o C++, o Java, o Python, o Ruby e o Javascript.

Além das classes propriamente ditas, os diagramas de classe representam as relações que eventualmente existam entre as classes. Com cada classe representando uma entidade de negócio, ou uma entidade do universo dos sistemas computacionais, equipamento ou software. O diagrama de classes completo de uma determinada solução irá representar todo o espaço do problema que precisa ser resolvido em um determinado domínio. O conceito de tempo não se aplica ao diagrama de classes, como disse antes, este é um diagrama estático. A Figura 4 apresenta um exemplo de diagrama de classes.

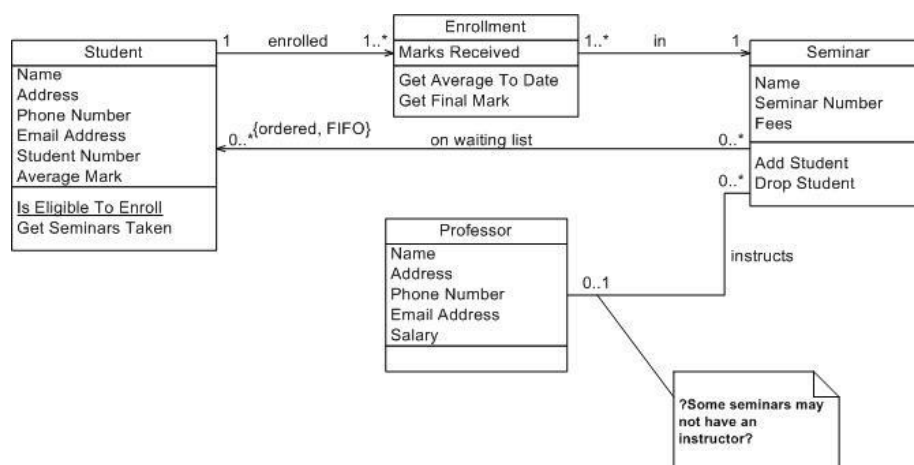


Figura 4 - Exemplo de diagrama de classes. Fonte: adaptado de Wikimedia (2016).

No diagrama mostrado na Figura 4 existem quatro classes: *Student*; *Enrolment*; *Seminar* e *Professor*. Sem mais informações, este parece ser o Diagrama de Classes de um sistema acadêmico para o controle de acesso a

seminários. Neste exemplo é possível ver a representação dos dados e métodos referentes a cada uma das classes. Por exemplo, a classe *Professor* armazena os dados: *Name*, *Seminar Number* e *Fee* possuindo os métodos: *Add Student* e *Drop Student*.

A relação entre algumas classes também pode ser vista na Figura 4 existem, neste exemplo, relações de dependência e posse, e relações de quantidade. Entretanto, vamos deixar para detalhar estas relações quando estivermos estudando este diagrama detalhadamente.

2.4 Diagramas de Sequência

Sequence Diagrams, ou Diagrama de sequência, apresentam as entidades que participam de um determinado caso de uso e as mensagens que são trocadas entre eles ao longo do tempo. O Diagrama de Sequência é um diagrama dinâmico que permite a visualização explícita das mensagens passadas entre entidades em uma interação específica. Trata-se da representação dinâmica das interações apresentadas no Diagrama de Casos de Uso.

A expressão ao longo do tempo, usada no parágrafo anterior ressalta a principal característica deste diagrama: apresentar a sequência em que as informações são trocadas.

Este é um dos diagramas mais populares desde sua criação por Jacobson. Representa o comportamento das entidades em um caso de uso e, graças a isso, são populares tanto entre os analistas de sistemas quanto entre os analistas de negócios.

O Diagrama de Casos de Uso, o Diagrama de Sequência pode ser inferido das observações realizadas durante o processo de eliciação de requerimentos e requisitos. E, assim como o Diagrama de Casos de Uso é voltado para a interação dos atores com o sistema e permite a visualização do processo de troca de informações entre atores e sistema e entre o sistema que está sendo desenvolvido e sistemas de apoio interligados.

A Figura 5 apresenta o exemplo de um Diagrama de Sequência.

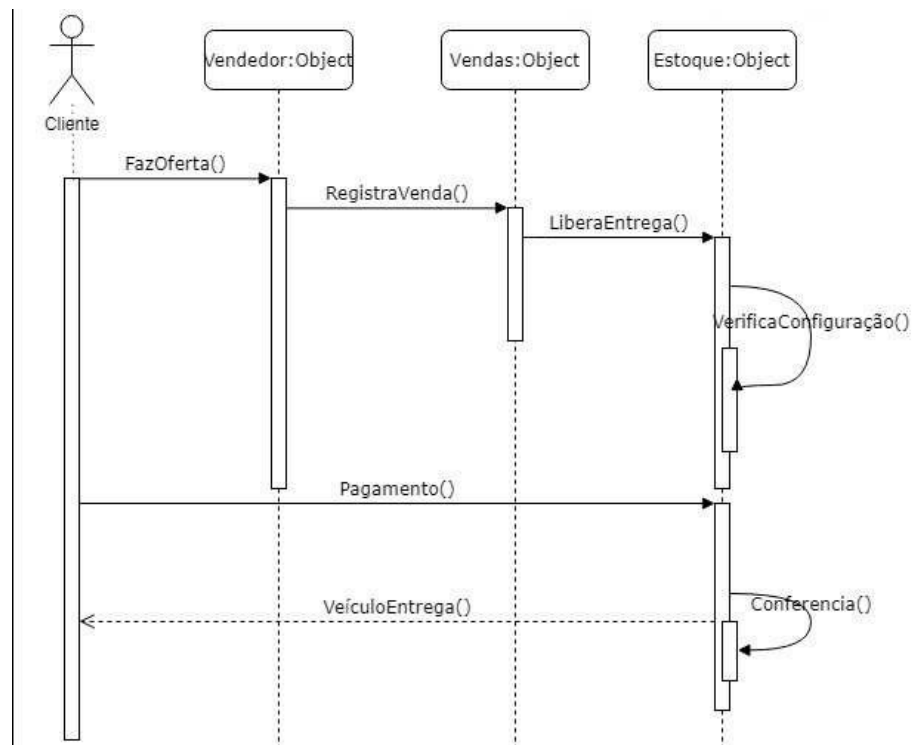


Figura 5 - Exemplo de Diagrama de Sequência.

No exemplo apresentado na Figura 5 é possível ver um ator, cliente, interagindo com objetos de um sistema de vendas de veículos. Este é um sistema totalmente automatizado onde o vendedor é uma abstração do sistema. As linhas verticais pontilhadas indicam o tempo de vida da entidade no sistema, as caixas retangulares verticais indicam uma ocorrência de execução e são utilizadas quando os objetos estão enviados e recebendo mensagens.

Uma mensagem é uma troca de informações entre entidades do sistema, podem ocorrer, neste diagrama, entre atores e objetos, objetos e objetos e entre diagramas. As mensagens são representadas por linhas contínuas chamadas de *links*. Este diagrama representa os eventos em uma sequência temporal mais precisa, e menos granulada, que o Diagrama de Atividade e com ele podemos entender o que acontece no período entre duas mensagens internamente no sistema.

2.5 Diagramas de Objetos

Object Diagrams, ou Diagrama de objetos, representa um determinado momento congelado no tempo destacando as entidades do sistema, objetos e atores, e seu relacionamento neste momento específico. Mostra os objetos em tempo de execução de forma estática. Como se fosse uma fotografia de um processo em andamento. Este diagrama tem uso tanto na modelagem do negócio, quanto na modelagem do sistema e é outro caso onde a relação entre a UML e o paradigma de programação orientado à objetos fica explícito. A Figura 6 mostra um exemplo deste diagrama.

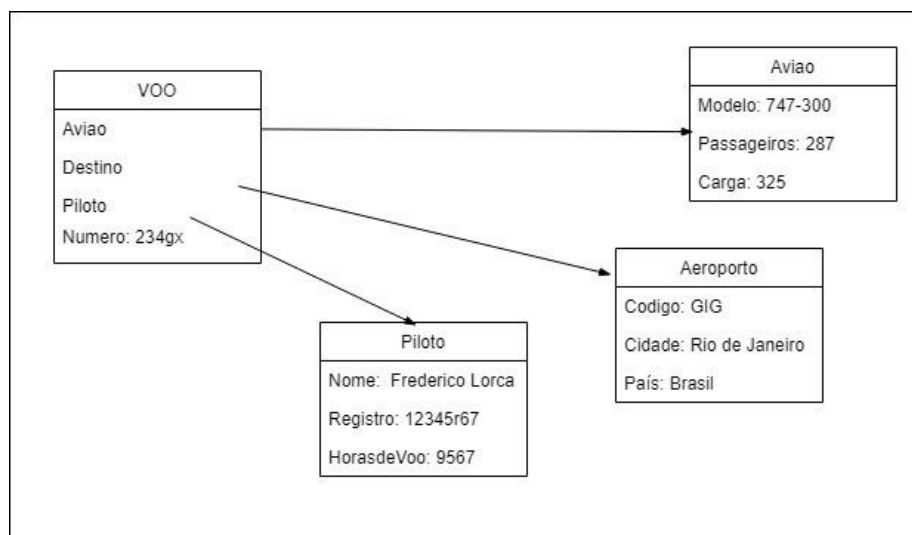


Figura 6 - Exemplo do uso do Diagrama de Objetos.

No exemplo apresentado na Figura 6 é possível ver diversos objetos, sua relação e seu estado em um dado momento do tempo. Não é comum, mas dado a flexibilidade da UML, podemos incluir atores no diagrama de objetos sempre que o estado, ou mensagem, originada de um ator, for importante para a definição do estado que está sendo modelado. A leitora deve lembrar, eu já citei a liberdade de criação que a UML permite. O uso de atores ocorre, principalmente, quando estamos modelando objetos de interfaces homem máquina. Neste momento do processo de análise, as ações dos atores influenciam diretamente o estado dos

objetos que suportam o processo de interface. Ocorre também quando estamos modelando a interface com outros sistemas.

3 DIAGRAMA DE CASOS DE USO

Os Diagramas de Casos de Uso são diagramas de comportamento, estáticos, usados para descrever um conjunto de ações entre atores e o sistema na forma de casos de uso. Estes casos de uso, representam interação com o sistema e todas as interações devem produzir um resultado de valor para os atores envolvidos na interação. ou para algum outro *stakeholder* envolvido no sistema.

Os Diagramas de Caso de uso são representações gráficas dos requisitos, e requerimentos, que foram elicitados durante as atividades de engenharia de requerimentos. Uma parte importante de toda a documentação necessária para a criação dos casos de uso deve ser criada durante o processo de elicitação de requisitos. Se isso acontecer de forma correta, os Diagramas de Caso de uso serão ferramentas para a visualização e validação destes requerimentos.

Em ambientes de análise e desenvolvimento ágil, como o *scrum*, os Diagramas de Casos de Uso e a análise de requisitos são substituídos por um processo menos documentado chamado de Estórias de Usuários. Veremos este processo neste livro e faremos uma comparação entre as duas técnicas. Assim a leitora poderá, por conta própria, adotar a técnica que mais se adeque ao seu problema. Mas agora, precisamos entender as duas principais metáforas do Diagrama de Casos de Uso: atores e casos de uso.

3.1 Atores

Um ator representa uma função do sistema que será realizada por uma pessoa, ou coisa, externa ao sistema. O ator, interage com o sistema para permitir que o sistema atinja seus objetivos. O ator é a função, não a pessoa que está

executado a função e, como tal, um ator pode participar de vários casos de uso. Podemos tentar condensar o conceito de ator em um conjunto de afirmações:

- *Um ator é uma função do sistema que gera valor;*
- *Um ator é uma função do sistema que inicia uma interação com o sistema;*
- *Um ator é uma função do sistema que se beneficia de uma interação com o sistema;*
- *O tempo é um ator, já que existem interações que podem ser iniciadas com o passar do tempo;*
- *Um ator pode ser um outro sistema qualquer que interaja com o sistema que estamos modelando;*
- *Um ator pode ser um dispositivo externo, como uma impressora.*

Um ator pode representar qualquer entidade, qualquer coisa, que envie mensagens para o sistema, ou se preferir, que interaja com o sistema. Toda interação com o sistema é um troca de mensagens. Mesmo quando você está clicando o botão de um mouse. Uma ação totalmente mecânica, para nós seres humanos, estamos enviando a mensagem *clique* para o sistema.

Os atores são o ponto de origem de todo o processo de modelagem da interface com o sistema. Sejam estas interfaces entre humanos e o sistema ou entre sistemas e sistemas.

Como um determinado ator pode participar de vários casos de uso, ele pode representar diversos objetivos e, conseqüentemente, ter expectativas diferentes quanto ao comportamento do sistema. Contudo, é preciso ressaltar que cada caso de uso deve representar uma criação de valor para o ator ou outro *stakeholder*, com um resultado palpável. Se não representar a criação de valor, ou não servir de suporte para a criação de valor, não é caso de uso.

Temos a tendência de só olhar os atores pelo ponto de vista da origem do caso de uso e, é necessário perceber que existem atores que representam o

resultado do caso de uso. São os atores que consomem as informações geradas no sistema. Encontrar e classificar os atores de forma correta é imprescindível para o sucesso da análise do sistema. Este é um processo que está baseado em técnicas de elicitación e na análise da geração de valor pelo sistema.

3.1.1 Encontrando Atores

A descoberta de atores requer prática. A leitora não deve tentar acertar da primeira vez. Por mais experiência que tenha na análise de sistemas. Comece criando uma lista dos atores que você acredita que irão interagir com o sistema. A identificação dos atores começa nos processos de elicitación de requerimentos. Técnicas como entrevista, questionário e observação podem, e devem ser usadas para a identificação dos atores. A lista dos atores deve ser um produto do processo de elicitación de requisitos.

A melhor forma de otimizar o processo de identificação de atores é garantir que um dos resultados do processo de elicitación de requisitos, um dos entregáveis, seja a lista de identificação dos atores que irão interagir com o sistema, já criando uma lista de funções e sistemas externos. Descobrimos atores respondendo perguntas sobre o sistema:

- Quais serão os usuários do sistema? (a função, não a pessoa);
- Existem usuários primários e secundários? Quem são?
- Quais serão os beneficiários das ações do sistema? (a função, não a pessoa);
- Quais são os sistemas, e dispositivos, que vão interagir com o sistema?
- Este sistema sofre a influência de algum evento relativo à passagem do tempo? Que evento é esse? Quando ocorre?

3.1.2 Classificação dos atores

Ao contrário da natureza dos diagramas de UML, a classificação dos atores é um passo importante na sua caracterização e indica a forma como estes irão interagir com o sistema. Podemos começar o processo de classificação dividindo os atores em primários e secundários. De tal forma que:

Atores primários: aqueles para os quais o sistema existe. Estes são os principais beneficiários do sistema.

Atores secundários: são atores cuja função existe no sistema, mas que não iniciam interações ou são beneficiados por elas.

A classificação entre atores primários e secundários depende do sistema, da interação e da visão do analista no momento da modelagem. Uma regra geral determina que os atores primários são aqueles para os quais o sistema está sendo desenvolvido. Por exemplo: pacientes, médicas e enfermeiras em um sistema hospitalar ou vendedor e comprador em um sistema de vendas, ou ainda alunos e professores em um sistema acadêmico. Além de primários e secundários, os atores podem ser diretos e indiretos. De tal forma que:

Atores diretos: são os atores que efetivamente usam o sistema. Voltando ao exemplo do sistema hospitalar podemos classificar o atendente de balcão como um ator direto já que é ele que irá dar entrada nos dados dos pacientes.

Atores indireto: são os atores afetados pelo sistema, mas que não usam o sistema. De volta ao exemplo do sistema hospitalar, um paciente é um ator indireto.

Mesmo que o paciente seja um ator primário já que o sistema será feito para atendê-lo, ele não interage diretamente com o sistema. Se imaginar um sistema de gestão hospitalar moderno, talvez, a única interação dos pacientes diretamente com o sistema esteja nos sistemas com alto nível de automação, nos quais o paciente precisa registrar sua entrada e sua saída de forma eletrônica e, em alguns casos, autorizar algum procedimento via aplicativo móvel. Ainda assim, ele só será classificado como **ator direto** nos casos de uso em que interagir diretamente com o sistema.

Além de principal e secundário, direto e indireto, ainda podemos classificar os atores em abstrato e concreto. Esta última classificação aparece explicitamente

nos Diagramas de Casos de Uso e requer mais cuidado na definição dos conceitos envolvidos.

A UML permite a generalização de atores. Esta generalização indica que atores podem *herdar* as características de outros atores. Novamente recorrendo ao exemplo do sistema hospitalar, podemos ter um ator *paciente_privado* que represente um paciente de plano de saúde, e um ator *paciente_sus* que represente os pacientes que serão atendidos segundo o SUS (Sistema Unificado de Saúde). Estes dois atores podem herdar as características do ator *paciente*. Assim, nos preocuparemos em descrever apenas as interações específicas para estes atores. Desta forma podemos classificar os atores em abstratos e concretos. Tal que:

Atores abstratos: *são os atores que servem como base para a definição das interações e características de outros atores.*

Atores concretos: *são os atores que herdam características de um outro ator.*

No exemplo que discutimos anteriormente sobre os pacientes de um sistema de gestão hospitalar que atenda tanto pacientes particulares quanto pacientes do SUS, os atores *paciente_sus* e *paciente_privado* são concretos enquanto o ator *paciente* é abstrato.

O processo de generalização de atores permite que a complexidade dos diagramas de casos de uso seja reduzida e, como este processo é chamado de herança, precisa ser explicitado de acordo com as normas da UML. Voltando ao nosso exemplo de sistema de gestão hospitalar, todas as interações que são comuns aos dois atores concretos podem ser diagramadas e avaliadas na interação do ator abstrato com o sistema. O processo de generalização é outro processo que está muito ligado aos conceitos da programação orientada à objetos.

Este é um daqueles momentos, a que me referi anteriormente, que a sintaxe e a gramática da UML devem ser obedecidas de forma estrita. A relação entre ator abstrato e ator concreto será feita com uma seta de herança, Esta seta em a ponta

aberta e é composta de uma linha reta, contínua que liga o ator concreto, ou herdeiro, ao ator abstrato. Como pode ser visto na Figura 7.

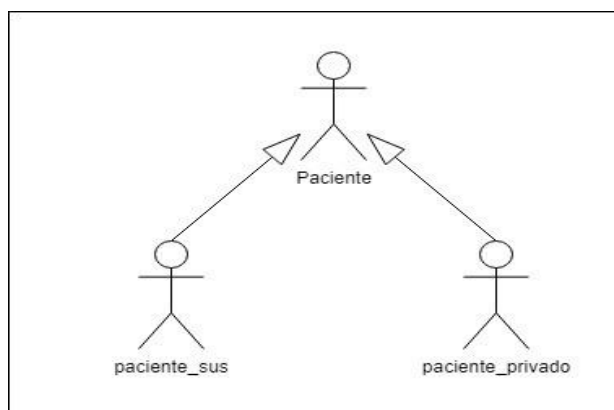


Figura 7 - Exemplo de atores concretos e abstratos do exemplo de sistema de gestão hospitalar

Observe que, na Figura 7, tomei algumas liberdades. Como a UML não define como devemos nomear nossos atores, eu usei a primeira letra maiúscula para indicar o ator abstrato. Uma opção melhor seria colocar uma indicação de que o nome que usamos como identificador se refere a um ator. Assim, por exemplo, se todos os atores dos seus diagramas começarem com a letra A, em qualquer outro diagrama, quando você encontrar um identificador começando por esta letra saberá que é um ator sem precisar recorrer ao Diagrama de Casos de Uso. Neste caso, eu poderia ter nomeados os meus atores como: A1_Paciente, A2_paciente_sus e A3_paciente_privado. **Mas NUNCA use apenas letras números.**

A principal característica de um bom modelo, ou código, é inteligibilidade e a inteligibilidade depende da legibilidade do seu código.

3.1.3 Documentação de atores

O nome dos atores deve ser descritivo da sua função. Este processo de identificação dos atores é responsabilidade do analista que está modelando o sistema. A leitora deve aproveitar esta oportunidade para garantir que todos os identificadores contenham algum significado para quem está modelando e para

quem irá desenvolver o sistema. A prática mais importante para obtenção deste significado, a qualquer momento, é um nome descritivo.

Uma vez que seus atores tenham sido identificados, uma boa prática de modelagem indica que, devemos documentar estes atores. O nome, a classificação, e uma descrição da sua interação e valores esperados devem ser suficientes.

A UML não especifica como devemos documentar os atores. Lembre-se a modelagem é, em si, um processo de documentação. E que o Diagrama bom é aquele que pode ser entendido em seu próprio domínio. Contudo, como nossos atores devem ser originados de um processo de elicitação de requisitos a documentação destes atores pode, e deve, ser criada durante o próprio processo de elicitação.

Em processos ágeis não existe uma documentação explícita sobre cada ator envolvido com o sistema. As Estórias de Usuários, identificam o ator, sua ação e o que ele espera do sistema na interação que está sendo contada. Sem se preocupar com as relações entre atores e as relações entre as interações.

O que ocorre, em sistemas complexos é a criação de estórias de usuários complexas e a divisão destas estórias depois de alguma interação entre os *stakeholders* e os responsáveis com a determinação do que será desenvolvido. Considerando nosso exemplo de sistema de gestão hospitalar, A Tabela 1 apresenta uma sugestão de documentação simplificada.

Campo	Valor
Nome	<i>paciente_sus</i>
Classe	ator concreto herdeiro de <i>paciente</i> .
Descrição	Ator indica as funções que serão exercidas pelos os pacientes que serão tratados no hospital de acordo com as regras determinadas pelo convênio com o SUS. Utilizará o sistema de

	forma direta apenas para registrar sua entrada no hospital e sua saída. Não esquecer a necessidade de assinatura nos documentos relativos ao convênio com o SUS.
Valores	atendimento de qualidade, rápido e seguro, sem a necessidade de enfrentar a burocracia do SUS ou do hospital.
Casos de Uso	UC-2020-08-10-123; UC-2020-07-28-020;

Tabela 1 - Exemplo 1 Documentando o ator paciente_sus

Esta documentação pode crescer para indicar a relação com outros atores, a descrição da interface entre o sistema e o ator, o histórico de modificações na documentação do ator e até mesmo, material de referência. No caso de atores que são outros sistemas, a interface e o histórico de modificações são indispensáveis. Este é um registro de como o sistema deverá ser construído para atender as necessidades de outros sistemas. Ainda em sistemas complexos, você pode precisar especificar os atores com identificadores e números. Estes números são seriais e estão relacionados a data e a ordem do processo de elicitação. A amável leitora fica livre para determinar como estes atores são identificados e me perdoe a audácia de sugerir um formato do tipo 2020-08-10-010. Neste formato temos ano, mês, dia e um número de ordem de elicitação, organizados de forma que a maior parte dos sistemas automáticos irá listar estes atores na ordem em que eles foram criados.

O último campo da Tabela 1, sugestão de documentação para registro dos atores, especifica os casos de uso onde existem interações com este ator. Um bom sistema de documentação, irá buscar os dados deste ator, assim que você começar a criar os Casos de Uso onde ele está envolvido.

3.2 Casos de uso

O Diagrama de Casos de uso é a documentação de um conjunto de interações de um ator com o sistema. O caso de uso é a descrição detalhada de cada uma destas interações. O conjunto de interações explicitadas no Diagrama de Casos de Uso deve prover um resultado para o ator na forma de valor, ele consegue algo do sistema ou consegue que o sistema faça algo por ele, ou na forma de suporte para outra atividade que irá gerar valor para o ator. Cada uma destas atividades será uma interação com o sistema, ou entre sistemas, que irá subsidiar a entrega de valor pelo sistema. Destaque-se que os casos de uso não explicitam como o sistema irá realizar cada uma destas atividades e nem como estas atividades serão desenvolvidas, ou testadas. Um caso de uso específico, uma interação é representado, no diagrama de casos de uso por uma elipse, como pode ser visto na Figura 8.

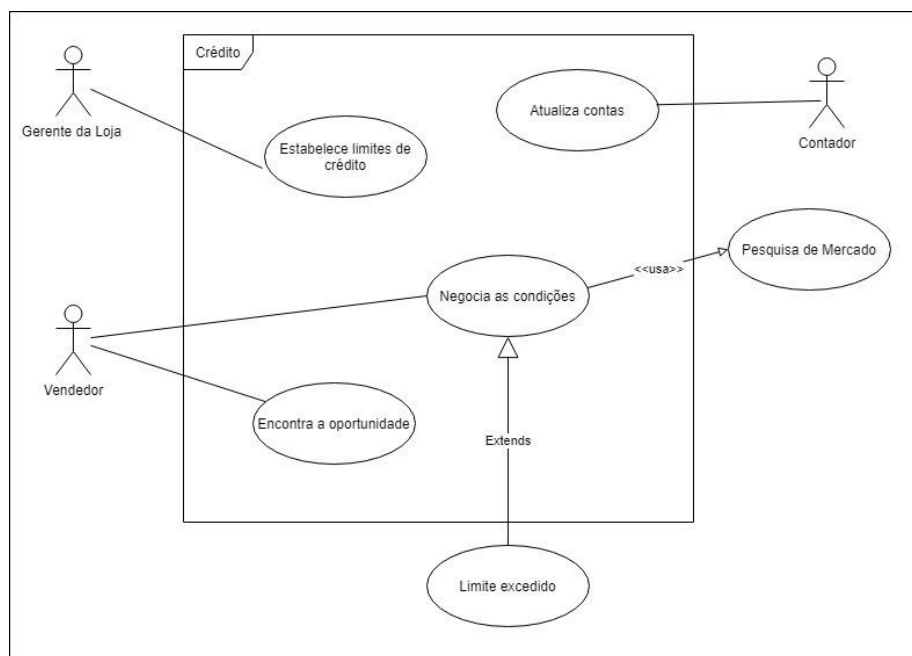


Figura 8 - Exemplo de Diagrama de Casos de Uso mostrando atores e casos de uso.

Na Figura 8 é possível ver três atores e cinco casos de uso. Por exemplo, o ator *gerente da loja* interage com o sistema *estabelecendo limites de crédito*.

Os casos de uso estão fortemente relacionados com as atividades da engenharia de requisitos. A análise de requerimentos deve incluir uma descrição dos processos de interação entre atores e sistemas, explicitando o que o ator espera de cada interação e, dessa forma, além de registrar estas interações, a análise de requisito determina o valor esperado em cada interação.

Casos de uso, não são ferramentas exclusivas da programação orientada a objetos ou das tecnologias de desenvolvimento ágil. Casos de uso são apenas a descrição escrita de uma, ou mais interações com o sistema e podem ser utilizados no mundo da análise de negócios para a otimização dos processos adotados em uma determinada organização. Em todos os casos, cada caso de uso deve explicitar o ator, a interação com o sistema e qual o valor que o sistema irá produzir. De forma extremamente simples, e ágil, se a amável leitora fosse criar um software para um jogo de dados um caso de uso poderia ser:

Caso 1: *O jogador joga os dados, se o resultado for 7 o jogador ganha o jogo.*

Neste exemplo temos um ator, *o jogador*, uma interação: *joga os dados* e um valor: *se o resultado for 7 o jogador ganha o jogo*. Há ainda uma condição: *se o resultado for 7 o jogador ganha o jogo*. Completando estas informações que são todas relacionadas ao ator e ao sistema há uma informação relacionada a análise do problema: caso 1. Observe que esta síntese, apenas uma frase, para uma interação com o sistema possui uma descrição completa da interação para a construção do programa sem explicitar nenhuma informação sobre a construção do sistema. Não diz nada sobre como o jogador joga o dado, nada sobre como ele ganha o jogo, nada sobre como o jogador atua no jogo. Apenas quem ele é, o que ele faz e o resultado esperado. Por falar em resultado, nesta frase está explícito apenas o melhor resultado. O resultado de sucesso. Não há nenhum detalhe sobre o que deve ocorrer se acontecer algum problema e, observe. Está explícito que existe uma possibilidade de o jogador jogar o dado e o resultado não ser sete. Ainda

assim este seria um resultado de sucesso no caso de uso descrito. Então, poderíamos completar nossa análise:

Caso 1: *O jogador joga os dados, se o resultado for 7 o jogador ganha o jogo. O jogador joga os dados, quando o resultado não é 7, o jogador pode jogar mais uma vez. Até um limite de três jogadas. Se em nenhuma destas jogadas o resultado for 7 o jogador perde o jogo.*

Por enquanto, esta descrição, deste único caso é suficiente para desenharmos nosso primeiro Diagrama de Casos de Uso que pode ser visto na Figura 9

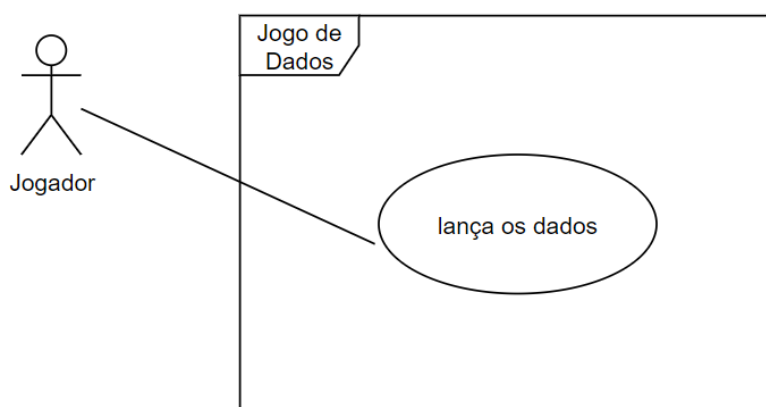


Figura 9 - Exemplo de diagrama de Casos de Uso para o jogo de dados

O Diagrama de Casos de Uso começa no Ator e a forma mais eficiente que temos de encontrar os atores é examinar a lista de atores que foram definidos durante a análise de requisitos, as mesmas entrevistas, os mesmos questionários e os mesmos eventos que usamos para entender os requisitos do sistema podem ser utilizados para a definição dos atores e seus casos de uso. Começamos, pela lista de atores principais vamos detalhando esta lista ao longo do processo. Para isso a leitora deve lembrar que não será fácil acertar na primeira vez e, com isso em mente, vou abordar o problema por meio de uma técnica de levantamento, análise e validação que a cada passagem irá produzir uma descrição mais precisa dos casos de uso.

A documentação dos casos de uso começa na análise de requisitos. Use, durante as entrevistas um modelo de documentação onde seja possível registrar:

1. **o nome do Caso de uso:** que seja uma frase verbal e imperativa. Também é interessante que estes casos de uso sejam identificados com algum identificador único. Talvez um número que contenha a data e a ordem da elicitação. Algo como; UC-2020-08-25-015, onde temos o tipo de identificador único, UC para casos de uso, a data 202-08-25, com ano mês e dia e um número sequencial. Este formato permite que os sistemas eletrônicos listem seus casos de uso na ordem em que eles forem criados. E, acredite, isso será muito útil em sistemas complexos e grandes.
2. **um resumo:** contendo nome, atores envolvidos e resultado esperado de forma simples, no máximo duas linhas de texto. Você pode se referir aos atores pelo identificador único que sugerimos na seção sobre atores.
3. **uma descrição do caso de uso:** para que ele serve, qual o seu objetivo principal, que ator, ou atores estão envolvidos. O objetivo principal é o valor que o caso de uso irá produzir. Descreva este valor em um cenário onde tudo deu certo e o ator conseguiu o valor desejado.
4. **o fluxo de informação para o sucesso:** qual ação deve ser executada e o que se espera em retorno. Esta é a descrição do processo necessário para que o ator consiga o valor.
5. **exceções:** uma descrição do que pode dar errado, problemas de comportamento do usuário e possível falhas. Esta é a descrição do comportamento do sistema em momentos em que algo acontece de errado e o sistema não consegue fornecer o valor desejado pelo ator nesta interação.
6. **Interface:** uma descrição da interface esperada para este determinado caso de uso. Aqui, fica a cargo da leitora. Pode ser uma simples descrição dizendo onde o ator clica como uma descrição detalhada do processo de

interface. Esta opção vai depender do sistema, da gestão do projeto, da interação e do resultado esperado.

7. **Prioridade:** que deve ser definida de comum acordo entre você, que está encarregado da análise do sistema, os desenvolvedores e os *stakeholders*. Esta prioridade determina a ordem de desenvolvimento deste caso de uso e a importância dele para o sucesso do sistema. Esta prioridade é importante e complexa. Por exemplo, na maioria absoluta dos sistemas deve existir um processo de autenticação de usuários. Sem isso não é possível controlar as informações e o uso que será dado a elas. A prioridade desta funcionalidade é alta. Contudo, muito raramente, algum *stakeholder* irá considerar a autenticação dos usuários como uma funcionalidade característica do sucesso do sistema. Este é um caso de uso indispensável, mas de valor relativo.

Você verá, ao longo deste capítulo que o processo de documentação deve ser interativo e incremental. Então, escolha uma ferramenta flexível, que lhe permita registrar os casos de uso à medida que vai progredindo no processo de análise. O sistema que você escolher para este registro deve incluir, por si só, sem a sua intervenção, informações de histórico, recuperação de versões anteriores e autoria. Além disso, a plataforma escolhida para este registro deve ser capaz de permitir a inclusão de material de referência, como legislação, trabalhos anteriores e exemplos. Eu, depois de muitas tentativas, uso o Microsoft Visio, ou o site draw.io, integrados no Github e tenho todas estas funcionalidades. Existem algumas dezenas de sistemas para apoiar a criação dos diagramas da UML, caberá a leitora escolher aquele que lhe atenda.

O processo de documentação dos casos de uso não é padronizado o que permite que você escolha um processo de documentação que seja adequado as suas necessidades para cada projeto. Esta falta de padronização na documentação torna difícil definir o que é um bom caso de uso. Também não existe uma granulação padronizada, ou seja, você irá encontrar, ao longo da sua vida profissional, casos de uso de uma linha ou várias páginas.

Se encontrar casos de uso muito longos desconfie. Os casos de uso devem ser de entendimento fácil para todos os envolvidos nos processos de análise, desenvolvimento e aceitação do sistema. Não foi Einstein quem disse isso, ainda assim é válido: se você não consegue explicar em um parágrafo, ainda não entendeu o problema.

Usar apenas uma frase também não é uma boa opção. Casos de uso devem ser descritivos e excetuando-se as interações que são comuns a todos os sistemas, uma frase não será suficiente para descrever a interação, o ator, as exceções e os objetivos que devem ser alcançados. Existem, contudo, algumas sugestões de boas práticas, que veremos ao longo deste capítulo que até podem não resolver o problema, mas vão levar a leitora a descobrir o melhor caminho para a solução.

Por fim, lembre-se, não importa quão boa seja a sua documentação. Casos de uso não são ferramentas de desenvolvimento. O conjunto de casos de uso comporá o conhecimento que você irá usar para modelar o sistema e, depois disso, desenvolver o código, interfaces, testes e ferramentas de entrega. Todas estas atividades têm origem nos casos de uso, mas os casos de uso não são as ferramentas que a amável leitora irá precisar para realizar todas estas tarefas.

3.2.1 Sintaxe dos casos de uso

A UML é uma linguagem e, como tal, possui uma sintaxe que precisa se obedecida para que a semântica possa ser percebida. É a semântica que dá significado ao que fazemos. No caso da UML, os componentes que podemos utilizar em cada diagrama, sua forma, descrição e a relação entre eles formam o conjunto de sintaxe e semântica que devemos utilizar para representar a parte do problema que estamos tentando modelar. No caso, devemos representar os casos de uso obedecendo as regras da UML 2.51 se quisermos que nosso trabalho seja entendido por outros analistas e desenvolvedores.

Existe um grau de liberdade envolvido na criação de qualquer um dos diagramas da UML. Esta liberdade pode ser utilizada, com restrições. Assim, a melhor forma de se fazer um diagrama, principalmente em uma equipe nova, ou

quando estamos começando a usar os diagramas da UML é utilizar o máximo possível aquilo que é sugerido para cada diagrama. A Figura 10 apresenta os componentes preconizados para a criação de Diagramas de Casos de Uso.

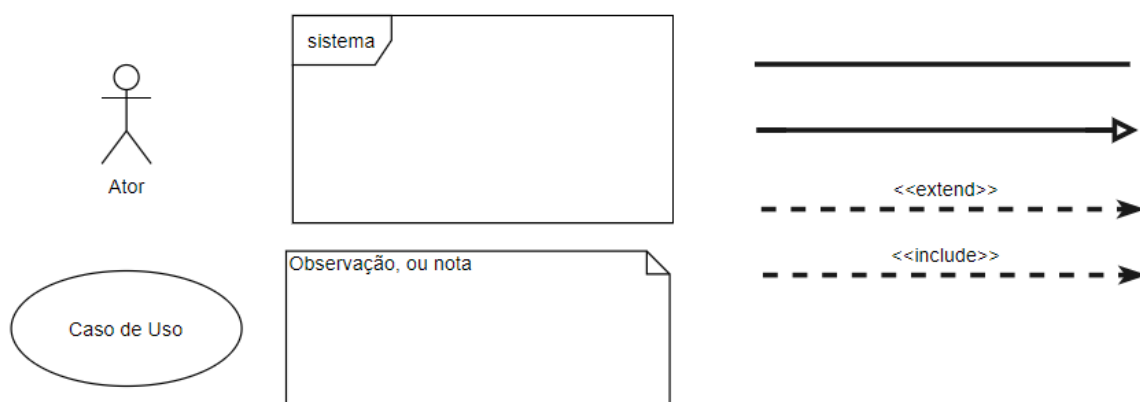
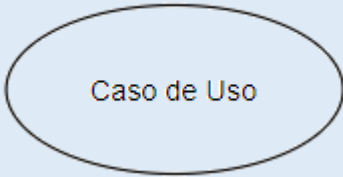
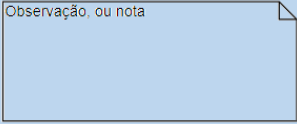
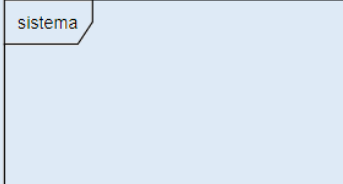




Figura 10 - Componentes que podem ser utilizados para criar um Diagrama de Casos de Uso.

É preciso destacar a representação oval do caso de uso, as notas, os limites do sistema e as relações representadas pelas setas e pela linha contínua. O ator já é nosso conhecido. Da mesma forma, a seta com a ponta aberta que representa a generalização de atores. A Tabela 2 apresenta uma descrição de cada um destes componentes.

Componente	Descrição	Representação Gráfica
Ator	Representa uma função do sistema. Um conjunto de pessoas, ou sistemas, que pode interagir com o sistema em busca de algum valor, ou para a criação das estruturas necessárias para a criação de valor.	

Caso de Uso	Representa, com uma frase imperativa, uma ação do ator no sistema. Ou alguma outra interação que seja provocada por esta interação, seja provocada por esta interação. Ou ainda, alguma função que seja necessária para a realização da interação com o usuário.	
Nota	Um campo para você colocar qualquer observação que seja necessária ao entendimento do caso de uso.	
Limites do Sistema	O contorno que engloba todas as funcionalidades que serão realizadas pelo sistema. Um caso de uso pode ter vários limites de sistema se tivermos vários sistemas interagindo.	
Associação	A reta contínua, sem pontas, que liga o ator ao caso de uso. Esta é a reta que indica o quê cada ator pode fazer no sistema, ligando o ator a funcionalidade.	
Generalização	Uma seta com a ponta aberta que irá ligar um ator concreto a um ator abstrato. A generalização permite que atores abstratos representem todas as funcionalidades comuns a um conjunto de atores	

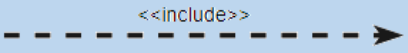
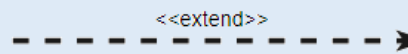
	concretos. Este é um mecanismo de herança	
Inclusão	<i>Include</i> . Representado por uma seta tracejada com ponta cheia. Sobreposta do símbolo <code><<include>></code> . Representa uma relação direta entre dois casos de uso. Onde um precisa incluir o outro para ter sucesso.	
Extensão	<i>Extend</i> . Representado por uma seta tracejada de ponta cheia. Sobreposta pelo símbolo <code><<extend>></code> . Representa a relação direta entre dois casos de uso onde um completa a funcionalidade do outro.	

Tabela 2 - Detalhamento dos componentes do Diagrama de Casos de Uso

A associação é a linha que liga o ator ao caso de uso esta linha indica a relação entre o ator e a funcionalidade. Um * colocado perto do ator, perto do caso de uso, ou perto de ambos indica uma relação de multiplicidade. No fragmento de caso de uso mostrado na Figura o * indica que o gerente pode aceitar ou rejeitar várias ofertas.

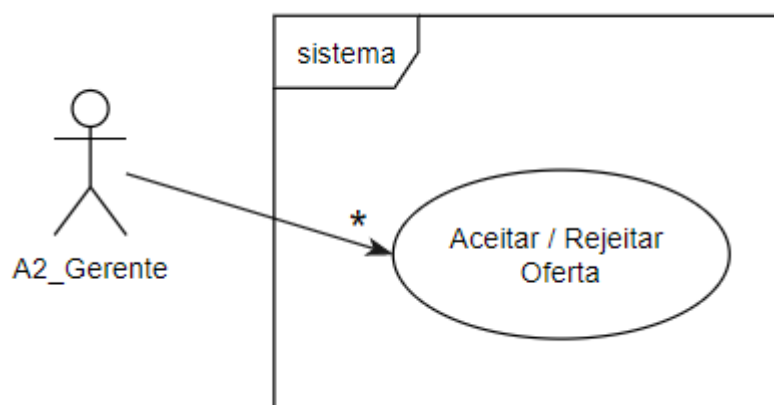


Figura 11 - Exemplo de associação múltipla entre ator e caso de uso.

A recíproca é válida, ainda que não seja muito utilizada. Um * próximo de um ator indica que vários atores daquela função poderão acessar o mesmo caso de uso.

O asterisco pode ser substituído, tanto no lado do ator quanto no lado do caso de uso, se for o caso, pela explicitação de um multiplicador. O número 1, para indicar apenas um uso, o número 2 para representar dois usos etc. Este multiplicador pode representar uma faixa específica representada por uma notação simples onde 0.. indica nenhum ou muitos, 0..1 indica nenhum ou um, e, qualquer podemos usar estas faixas para representar qualquer relação de multiplicação imaginável, 1..5, para representar um ou cinco, por exemplo.

As relações entre os casos de uso são definidas pelas associações de generalização, inclusão e extensão. A generalização, semelhante ao processo de generalização que estudamos para os atores, deve ser indicada por uma seta de linha contínua e ponta aberta, exatamente como na generalização de atores. Já a inclusão e extensão, respectivamente, *include* e *extends*, são representadas por setas tracejadas de ponta cheia sob um símbolo para indicar a característica de inclusão ou extensão.

A inclusão ocorre quando parte das funcionalidades do caso de uso apresentam alguma possibilidade de uso por outros casos de uso. Neste cenário, as boas práticas indicam que o melhor procedimento é dividir este caso de uso, colocando estas funcionalidades em outro caso de uso. Feito isso, o novo caso de uso deve ser incluído no caso de uso original. Este novo caso de uso, aquele que representa as funcionalidades que podem ser usadas por outros casos, fica disponível para associação com qualquer outro caso de uso. A seta que representa a inclusão deve apontar para o caso de uso que foi criado. Como pode ser visto no Exemplo 1.

Exemplo 1 – Associação de casos de uso por inclusão

Você está diagramando o caso de uso de compra via internet e percebe que a funcionalidade de listar itens pode ser utilizada em outras partes do

sistema então você fica com dois casos de uso: comprar e ver itens. Como pode ser visto na Figura 12.

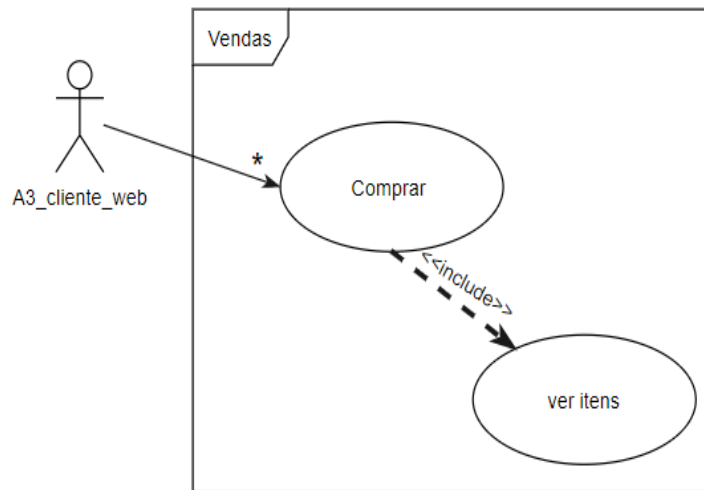


Figura 12 - Exemplo de associação por inclusão no diagrama de casos de uso

No Exemplo 1, Figura 12, tomei a liberdade de criar um ator concreto, usando o padrão de letras minúsculas que comentara antes. Se a leitora adotar este padrão, apenas olhando para a Figura 12 é possível perceber que este sistema tem outros atores. Pelo menos um ator abstrato. Mesmo que não estejamos vendo este ator abstrato no diagrama da Figura 12 e esta é a magia da modelagem.

A outra associação que pode ser realizada entre casos de uso é a extensão. Esta associação, a extensão, ocorre quando podemos criar um caso de uso que será utilizado para descrever o comportamento de um outro caso de uso.

Durante a realização do caso de uso principal, o uso do caso de uso de extensão não é obrigatório, para atingir o valor desejado podemos, ou não, usar a extensão representada pelo caso de uso de extensão. A representação da extensão deve ser feita por uma seta tracejada com o símbolo `<<extends>>`, apontando para o caso de uso que recebe a extensão. Como pode ser visto no Exemplo 2.

Exemplo 2 – Associação de casos de uso por extensão.

Ainda diagramando o processo de venda online, durante modelagem do processo de login você percebe que este caso pode

estender o caso alterar senha e a Figura 13 apresenta um fragmento deste Diagrama de Casos de Uso.

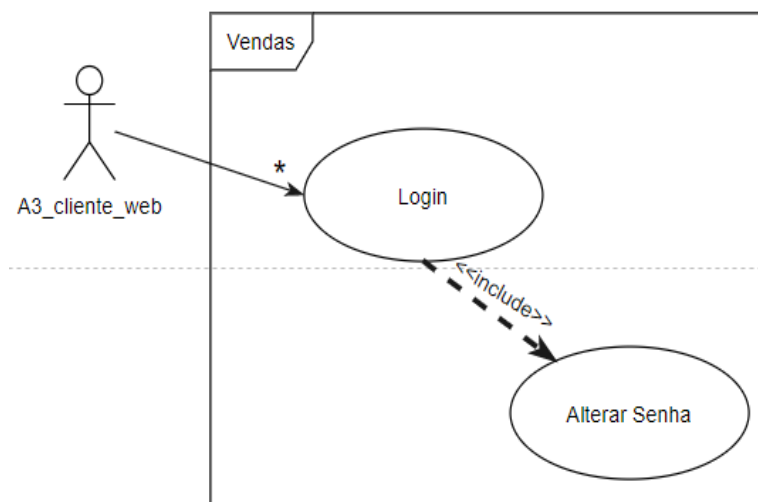


Figura 13 - Exemplo de relação de extensão entre casos de uso.

A última relação entre casos de uso é a generalização. Assim como acontece em relação aos atores é possível descrever um caso de uso abstrato e, a partir deste criar um conjunto de casos de uso concreto.

No caso da generalização, o caso de uso concreto irá herdar as funcionalidades do caso de uso abstrato. Esta associação é rara por causa da sua complexidade.

Um caso de uso diz respeito a uma funcionalidade do sistema, uma funcionalidade que gere valor. Quando há uma relação de herança, significa que o caso concreto herda esta funcionalidade, mas inclui alguma característica extra, ou diferenciada. Quando falamos de objetos, a herança é clara. O mesmo quando nos referimos a atores. Em relação a casos de uso, isso não é muito intuitivo. Talvez o Exemplo 3 ajude.

Exemplo 3 – Associação de casos de uso por generalização.

*O caso **fazer nova oferta** relativo a um cliente novo, é ligeiramente diferente. Se o cliente for novo o caso **fazer nova oferta** precisa incluir o caso **criar cliente**. Então, o caso **fazer nova oferta***

deve herdar as características do caso **fazer oferta**. Como pode ser visto na Figura 14.

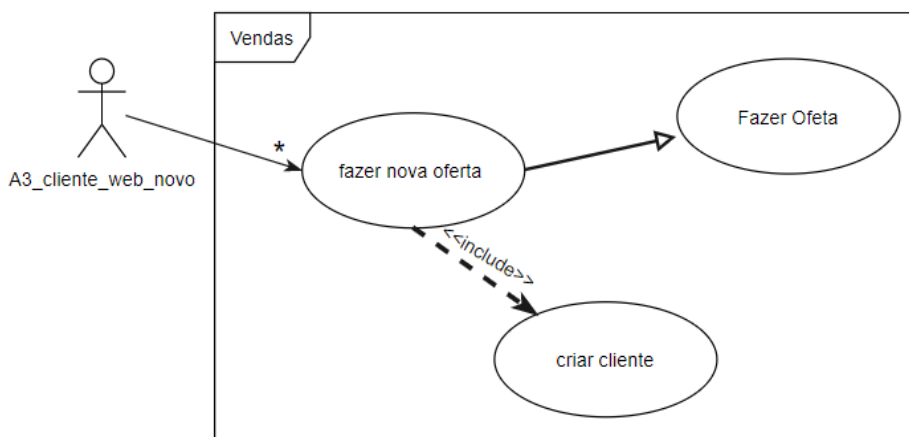


Figura 14 - Associação de casos de uso por generalização.

3.2.2 Documentação de casos de uso

A documentação depende o tamanho do projeto e do tipo de gestão que está sendo usado. Se for uma gestão ágil, a documentação pode ser bem simples.

Identifique os casos de uso: antes de começar, identifique os casos de uso. Esta identificação deve ser simples se a análise de requisitos foi corretamente realizada. Se não, você terá que rever a análise de requisitos ou criar os casos de uso. Nos dois casos, já agrupe os casos de uso por similaridade ou proximidade. Um caso de uso chamado *autenticação por e-mail* deve ficar perto de um caso de uso chamado *login*.

Identifique os atores: os atores devem estar explicitados na análise de requisitos. Se não, você terá que fazer isso agora. Este é o momento em que você deve dividir os atores em grupos em busca de possibilidades de generalização.

Identifique as relações entre casos de uso: cada caso de uso deve ser considerado na busca de relações com outros casos de uso. Todas as vezes que você encontrar uma relação. Explicita esta relação.

O documento de registro do caso de uso deve conter, no mínimo, os atores, os casos de uso e as relações entre eles. A documentação serve de apoio a criação do Diagrama de Casos de uso. Então deve ser mais explícita. A leitora deverá

escrever o valor esperado do caso de uso, o que ele deve fazer, a interação com ator e as possíveis relações de inclusão e extensão com outros casos de uso. Escrever um caso de uso é tanto uma técnica quanto uma arte.

3.3 Escrevendo casos de uso efetivos

O conceito de casos de uso foi definido em 1986, por Ivar Jacobson, mas foi **Alistair Cockburn**, em seu livro *Writing Effective Use Cases*⁴ quem popularizou o uso desta ferramenta. Efetivo é aquilo que produz resultados, aquilo que é definitivo. O objetivo de Cockburn ao escrever seu livro foi melhorar a qualidade dos produtos de software por meio do entendimento dos requisitos. Assim, em seu livro ele destaca uma sequência de cinco níveis de precisão que devem ser seguidas durante o processo de escrita destes casos e você deve começar da precisão mais baixa:

Nível de precisão 0: definição do escopo;

Nível de precisão 1: identifique o ator primário e seu objetivo;

Nível de precisão 2: descreva o caso de sucesso principal desta interação;

Nível de precisão 3: analise as possíveis extensões deste caso;

Nível de precisão 4: descreva os passos para o tratamento das extensões.

Não existe nenhuma obrigatoriedade na UML com relação ao processo de escrita de casos de uso proposto por Cockburn, ou qualquer outro. A leitora pode optar por usar este processo, ou qualquer outro que achar mais adequado a sua prática. Este pobre autor, descreve o processo de Cockburn apenas por acreditar se tratar de uma boa prática para quem está aprendendo análise de sistemas.

Durante a redação dos casos de uso responderemos a um conjunto de perguntas e, todas as vezes que conseguirmos responder uma destas perguntas progrediremos na redação dos casos de uso e aumentaremos a precisão com que

⁴ Em tradução livre: Escrevendo casos de uso efetivos.

nossos casos são descritos. Cada uma destas perguntas, permitirá que a leitora reveja seu trabalho e inclua novos detalhes na criação de casos de uso. Observe, contudo, que este processo está intrinsicamente ligado a engenharia de requisitos. Ou seja, a leitora terá que rever seu processo de elicitação de requisitos para adequá-lo a este sistema de redação de casos de uso. A primeira pergunta que precisa ser respondida é: qual é o escopo?

3.3.1 Qual é o Escopo?

Precisamos responder esta pergunta entendendo qual é o objetivo geral do sistema que será desenvolvido, determinar quais são os objetivos que precisam ser satisfeitos para atingir o objetivo geral do sistema, identificar um objetivo principal para cada interação com o sistema e identificá-lo com uma sentença verbal, imperativa. Logo em seguida, devem ser determinados os níveis de cada ator envolvido (primário, secundário) e suas características. Esta informação precisa ser validada com os *stakeholders*.

Na prática, esta deve ser um processo ágil, interativo e incremental. Você irá precisar de um, e somente um, caso de uso bem documentado relativo a cada objetivo principal do sistema para dar continuidade ao processo.

Para determinar o objetivo geral do sistema, precisamos saber qual o problema que ele irá resolver, como se espera que o sistema resolva este problema e, principalmente, por que esta forma de solução é adequada. Você pode recolher estas informações durante o processo de elicitação de requisitos, mas aqui, na definição dos casos de uso, esta informação deve estar claramente explicitada em um parágrafo, como apresentado no Exemplo 4.

Exemplo 4 – descrição do objetivo principal de um sistema de venda de fraldas.

*A empresa **FraldasVerdes**, uma fabricante de fraldas ecológicas precisa que todas as suas vendas sejam controladas pelo sistema **FraldasVerdes Vende**, um sistema de vendas ao consumidor final,*

online, móvel e acessível. Este sistema irá permitir que o usuário final da fralda ecológica entenda o processo de fabricação e possa fazer seus pedidos sem a necessidade de sair de casa.

No Exemplo 4 temos a empresa, seu objetivo, a explicitação do ator principal e os objetivos do sistema e do ator principal. Uma vez que este objetivo principal tenha sido determinado precisamos listar os casos de uso que irão permitir que o sistema atinja os objetivos determinados. Neste momento precisaremos identificar os atores e seus casos de uso.

Vimos anteriormente que atores são funções realizadas por pessoas, ou dispositivos, que interagem com o sistema. No caso das **FraldasVerdes**, teremos pelo menos um ator, que poderemos chamar de *cliente*. Esta é a função das pessoas que vão acessar o sistema **FraldasVerdes Vende** para comprar o produto do nosso cliente, fraldas. Este é o ator que tem o objetivo de comprar fraldas e podemos criar o caso de uso usando uma frase verbal: *Comprar Fraldas*.

Para cada objetivo de um ator existem dois atributos importantes, o nível do objetivo e sua categoria, ou tipo. Vamos classificar os objetivos em três níveis: sistema, interno e contexto.

Os objetivos, representados por casos de uso, em **nível de sistema**, são aqueles em que podem ser satisfeitos por uma interação direta com o sistema, como por exemplo, *imprimir o Extrato*, *Procurar Fraldas*, ou *Autorizar Pagamento*. Os casos de uso que fazem parte do sistema, mas não estão diretamente relacionados com o objetivo principal, são pequenos objetivos que são usados para construir, o processo, ou para informar o sistema são classificados como **Internos**. Neste caso estão, por exemplo, *Logar* e *Esvaziar o Carrinho de Compras*.

Por fim, se tivermos objetivos que envolvam outros sistemas, vamos classificá-los com **contextuais**. Nesta classe estão, por exemplo: *Autorizar a Compra no Cartão* e *Buscar Endereço pelo CEP*.

Em resumo se o caso de uso diz respeito a uma ação direta para um objetivo com valor definido, classificamos como **caso de uso de sistema**. Se a interação é

incremental para atingir o objetivo, classificamos como **caso de uso interno** e, por fim, se a interação é entre sistemas classificamos com **caso de uso contextual**.

Quanto aos tipos, neste momento, devemos simplificar e classificar os casos de uso em apenas três tipos: sistema, sumário e abuso.

Os **casos do tipo sistema**, complementam os casos de uso do nível sistema. Estão neste tipo os casos de uso que incluem uma interação direta com o sistema e um objetivo. Os **casos de uso do tipo sumário** são dizem respeito a descrição de um objetivo que contém um conjunto de outros objetivos. Podemos utilizar estes casos do tipo sumário para organizar um conjunto de outros casos, focando em um objetivo principal para o conjunto e mantendo uma ordem coerente para os casos de uso internos deste objetivo. Muitas vezes a descrição destes casos de uso do tipo sumário começa com a palavra gerenciar. Por exemplo: *Gerenciar Contas de Usuário* que pode conter os casos *Criar Conta de Usuário*, *Modificar Conta de Usuário* e *Remover Conta de Usuário*.

Os **casos de uso do tipo abuso**, estão relacionados com os objetivos que não queremos no nosso sistema. Estes são os casos de uso que estarão relacionados com as interações de atores que podem prejudicar, de alguma forma, o funcionamento do sistema. Estes atores, devem ser listados e suas ações precisam ser contempladas. Neste tipo estão os casos de uso: *Roubar dados de Usuários* e *Comprar como outro Cliente*. Definir estes atores e estes casos de uso é uma boa prática de análise de sistemas.

De forma prática, em um ambiente de práticas ágeis, partimos de um caso de uso do tipo sumário e detalhamos todos os seus casos de uso, para encontrar os que são de sistema, internos e contextuais. Depois analisamos os casos de uso do tipo abuso.

3.3.2 Quem quer o quê?

Esta é a pergunta que nos levará ao **nível de precisão 1**. As respostas para esta pergunta são originadas tanto da análise de requisitos quanto da análise de *stakeholders*. O conhecimento dos *stakeholders*, seus objetivos e anseios permite

a identificação dos atores primários e formata os casos de uso de sistema tanto do tipo sumário quanto do tipo sistema. Ou seja, a partir do conhecimento dos *stakeholders* podemos definir os atores e sua interação com o sistema.

Na definição dos casos de uso precisarmos conhecer a condição do ator, antes da interação com o sistema, estas condições são chamadas de condições prévias. O sistema deve verificar estas condições antes do caso de uso ser iniciado. Em casos de uso de interação direta homem-máquina, estas condições prévias podem ser definidas no processo de interface e não precisam ser explicitadas no sistema, um bom exemplo é a condição prévia *uso do mouse*. Contudo, em casos de uso onde os atores são outros sistemas, estas condições prévias podem, ou não, ser motivo de verificação e validação, e, neste caso, o sistema precisa certificar as condições prévias antes de iniciar o caso de uso.

O conhecimento dos *stakeholders* também irá determinar o disparo, em inglês *trigger*, do caso de uso. Este disparo especifica a ação que deve ser executada para que o caso de uso seja iniciado. Pode ser um clique, a digitação de uma informação e até mesmo a ocorrência de um evento temporal como um dia, ou hora específicos.

3.3.3 Como será realizado?

Até agora respondemos perguntas analisando *stakeholders* e requisitos, pensamos em atores e sua interação com o sistema. Mesmo que estes atores representem funções que serão realizadas por outros sistemas, tudo o que fizemos foi determinar o porquê das interações e quais interações são necessárias. Agora precisamos nos preocupar um pouco com o como estas interações serão realizadas. Importante, este como não tem relação com desenvolvimento, ou design de interação, ou qualquer outra função diretamente relacionada com a construção do sistema. Ao responder esta pergunta, estaremos no **nível de precisão 2** e vamos começar a dar forma aos casos de uso. Desta vez, a leitora terá que interagir aumentando suas entrevistas e incluindo desenvolvedores e testadores.

Ao responder esta pergunta você precisa documentar o sucesso do caso de uso. Escolha, neste momento, o caso de uso mais importante, e documente o sucesso, documente os passos que precisam ser seguidos para que a interação do ator com o sistema produza o valor desejado. Esta documentação pode ser uma lista de passos na ordem em que eles precisam ser executados. Depois de documentar, valide sua concepção com os *stakeholders*, desenvolvedores e testadores. A leitora precisa lembrar que neste momento, não interessa o quanto ela conhece de desenvolvimento. Se esta não for a sua tarefa, consulte os desenvolvedores. Valide sua modelagem com eles. Mas, lembre-se, a validação não é uma atividade passiva. Você tem opinião e pode justificar suas escolhas e discutir as escolhas alheias. A forma como essa validação será conduzida terá um grande impacto na qualidade dos casos de uso.

O sucesso de um caso de uso ocorre quando a interação entre ator e sistema produz o valor desejado para o ator. Este é o cenário onde o mundo é rosa, o céu é azul e tudo dá certo. Não é raro que um caso de uso contenha vários cenários, mas independente do caso de uso, existe apenas um cenário principal, um cenário onde o valor é produzido e entregue ao ator. Este é o cenário que precisa ser descoberto, documentado e validado neste nível de precisão. Todos os outros cenários que possam existir em um determinado caso de uso serão documentados só não serão documentados neste nível de precisão. Neste momento estamos preocupados com o cenário de sucesso do seu caso de uso principal.

Na documentação a leitora precisará ser clara, positiva e simples. No momento, certo, veremos algumas dicas que podem ser usadas para a documentação de casos de uso. Lembre-se, se não conseguir explicar em um parágrafo, ainda não entendeu o problema.

3.3.4 O que mais pode acontecer?

Você respondeu à pergunta anterior e neste processo documentou o cenário de sucesso dos seus casos de uso. Uma vez que esta documentação tenha sido validada com *stakeholders*, desenvolvedores e testadores, podemos progredir para

o **nível de precisão 3**. A pergunta que responderá agora será a base do detalhamento da estória que estamos escrevendo para nosso caso de uso. Especificamente vamos olhar para coisas que podem acontecer e nos impedir de terminar este caso de uso, ou que precisam acontecer para garantir que este caso de uso termine. A estas condições, chamamos de extensões. A leitora atenta, vai lembrar que já falamos de extensões.

Para responder esta pergunta vamos considerar, o que acontece quando o caso de uso depende de algum processo de validação, cartão de crédito, por exemplo, e esta validação falha. Ou ainda, precisamos destacar áreas onde a eficiência do sistema é crítica para o sucesso do caso de uso. Para cada caso de uso, precisamos observar quais são os limites aceitáveis de eficiência. Estes limites de eficiência irão determinar a possibilidade ou não do caso de uso chegar ao sucesso. Na internet, por exemplo, um tempo de carga de 10 segundos seria considerado excelente nos anos 2000, no final dos anos 2010, uma página web com tempo de carga maior que 2s já seria penalizada pelos sites de busca. No começo dos anos 2020, todos estão buscando tempos de carga inferiores a 1s. Observe que um sistema perfeito para 2009 não seria aceitável em 2015.

Outro fator importante que precisa ser considerado neste momento são os caminhos, *paths* em inglês, alternativos para o sucesso do caso de uso. Este cenário ocorre quando o ator precisa escolher uma de várias opções durante a interação com o sistema. Estas opções devem ser documentadas como extensões. Lembre-se nós já determinamos qual era o caminho de sucesso para este caso de uso e documentamos o principal caminho para o sucesso. Todos os outros são complementares e devem ser documentados na forma de extensões.

O ator pode não se comportar da forma desejada e, neste caso, este é um cenário que também precisa ser considerado e, se for o caso, documentado. Por exemplo, o ator, cliente, das **FraldasVerdes**, pode *abandonar o carrinho* de compras. Ou pode clicar no lugar errado. Estes dois casos de uso representam extensões negativas para o caso de uso *comprar fraldas*. Contudo, são casos de

uso e precisamos analisar o que ocorre com o sistema nestes casos e se, e como, conseguimos reverter a situação negativa.

Por fim, precisamos considerar as condições de falha no sistema. Estes são cenários importantes que na maior parte dos casos de uso são negligenciados. Aqui, há uma crítica implícita as práticas ágeis. Nenhum dos casos de uso que analisei, durante a construção de sistemas desde 2005, utilizando práticas ágeis, considerou as condições de falha na forma de casos de uso ou de histórias de usuários. Existe nas práticas ágeis, uma noção implícita e pouco discutida, que segurança e continuidade do serviço são atribuições da equipe de desenvolvimento. Não são! São atribuições do analista de sistemas, cabe a este desenhar os casos de uso, ou histórias de usuários considerando possíveis falhas e indicando os cenários que devem ser obedecidos.

3.3.5 Como lidamos com os casos de uso?

Esta é a última pergunta que deve ser respondida, quando a leitora chegar neste ponto, estaremos no **nível de precisão 4**, o mais fino de todos. Aqui, vamos trabalhar a documentação das extensões. Cada extensão realiza uma função. Estas funções podem ser simples e complexas, mas são, com certeza, indispensáveis para a conclusão do caso de uso e, conseqüentemente, para a geração de valor. Lembre-se que verificamos a existência de extensões à medida que fomos aumentando a precisão dos nossos casos de uso. Agora precisamos rever cada uma e, se for o caso, removê-la ou documentá-la.

É neste momento que também precisamos olhar os casos de uso de inclusão. Quantos e quais dos seus casos de uso podem ser divididos em casos de uso mais específicos de forma que estes possam ser usados por outros casos de uso do sistema.

Em alguns casos de uso, vamos perceber que as interações que denominamos de extensão são, na verdade, casos de uso que não havíamos percebido, ou partes importantes de outros casos de uso que devem ser transformadas em inclusões. Em qualquer um destes cenários, cabe a você que

está fazendo a análise do sistema, considerar cada extensão, sua função, necessidade e interação dentro do caso de uso. Caso você se isente desta responsabilidade caberá ao desenvolvedor tomar as decisões necessárias para fazer o sistema funcionar.

Os estadunidenses têm um dito popular muito interessante: não existem almoços de graça. Com isso eles querem dizer que sempre há um preço que precisa ser pago, ainda que não o estejamos vendo. Em práticas ágeis, prega-se o uso de histórias de usuário. Basicamente um caso de uso descrito apenas com o cenário de maior sucesso. Esta prática diminui o tempo de análise e inicia o processo de desenvolvimento em um tempo mais curto. Isto é uma troca de precisão por velocidade no desenvolvimento. Ou seja, estamos trocando a possibilidade de ter o sistema imediatamente, pela possibilidade de termos falhas e erros neste sistema. Trata-se de um compromisso baseado na confiança. Confiança no *stakeholder* e deste na equipe de desenvolvimento. Se a leitora usar este processo de descrição cuidadosamente irá limitar os erros que o sistema terá, mas infelizmente irá demorar mais para colocar o sistema em produção.

4 ESTÓRIAS DE USUÁRIOS

As Estórias de usuários, *user stories* em inglês, são pequenos textos que descrevem uma funcionalidade do sistema do ponto de vista de um *stakeholder* e descrevem a forma como ele irá obter valor do sistema. Esta é uma ferramenta de elicitação de requisitos que tem o objetivo de substituir tanto a análise de requisitos quanto a diagramação dos casos de uso. Trata-se de uma prática incentivada tanto nas práticas conhecidas como *eXtreme Programming-XP* quanto no *Scrum*. Existe uma redação sugerida para a criação de histórias de usuário durante a entrevista com os *stakeholders*: *como um <insira o tipo de usuário>, eu quero < insira o objetivo > já que < insira a razão >*. O Exemplo 5 lista alguns tipos de histórias de usuário.

Exemplo 5 – Exemplos de histórias de usuários com a forma sugerida

1. *como um **aluno**, eu quero **ver minhas notas** já que **preciso saber se fui aprovado**.*
2. *como um **cliente**, eu quero **me autenticar no sistema** já que **gostaria de comprar fraldas***
3. *como um **paciente do sus**, eu quero **receber alta do hospital**, já que **meu tratamento acabou**.*

Em práticas ágeis de desenvolvimento de software não é raro que as histórias de usuários sejam escritas em *post-its* e coladas em um quadro, durante a entrevista com os *stakeholders* e a definição do projeto. São estas histórias que serão, mais tarde, ordenadas por prioridade para criar uma lista, chamada de *backlog* das tarefas que devem ser realizadas pela equipe de desenvolvimento. A prática de desenvolvimento ágil tem pequenas diferenças entre seus praticantes, mas todos concordam que as entregas, versões do sistema, devem ser realizadas em tempos curtos, inferiores a 6 semanas e devem contemplar um determinado conjunto de histórias. Este processo de entregas incrementais funciona como uma barreira as mudanças dos requisitos pelos *stakeholders*.

Observe, porque isso é muito importante. Nenhuma prática ágil lhe proíbe de fazer uma engenharia de requisitos detalhada e gerar todos os diagramas de caso de uso que desejar. Estas técnicas não são incentivadas porque são lentas e buscamos agilidade na entrega.

As boas práticas da criação de histórias de usuário determinam que todas as histórias devem estar relacionadas com um benefício que o *stakeholder* irá usufruir do sistema. Sendo assim, histórias de usuário que incluem a linguagem do sistema, a estrutura de distribuição ou qualquer outro requisito técnico não fazem parte do conjunto de histórias que iremos discutir com os *stakeholders* durante este processo ágil de análise de sistemas.

A frase padrão para a determinação das histórias de usuário, *como um <insira o tipo de usuário>, eu quero < insira o objetivo > já que < insira a razão >*, representa um bom começo, mas com o tempo a leitora vai perceber que faltam detalhes para que uma determinada história seja útil ao desenvolvedor. No Exemplo 5.2 dissemos: *como um **cliente**, eu quero **me autenticar no sistema** já que **gostaria de comprar fraldas***. Faltam informações importantes no momento do desenvolvimento. Se **autenticar** como? E-mail? Google? Facebook? Isso sem considerar a consequência: **gostaria de comprar fraldas**.

A definição dos detalhes deve ocorrer durante a reunião com os *stakeholders*. Quando a equipe de desenvolvimento acompanha esta reunião é possível incluir detalhes técnicos em cada cartão, ou *post-it*, que usamos para escrever as histórias. Ainda que estes detalhes não sejam parte da história servirão como base para o desenvolvimento. Uma boa história de usuário tem características específicas:

Independentes: cada história deve ser independente das outras histórias e da ordem em que serão construídas. As histórias dependentes de outras histórias, ou da ordem de construção adicionam níveis de complexidade desnecessários ao desenvolvimento de software.

Negociáveis: as histórias não são contratos nem detalhamento de requisitos. Tanto a equipe de desenvolvimento quanto os *stakeholders* devem se envolver na definição da história. Ela não deve ser muito detalhada justamente para não criar a sensação de que todos os detalhes são conhecidos.

Valorosa: as histórias devem agregar valor ao sistema que está sendo desenvolvido. Este valor deve ser percebido pela equipe de desenvolvimento e pelos *stakeholders*. Se a história não tiver um valor claro, será difícil determinar sua prioridade no backlog.

Estimável: todas as histórias devem ser estimadas em termos de tempo e recursos necessários para a sua conclusão. Esta estimativa requer tempo e experiência no processo de desenvolvimento. Precisa ser justa tanto para a equipe de desenvolvimento quanto para os *stakeholders*.

Uma história muito grande é difícil de estimar, difícil de priorizar e irá causar problemas no desenvolvimento por quebra de expectativas.

Tamanho apropriado: eventualmente as histórias precisarão ser divididas, ou reduzidas, para caber no tempo de desenvolvimento de uma determinada versão do software. Este é o momento em que serão negociados os requerimentos de cada história, a ordem de sua execução e a prioridade.

Testáveis: todas as histórias devem permitir que tanto os desenvolvedores quanto os stakeholders sejam capazes de testar o resultado do processo de desenvolvimento e determinar se o valor esperado foi atingido.

Quase consegui, a primeira letra de cada uma destas condições, em inglês forma a palavra *INVEST*, infelizmente este pobre escriba não encontrou um sinônimo em português para *sizable*. Estas características precisam ser observadas para cada história. Como não existe um contrato sobre os requerimentos e requisitos detalhados do sistema, se a histórias não forem explicitadas com estas características o projeto corre o risco de fracassar por quebra de expectativas.

Neste ponto talvez seja importante destacar que todas as práticas de desenvolvimento ágil ressaltam que histórias de usuários não são requerimentos. São, na verdade, referências para o desenvolvimento e para as novas reuniões com *stakeholders* para definir novos ciclos de desenvolvimento. As histórias de usuário também se a principal forma de decidir a prioridade de cada passo que será tomado durante o desenvolvimento do sistema.

4.1.1 A complexidade das histórias de usuário

Todo o processo de desenvolvimento ágil deve ser simples, direto, interativo e incremental. Contudo, existem momentos em que isso não é possível. Isto ocorre por culpa da complexidade do sistema ou por características específicas de desenvolvedores e *stakeholders* que dificultam o entendimento e a compreensão do processo para este cenário existem tipos especiais de histórias de usuários.

Estórias para documentação: mesmo que todos os processos ágeis se recusem a priorizar a documentação. Existem momentos em que esta documentação é inevitável. Como por exemplo, a documentação de uma determinada interface com outro sistema, ou algum manual de uso. Para estes momentos existem as estórias de documentação. Estas estórias são simples e devem ser escritas com a especificação da documentação para que entrem no backlog e possam ser priorizadas em um dos ciclos de desenvolvimento.

Estórias de falhas: são estórias que serão escritas para especificar a correção de falhas que escaparam do processo de teste e validação. Neste caso, a estimativa do esforço para a correção pode ser difícil, em alguns casos, encontrar o problema que está causando a falha em um sistema pode requerer um esforço muito grande, em outros este esforço será realizado para corrigir a falha. Estas falhas, não são apenas erros no código, podem ser erros grosseiros de interpretação dos valores dos stakeholders.

Estórias Spike⁵: são estórias de usuário com pouca ou nenhuma definição do que precisa ser feito. São estórias cuja estimativa de recursos e tempo é completamente imprecisa. Ou são estórias criadas para estimar outras estórias. Este nome tem origem no desenvolvimento de pequenas soluções para validar um conceito nas práticas de eXtreme Programming. Os desenvolvedores chamam estas soluções de spike solutions. Termo importado do esporte voleibol. Um spike é uma bola rápida, cortada diretamente para a quadra do time adversário.

Observe, amável leitora, por favor, que estas estórias são consideradas exceções as regras de criação de estórias de usuário. E estas regras são fundamentais para o bom funcionamento das práticas ágeis.

A prática, por outro lado, parece indicar que elas são comuns em todos os desenvolvimentos. Pelo menos, é essa a sensação que o humilde escritor tem quando pensa nas dezenas de projetos que acompanhou nos últimos anos. Todos

⁵ Em inglês prego, ou ponta afiada, um significado mais adequado para esta palavra na análise de sistemas.

com práticas ágeis e todos com histórias de documentação, falhas e *spike*. Em práticas ágeis, ou não, os problemas precisam ser resolvidos.

5 EXERCÍCIOS PRÁTICOS

Exercício 1 – Máquina de venda

Crie um diagrama de casos de uso para uma máquina automática de venda que vende apenas refrigerantes e salgadinhos. Esta máquina aceita cartão e dinheiro em notas. Tente usar as associações de extensão, inclusão e generalização e lembre-se que esta máquina irá precisar de manutenção de tempos em tempos. Esta máquina é diferente das máquinas que vemos todos os dias. Os produtos não estão visíveis para o cliente. O cliente precisa escolher o produto em uma interface do sistema.

Análise:

Detalhamento do domínio: a especificação do problema não apresenta nenhuma informação sobre a máquina estar, ou não conectada a um sistema central de controle. Por simplicidade vamos considerar que não. Então a máquina deve resolver todos os seus problemas de forma independente. A exceção será feita para o técnico, a máquina precisa sofrer manutenção, vamos considerar apenas manutenção corretiva.

Atores:

A1_cliente: ator que irá comprar um dos produtos ofertados pela máquina o sucesso desta relação será determinado pela entrega do produto. É preciso considerar que a máquina pode dar defeito e, neste caso, o ator deve receber o dinheiro de volta e, em casos extremos, chamar o técnico.

A2_técnico: ator responsável pela manutenção. Este ator será responsável por recarregar a máquina e por concertar eventuais defeitos. Como no domínio determinamos que a existência de só e somente só manutenção corretiva, este ator não terá acionamentos temporais.

Casos de uso:

UC_1: escolher o produto: o cliente escolhe o produto que pode ser um refrigerante ou um salgadinho e digita o código deste produto.

UC_2: pagar o produto: o cliente escolhe o meio de pagamento, dinheiro ou cartão e paga pelo produto.

UC_3: o cliente confirma a compra.

UC_4: receber o produto: o cliente recebe o produto e encerra a compra.

UC_5: o cliente cancela a compra, antes de receber o produto, durante a confirmação, o cliente cancela a compra.

UC_6: o cliente não recebe o produto por um problema da máquina e chama o técnico.

UC_7: o técnico concerta a máquina. Sempre que for chamado por um cliente.

UC_8: o técnico recarrega a máquina.

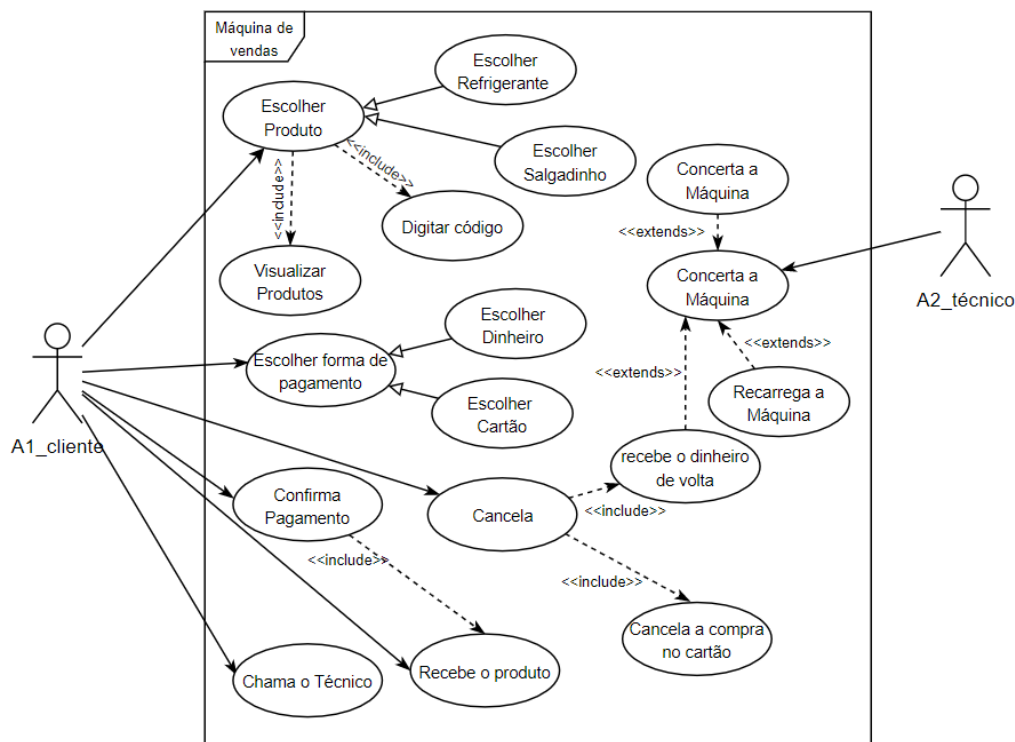
Diagrama de Casos de uso:

Figura 15 - Diagrama de Casos de uso do Exercício Máquina de venda.

6 REFERÊNCIAS

BAUSOLA, D. **Activity Diagram**. **zeroinfluence**, 2012. Disponível em: <http://zeroinfluence.wordpress.com/uml>. Acesso em: 04 Ago. 2020.

BECK, Kent et. Al. **Manifesto para Desenvolvimento Ágil de Software**. 2001. Disponível em: <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 10 ago 2020.

BECK, Kent e ANDRES, Cynthia. **Extreme Programming Explained**, 2012. Boston, MA. USA. Addison Wesleyt. 2º Edição.

DENNIS, A.; WIXOM, B. H.; ROTH, R. M. **System Analysis and Design**. Danvers, MA. USA: John Wiley & Sons, Inc., 2009.

OBJECT MANAGEMENT GROUP. **Unified Modeling Language**. Object Management Group, 2017. Disponível em: Acesso em: 04 ago. 2020.

SOMMERVILLE, I. **Engenharia de software**. São Paulo, SP. Brasil: Pearson , 2012.

UNHELKAR, B. **Software Engineering with UML**. Boca Raton, FL. USA: Taylor & Francis Group, LLC, 2018.

WIKIMEDIA COMMONS, THE FREE MEDIA REPOSITORY. File:ClassDiagramInitial.jpg. Wikipedia, 2016. Disponível em: <https://commons.wikimedia.org/w/index.php?title=File:ClassDiagramInitial.jpg&oldid=407262714>. Acesso em: 04 ago 2020.

WIKIPEDIA CONTRIBUTORS. Ivar Jacobson. Wikipedia, 2020. Disponível em: https://en.wikipedia.org/w/index.php?title=Ivar_Jacobson&oldid=944058726. Acesso em: 4 ago. 2020.