FIAP ON

DEVELOPMENT ENVIRONMENT

PRIMEIROS PASSOS NA CONSTRUÇÃO DE UM SOFTWARE

CRISTINA BECKER & ELISA SUEMASU

04

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 PRIMEIROS PASSOS NA CONSTRUÇÃO DE UM *SOFTWARE*

1.1 Introdução

Quando vamos desenvolver um *software*, precisamos saber o “passo a passo” que deve ser seguido para que isso aconteça. Neste capítulo, descreveremos o processo para a construção de um *software*, quais são as ações realizadas em cada etapa, quais artefatos (produtos) são construídos e quem são os profissionais envolvidos em cada passo.

Para abordar os conceitos relacionados ao processo de desenvolvimento de *software,* é necessário entender alguns conceitos básicos, como: informação, processo e sistema de informação, conforme descritos a seguir.

1.2 Conceitos

1.2.1 Definição de informação

Informação é um conjunto de dados ou um dado, desde que seja contextualizado em uma determinada realidade.

Por exemplo, um número de matrícula: 18307816115 – a princípio, essa informação parece ser simplesmente um dado, mas, na realidade, é um contexto relacionado a uma informação de matrícula de um aluno em uma unidade de ensino. Nesse exemplo: Unidade de Ensino: 183; Curso: 078; Ano de ingresso: 2016; Semestre: 1; e 15 representa o número de classificação de um vestibular. Portanto, um aparente dado, nesse caso, é uma informação conforme sua estrutura e composição. (Obs.: O exemplo de matrícula é apenas ilustrativo.)

Figura 1 – Exemplo de um número de matrícula
Fonte: FIAP (2017)

Outro exemplo: podemos dizer que uma nota fiscal é uma informação, pois possui número, data, cliente e itens de produtos – nesse caso, um conjunto de dados, portanto, também é uma informação.

Temos dois exemplos de informação: um relacionado à contextualização de um dado, ou seja, o que significa e o que representa; e o outro relacionado a um conjunto de dados, que é o mais comum e facilmente identificável.

Por que é importante entender o conceito de informação?

Quando falamos do processo de desenvolvimento de *software,* a informação é nosso meio de estudo, pois as organizações executam as tarefas com base em um fluxo de **informação**, por meio do qual definem o objetivo da área de negócio onde atuam.

1.2.2 Definição de processo

Nesta seção, abordaremos o conceito de processo em dois momentos: quando falamos do processo de desenvolvimento de *software* e quando falamos de processo relacionado à rotina das organizações.

Podemos entender como processo um conjunto de ações, etapas ou tarefas sequenciadas para atingir um determinado objetivo.

Para Davenport (1994, p. 390): “Um processo seria uma ordenação específica das atividades de trabalho no tempo e no espaço, com um começo, um fim, inputs e outputs claramente identificados, enfim, uma estrutura para ação”.

Harrington (1993, p. 16) define processo como: “Um grupo de tarefas interligadas logicamente que utilizam os recursos da organização para gerar os resultados definidos, de forma a apoiar os seus objetivos”.

De acordo com Hammer e Champy (1994, p. 8), “Um processo de negócio é um grupo de atividades realizadas numa sequência lógica com o objetivo de produzir um bem ou um serviço que tem valor para um grupo específico de clientes”.

Por que é importante entender o conceito de processo? Porque são os processos que recebem, transformam e geram as informações que vamos manipular no *software* que será desenvolvido. Além disso, o *software* é construído com base no processo de negócio, que representa a rotina da organização, conforme o fluxo de informação da área de atuação.

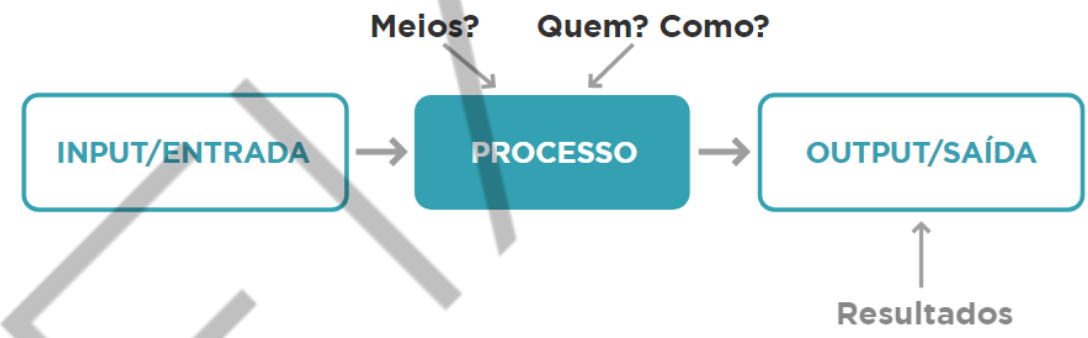

Figura 2 – Fluxo de processo
Fonte: Elaborado pelo autor, adaptado por FIAP (2017)

1.2.3 Definição de sistema de informação

A Sociedade Brasileira de Computação (SBC) define um sistema de informação como a “Combinação de recursos humanos e computacionais que inter-relacionam a coleta, o armazenamento, a recuperação, a distribuição e o uso de dados, visando à eficiência gerencial (planejamento, controle, comunicação e tomada de decisão) nas organizações. Pode também ajudar os gerentes e os usuários a analisar problemas, criar novos produtos e serviços e visualizar questões complexas”.

Podemos afirmar que o sistema de informação tem como objetivo processar e organizar dados para gerar informação como conhecimento.

Três atividades básicas produzem as necessidades de informação dentro de uma organização:

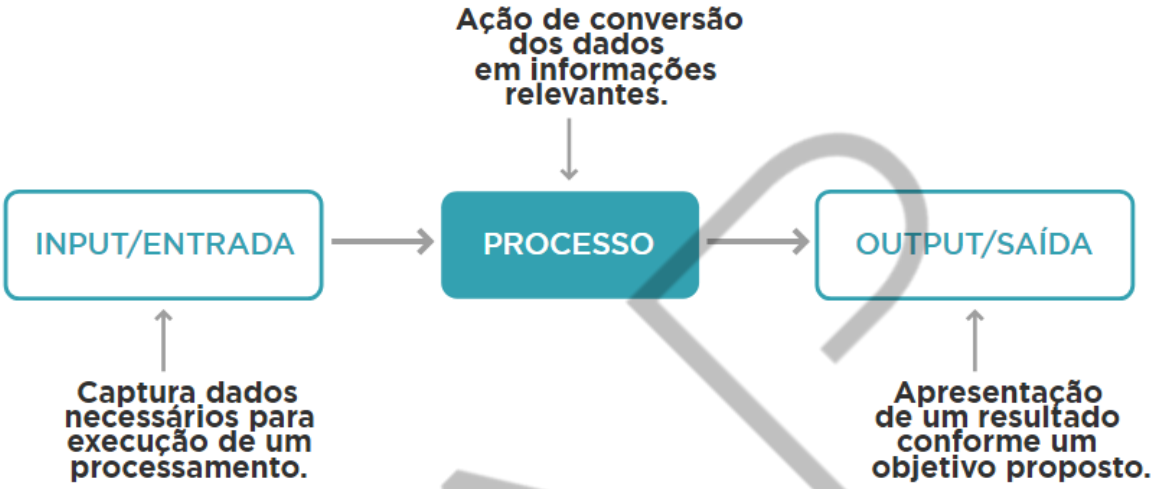

Figura 3 – Fluxo de sistema de informação
Fonte: Elaborado pelo autor, adaptado por FIAP (2017)

Os componentes do sistema de informação são:

- **Pessoas:** manipulam as informações dos processos de negócio por meio do uso dos sistemas computacionais.
- **Organizações/Procedimentos:** moldam os sistemas de informação conforme o fluxo de informação do seu processo de negócio.
- **Tecnologia:** representam os computadores e *softwares* que são as ferramentas de um sistema de informação.

A Figura “Sistemas de informação” traz um exemplo de sistema de informação e seus componentes.

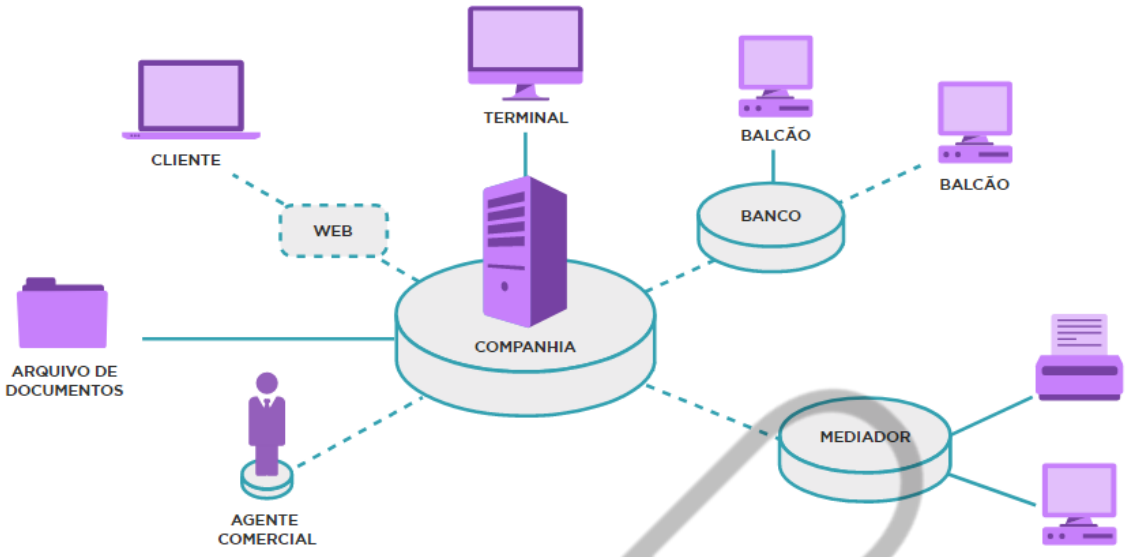

Figura 4 – Sistemas de informação
Fonte: Adaptado por FIAP (2017)

A classificação dos sistemas de informação baseados em TI de acordo com o tipo de informação processada é:

- **Sistemas de Informação Operacional:** tratam das transações rotineiras da organização (exemplo: caixa de supermercado).
- **Sistemas de Informação Gerencial**: agrupam e sintetizam os dados das operações para facilitar a tomada de decisão pelos gestores da organização (exemplo: supervisor dos caixas de um supermercado).
- **Sistemas de Informação Estratégicos**: integram e sintetizam dados de fontes internas e externas da empresa, utilizando complexas ferramentas de análise e comparação, simulação e outras facilidades para a tomada de decisão da cúpula estratégica da organização (exemplo: diretor).

Como mostrado na Figura “Tipos de sistemas de informação”, nas diversas áreas temos os tipos de sistemas de informação classificados conforme a hierarquia organizacional.

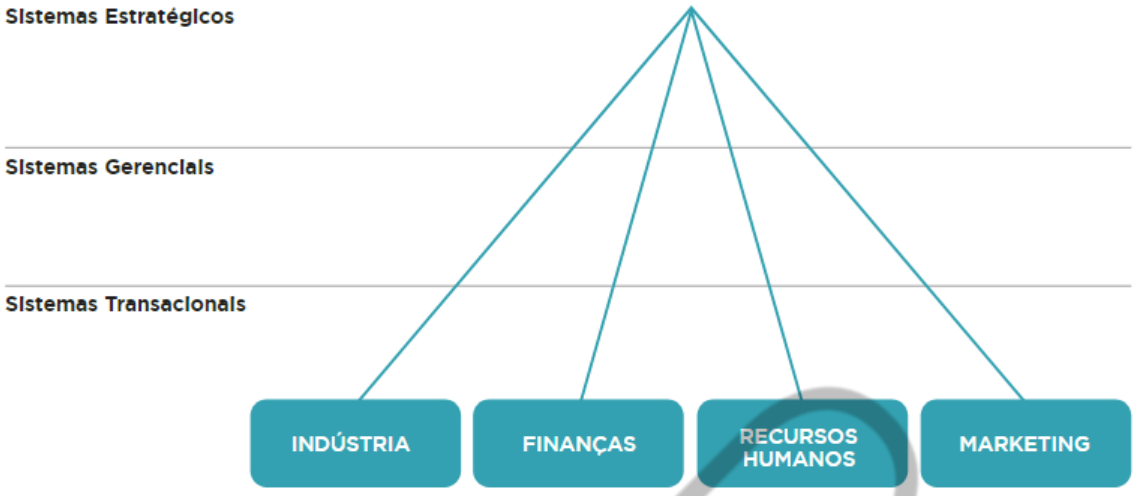

Figura 5 – Tipos de sistemas de informação
Fonte: Elaborado pelo autor, adaptado por FIAP (2017)

1.3 Processo de desenvolvimento de *software*

Sendo o *software* um dos componentes do sistema de informação, é muito importante que seu desenvolvimento tenha qualidade e atenda ao fluxo do processo de negócio conforme a necessidade e realidade das organizações. Para isso, é preciso seguir um processo de desenvolvimento de *software.* Nesta seção, vamos descrever suas etapas, artefatos e pessoas envolvidas.

No início da computação, poucos programadores seguiam algum tipo de metodologia, baseavam-se somente na própria experiência, criando o que chamamos hoje de Modelo Balbúrdia, sistemas desenvolvidos na informalidade, sem nenhum tipo de projeto ou documentação.

Nesse modelo de programação, o *software* tende a entrar num ciclo de duas fases apenas: implementação e implantação.

Os ajustes do *software* para atender aos novos requisitos sempre são em clima de urgência e de estresse, motivados por vários fatores, e algumas vezes por pressão política. Esses fatores implicam longas jornadas de trabalho e contribuem para o desgaste físico e psicológico dos profissionais envolvidos no projeto, desmotivando-os. Consequentemente, geram baixa qualidade e confiabilidade do *software* desenvolvido e também dificultam cada vez mais a sua manutenção.

O objetivo de um processo de desenvolvimento é definir quais atividades deverão ser executadas ao longo do projeto. Quando, como e quem as executará.

O processo de desenvolvimento de *software* deve seguir as etapas que definem desde o entendimento da necessidade do usuário até a implantação, como exemplifica a Figura “Etapas do processo de desenvolvimento de software”.

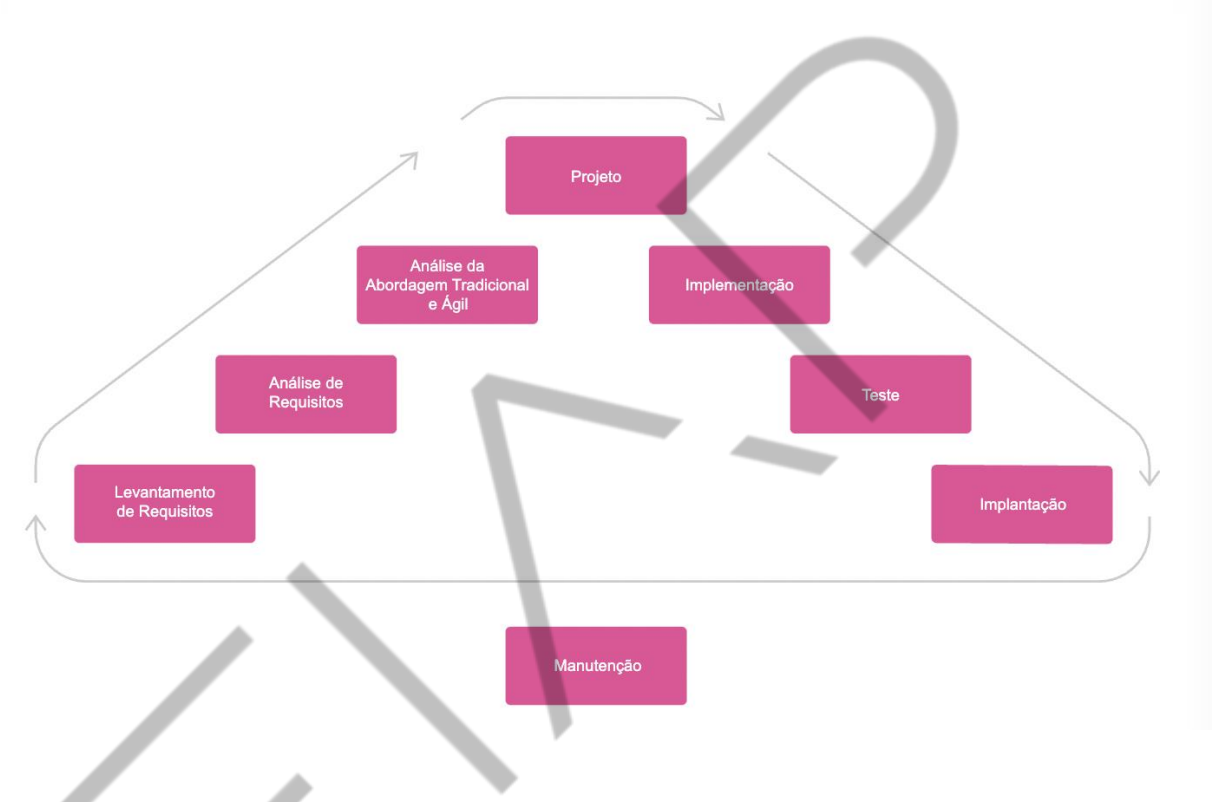

Figura 6 – Etapas do processo de desenvolvimento de software
Fonte: FIAP (2017)

Há vários processos de desenvolvimento propostos, mas é um consenso entre a comunidade de desenvolvimento que não existe um processo que seja o melhor, e sim o que melhor se aplica à situação de desenvolvimento (BEZERRA, 2014).

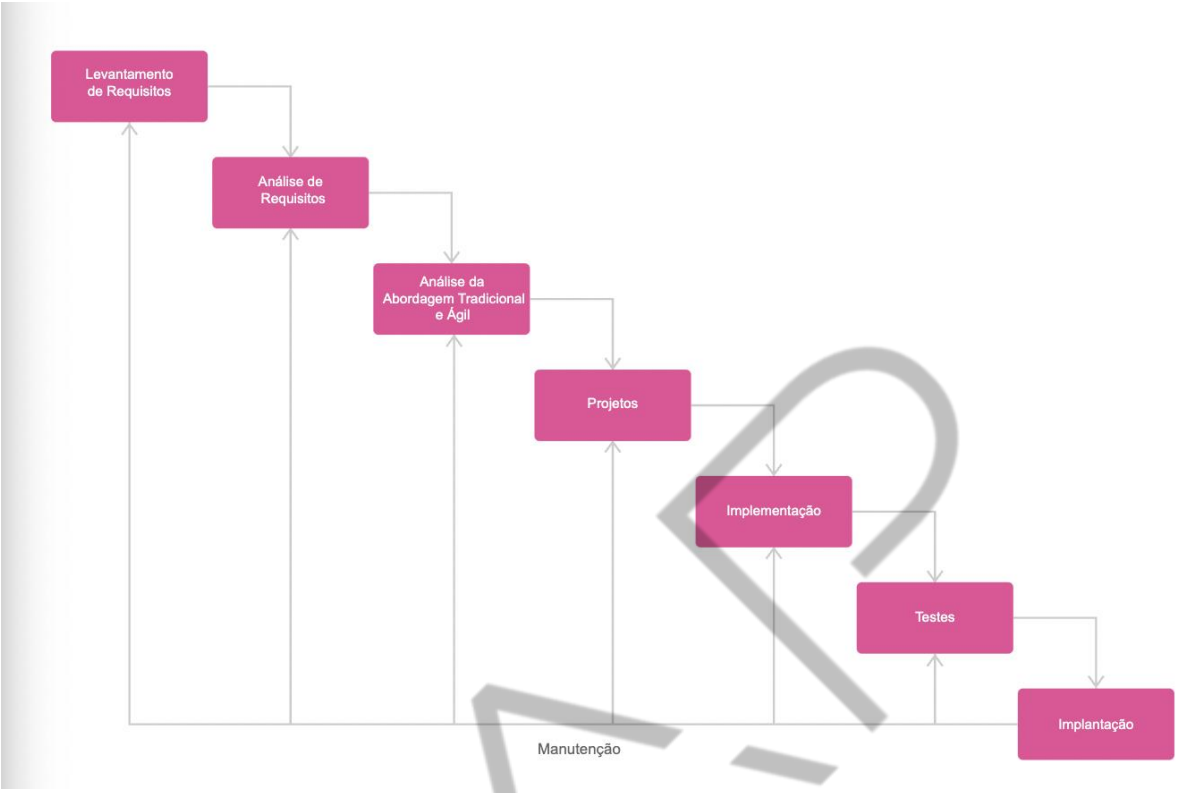

Figura 7 – Etapas do processo de desenvolvimento de *software* – modelo clássico
Fonte: FIAP (2017)

1.4 Atuação profissional

A utilização em grande escala das variadas metodologias de desenvolvimento ágil no mercado de trabalho atual introduziu o conceito de times multidisciplinares. Com isso, criou-se a necessidade de novos papéis e responsabilidades a serem desempenhados na organização, como, por exemplo, Scrum Master e Product Owner, implementados pelo *framework* Scrum. Vamos abordar, a seguir, cada um desses papéis e suas respectivas funções de forma resumida.

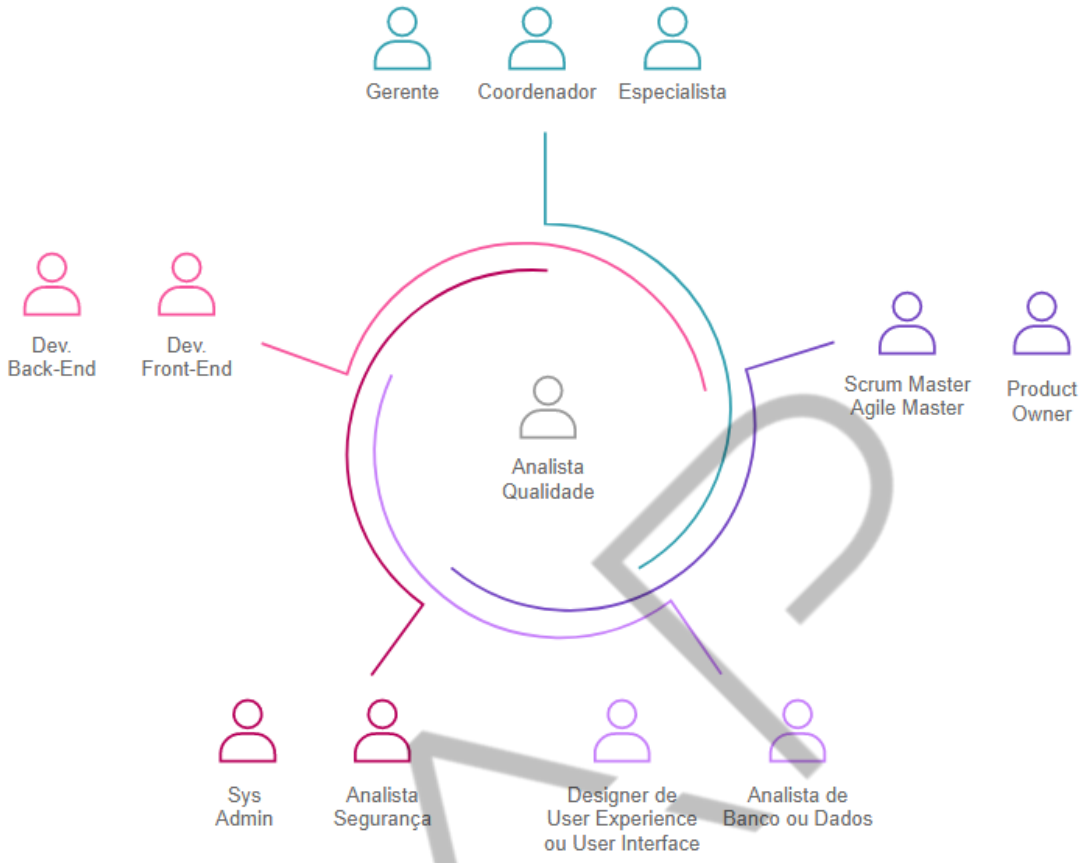

Figura 8 – Atuação profissional no mercado de trabalho
Fonte: Adaptado por FIAP (2019)

O time de desenvolvimento normalmente é composto de:

- **Desenvolvedor back-end:** profissional que trabalha na parte de “trás” da aplicação, na maioria das vezes é responsável pela codificação de grande parte das regras de negócio. Geralmente esse profissional atua pouco com a parte visual e domina linguagens de programação (exemplos: java, c#, go, php, python, ruby etc.) que dão suporte ao seu trabalho.
- **Desenvolvedor front-end:** ao contrário do desenvolvedor back-end, o desenvolvedor front-end trabalha na parte da “frente” da aplicação, desenvolvendo as telas que vão interagir com os usuários, preocupando-se com a experiência desse público. Esse profissional geralmente domina linguagens como JavaScript, Node, CSS, HTML, entre outras, para trabalhar.

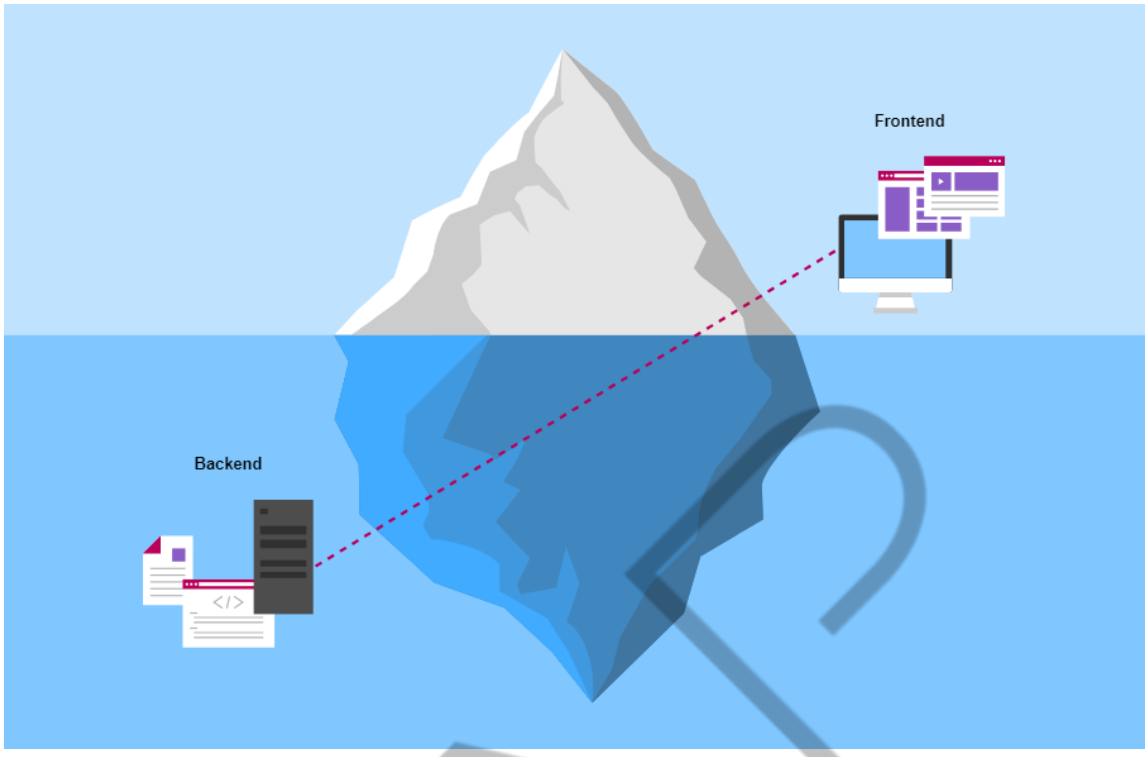

Figura 9 – Ilustração de back-end e front-end
Fonte: Adaptado por FIAP (2019)

- **Desenvolvedor full stack:** um profissional muito difícil de se encontrar no mercado, o desenvolvedor full stack é aquele que desempenha tanto a função de back-end como a de front-end. A dificuldade em se tornar full stack deve-se à grande quantidade de tecnologias que o desenvolvedor precisa dominar no âmbito back e front.

Figura 10 – Ilustração de desenvolvedor full stack
Fonte: Adaptado por FIAP (2019)

- **Analista de Qualidade (Quality Assurance – QA):** profissional responsável pela realização dos testes do software. É importante enfatizar que, nas metodologias ágeis, a qualidade é uma responsabilidade de todos, porém o QA faz o papel de guardião. O QA normalmente domina técnicas de teste unitário, aceitação, integração, carga, entre outros. Muitos QA também sabem desenvolver e acabam contribuindo para o desenvolvimento do software quando possível.
- **Scrum Master / Agile Master:** o Scrum Master é responsável pela cultura ágil dentro do time e por disseminar todas as práticas do *framework* Scrum. Ele atua como um líder do processo Scrum e um facilitador, contribuindo para a resolução de conflitos e remoção de impedimentos. O Agile Master exerce as mesmas atividades do Scrum Master, com a diferença que também domina outras metodologias de desenvolvimento ágil, como método Kanban ou XP.

Além do time de desenvolvimento, também temos:

- **Product Owner (Dono do Produto):** responsável por decidir quais recursos e funcionalidades o produto deve ter, e em que ordem de prioridade deve ser desenvolvido. Também mantém e comunica uma visão clara do que o Time Scrum está trabalhando no produto, sendo um ponto de intersecção entre a área de negócio e a área de desenvolvimento.

Os profissionais a seguir normalmente são compartilhados entre times, mas também podem fazer parte de um único time de desenvolvimento:

- **Designer de User Interface (UI) ou User Experience (UX):** o designer pode atuar com foco tanto na experiência do usuário, aplicando a melhor usabilidade e interação que o software terá, quanto na interface do software, construindo o design que represente a identidade visual da empresa e melhores padrões de design que o desenvolver front-end codificará.
- **Analista de Segurança:** com o objetivo de olhar para a segurança na organização, o Analista de Segurança é fundamental para promover a Cultura do DevSecOps e trazer o quanto antes a preocupação com segurança para o início do processo de trabalho, identificando, protegendo, detectando, respondendo e recuperando os pontos necessários sobre segurança.

Figura 11 – Ilustração de Analista de Segurança
Fonte: Adaptado por FIAP (2019)

- **SysAdmin (Administrador de Sistemas):** a função de SysAdmin é bem abrangente, mas geralmente são responsáveis por instalar, dar apoio e manter os servidores e sistema da organização junto ao time de desenvolvimento. Também é uma função-chave para a disseminação da cultura DevOps na empresa.
- **Analista de Dados (AD) / Administrador de Banco de Dados (DBA):** quando o assunto é banco de dados, existem dois perfis, o de DBA, responsável por gerenciar, instalar, configurar, atualizar e monitorar um banco de dados, e o AD, responsável por coletar, compilar, analisar e interpretar os dados do banco. O DBA acaba atuando mais no nível de hardware e software; e o AD, no nível de dados e negócio.
- **Especialista:** o especialista tem como função ser o ponto de referência por atuação, contribuindo para a formação dos profissionais e a resolução de demandas complexas ou que envolvam diversas áreas da organização. Para exemplificar, é possível ter um Especialista Back-End e outro Especialista Front-End, e ambos darem suporte aos demais desenvolvedores back-end e front-end. O mesmo serve para as demais funções de Especialista de Segurança, Qualidade, entre outros.

Na parte de gestão da empresa, geralmente temos:

- **Coordenador e Gerente:** responsáveis por coordenar e gerenciar as atividades das equipes de TI, avaliar e identificar soluções tecnológicas para otimizar os processos, planejar os projetos de implantação de sistemas e acompanhar as necessidades do negócio e dos clientes. Com relação às pessoas, realizar o acompanhamento e a evolução dos profissionais, manter a motivação, realizar feedbacks constantes e mentorias. Os papéis de Coordenador e Gerente possuem atividades similares. A principal ideia é que o Coordenador consiga escalar o trabalho do Gerente. Se um Gerente possuir 6 times, ele poderia ter 2 Coordenadores responsáveis por 3 times cada um, e os Coordenadores realizariam o *report* desses times para o Gerente. Dependendo da organização, o Gerente possui algumas funções a mais às quais o Coordenador pode não ter acesso, como planejamento

financeiro, aprovação de contratos, *report* para o *board* executivo, entre outras.

Figura 12 – Ilustração de Coordenador e Gerente
Fonte: Adaptado por FIAP (2019)

2 ETAPAS DO PROCESSO DE DESENVOLVIMENTO

2.1 Levantamento de requisitos

Chamada também de levantamento de dados, esta é a fase de levantamento de requisitos, na qual identificamos as necessidades e/ou problemas dos usuários. É a fase mais importante, que detecta a realidade do usuário, seus problemas e suas necessidades. Deve existir a clareza e a objetividade no processo de levantamento de dados, pois, caso algum entendimento seja diferente da real necessidade, isso pode acarretar maior prazo e custo ao projeto, além da falta de credibilidade na equipe de desenvolvimento e no *software*. As fontes de informações durante essa fase incluem documentação, usuários do sistema, *stakeholders* e especificações de sistemas, quando existentes.

A interação com os usuários do sistema e *stakeholders* ocorre por meio de entrevista, observações, *brainstorming* e *workshop*, podendo ser usados cenários e protótipos para auxiliar na obtenção de requisitos com base no processo de negócio e nas regras estabelecidas. As técnicas de levantamento de dados serão discutidas na próxima seção.

O levantamento de dados ou o levantamento de requisitos não é um momento único, é necessário demandar tempo para essa fase. Mas como os projetos de desenvolvimento sempre estão correndo contra o tempo, na prática, o importante é que a equipe de desenvolvimento entenda o processo e as regras de negócio do usuário/cliente.

Como descrevem algumas boas práticas, por exemplo: o XP – *Extreme Programming* define que o "*Business Value*" é o que precisa ser identificado inicialmente. Os analistas devem verificar o que pode agregar valor ao usuário/cliente no primeiro momento, o que vai garantir a entrega inicial de um módulo do *software* que satisfaça as necessidades iniciais do usuário.

Durante o levantamento de dados, devem ser registrados todos os fatos solicitados pelo cliente/usuário, para garantir que o entendimento seja mútuo.

É importante entender qual nível de usuário necessita do *software*, pois o levantamento de dados será realizado de acordo com o perfil do usuário.

Figura 13 – Perfis do usuário
Fonte: Adaptado por FIAP (2017)

Observando a Figura “Perfis do usuário”, podemos identificar três perfis de usuário:

- **Estratégico:** que visa à informação não estruturada, o entendimento do analista na visão de resultados. Lembre-se de que, para chegar a resultados, é necessário conhecer a origem da informação.
- **Tático:** a informação é semiestruturada e visa ao resultado entre os níveis operacional e estratégico.
- **Operacional:** conhece e executa as rotinas do processo de negócio.

É importante que, desde o início do levantamento de requisitos, sejam definidos os papéis dos envolvidos. Pois, na prática, o entendimento não é muito simples e, algumas vezes, o analista e os usuários ou *stakeholders* têm dificuldade de compreender a forma de pensar dos outros.

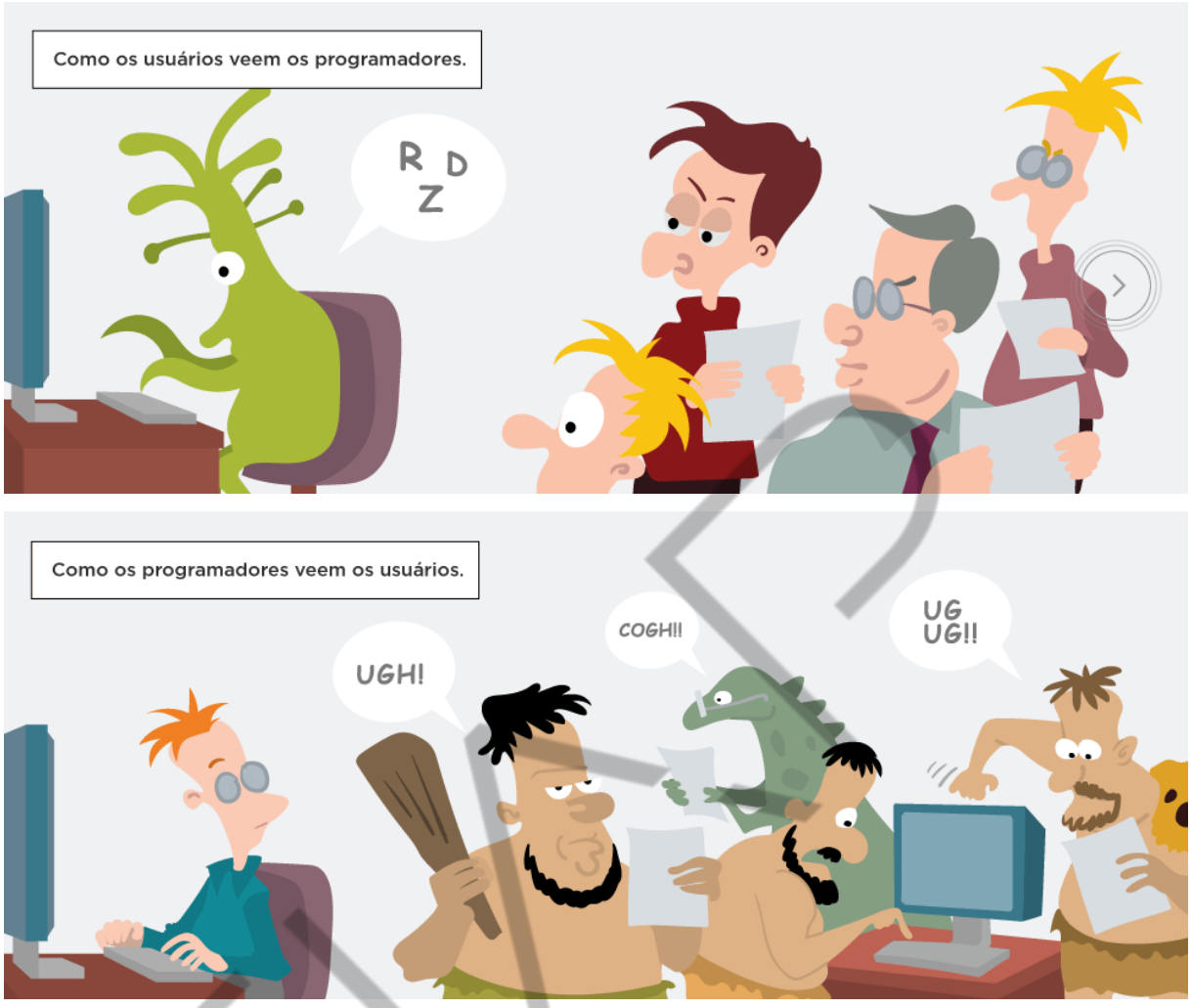

Figura 14 – Analistas e usuários
Fonte: FIAP (2017)

A falta de entendimento da necessidade do usuário pode resultar em objetivos completamente diferentes da necessidade real, conforme cenário clássico apresentado na Figura “Levantamento de dados” abaixo:

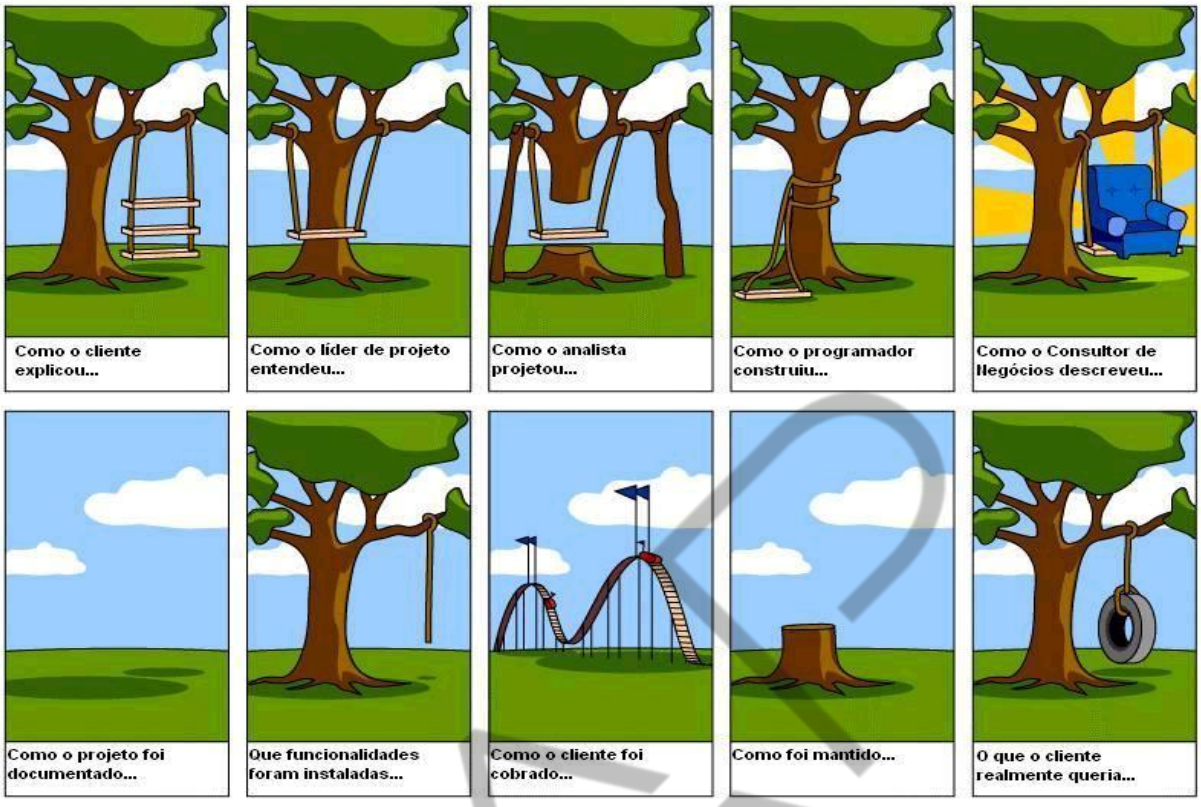

Figura 15 – Levantamento de dados
Fonte: Adaptado por FIAP (2018)

Desenvolver um *software* para atender às atividades operacionais exige o conhecimento do processo e das regras de negócio. É ideal que o usuário participe do processo de desenvolvimento de *software* em todas as suas etapas, como recomenda a boa prática do XP – Extreme Programming – e de outras metodologias ágeis, como Scrum, apesar de sabermos que este nem sempre é um cenário possível. O usuário é o que denominamos Key-User. Ele carrega todo o conhecimento sobre o processo e as regras de negócio e desenvolve as atividades operacionais.

Pensando em desenvolvimento de *software*, é necessário gerar alguns artefatos que contribuem para o entendimento e os esclarecimentos do processo de negócio. Alguns artefatos essenciais dessa fase são: especificação de processo de negócio, regras de negócio e requisitos que definem as funcionalidades do sistema.

Precisamos compreender o funcionamento da organização em relação às suas necessidades de informação e seus processos de negócio, para que possamos conceber, construir e entregar um *software* que satisfaça as reais necessidades dos usuários.

Figura 16 – Fluxo de processo
Fonte: Shutterstock (2017)

A especificação de processo de negócio é coletada com participação central do usuário, pois é ele que consegue definir as atividades que executa, suas necessidades e problemas. Essa coleta normalmente é realizada em processos de levantamento de dados liderados por analistas de negócios ou Product Owners. Além da colaboração com usuários, podem surgir, a partir de documentos e mapas de fluxo de processos de negócio, definições sobre a concepção e a rotina do processo. Em muitos casos, esses documentos não existem ou estão desatualizados, e acaba sendo necessário criar esse fluxo de processos a quatro mãos. Assim, o usuário descreve sua rotina e o analista interpreta e valida se está de acordo com o seu entendimento. Nesse momento, pode ser utilizado, por exemplo, o diagrama de atividades da **Linguagem de Modelagem Unificada (UML)** na modelagem do processo de negócio, o que garante o entendimento por meio de uma notação visual.

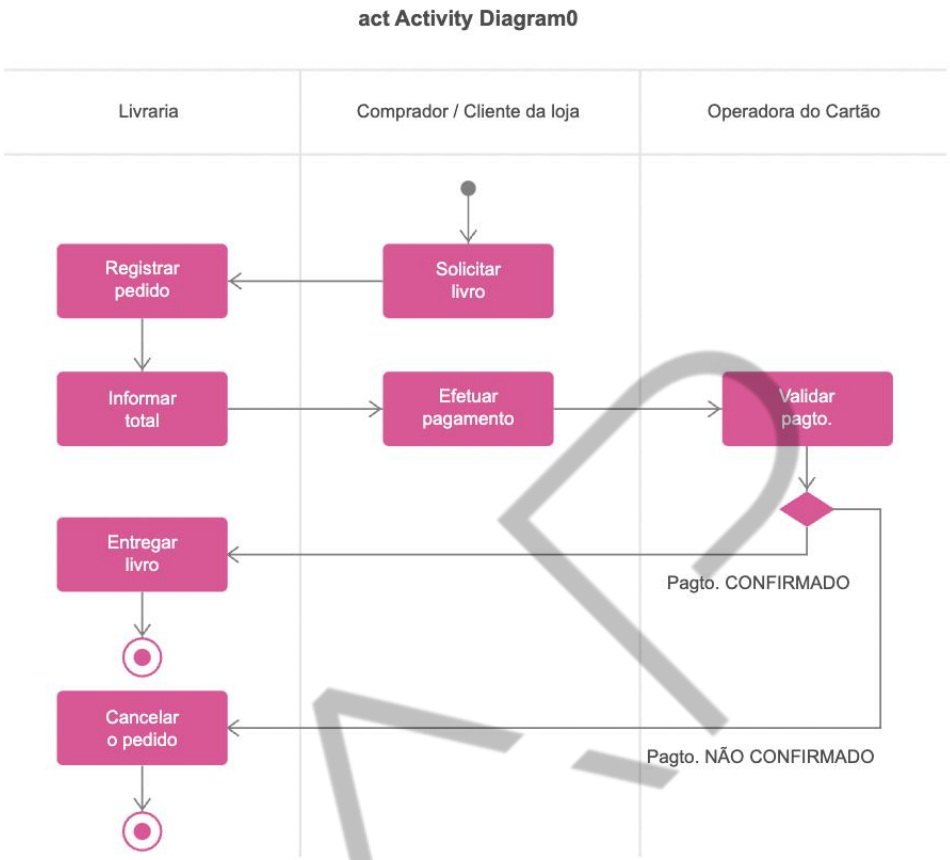

Figura 16 – Diagrama de atividades do processo de negócio
Fonte: Elaborado pelo autor (2017)

Existem algumas possibilidades na representação do processo de negócio, que pode ser desenvolvido pela notação visual, o que o torna muito mais simples, mas também pode ser descritivo: *o cliente solicita o livro, o funcionário da livraria registra o pedido e informa o valor. O cliente faz o pagamento e a operadora do cartão de crédito realiza a validação. Se o pagamento for confirmado, a livraria entrega o livro, caso contrário, o pedido é cancelado.*

Os **requisitos funcionais** representam as funcionalidades do sistema. É o resultado esperado para o usuário, o principal artefato da fase de levantamento de requisitos. Por esse fator, temos um capítulo que aborda somente tais conceitos.

Exemplo: quando você compra em um site de e-commerce, o sistema solicita que você informe o usuário e a senha para, em seguida, validá-los. Esse é um requisito funcional. Há outro exemplo de requisito funcional quando você informa os

dados do pagamento e o sistema registra o pagamento. Nesse momento, é importante entender que o requisito funcional tem origem no processo de negócio.

O conceito de “regras de negócio” começou a ser desenvolvido nos anos 1990, com a criação do Guide Business Rule Project, que posteriormente evoluiu para a criação do Business Rule Group. Sua declaração de princípio diz que “regras de negócio definem a estrutura e controlam a operação das empresas”.

As regras de negócio representam os procedimentos, as restrições, as normas, as cláusulas, os critérios, as políticas, ou seja, “como” executar o processo de negócio. De modo geral, representam a relação da informação para a execução do processo de negócio e determinam as condições para que os fatos sejam válidos.

As regras de negócio estão relacionadas aos requisitos funcionais em sua execução, quando falamos que o sistema deve registrar o pagamento, a regra de negócio relacionada ao pagamento que pode ser com cartão de crédito ou boleto bancário. As regras podem ser simples ou complexas, como, por exemplo, em casos de cálculos financeiros.

A notação para a definição de Regra de Negócio é RN, o que facilita e simplifica a leitura do documento do *software*, assim como a relação da regra de negócio ao requisito funcional, cuja notação é RF. Para exemplificar, podemos ter:

- Requisito Funcional:

RF01. Registrar Pagamento.

- Regras de Negócio:

RN01. Um Pagamento está relacionado a uma Pessoa Física.

RN02. Um Pagamento pode ser com cartão de crédito ou boleto bancário.

Ao relacionar as Regras de Negócio ao Requisito Funcional, temos:

RF01. Registrar Pagamento – RN01, RN02

As regras de negócio podem ser descritas em formatos diferentes, conforme quadro a seguir:

Nome	Quantidade de Inscrições Possíveis (RN01)
Descrição	Um aluno não pode ser inscrito em mais de seis disciplinas por semestre letivo.
Fonte	Coordenador da escola de informática.
Histórico	Data de identificação: 12/07/2015.

Quadro 1 – Notação para Regra de Negócio
Fonte: Elaborado pelo autor (2017)

Outros exemplos de Regras de Negócio:

- RN01. Uma venda tem 5% de desconto para clientes conveniados.
- RN02. Uma compra tem no mínimo um produto.
- RN03. O valor total de um pedido é igual à soma dos totais dos itens do pedido acrescido de 5% de taxa de entrega.

No levantamento de dados, quando o usuário definir seu processo, o analista deve fazer inúmeros questionamentos para entender como as informações estão estruturadas para a execução do processo de negócio. Não é muito simples quando o analista não está a par da área de negócio, por esse motivo, o diferencial do profissional de mercado é o conhecimento não somente em análise e desenvolvimento, que é área técnica, mas também o conhecimento de negócio.

As regras de negócio carregam as decisões do processo de negócio, que são tomadas de acordo com modelos e estratégias organizacionais. Por isso regras de negócio declaradas de maneira simples e objetiva, bem conectadas à lógica e à estratégia dos negócios e compreensíveis a todos os interessados, são de extrema relevância para o sucesso do desenvolvimento de *software*.

O Object Management Group (OMG), em seu documento “Semantics of Vocabulary and Business Process (SVBR)”, apresenta as seguintes definições (p. 150): “Uma regra é uma proposição que reivindica obrigação ou necessidade” e “uma regra de negócio é uma regra que está sob a jurisdição do negócio”.

Graham (2005), em *Business Rules Management and Service Oriented Architecture*, afirma: “Uma regra de negócio é uma declaração afirmativa compacta, atômica, bem formada de um aspecto do negócio, que pode ser expressada em termos diretamente relacionados ao negócio e seus colaboradores, usando linguagem

simples e não ambígua, que seja acessível a todas as partes interessadas: *business owner* (dono do negócio), *business analyst* (analista de negócio), *technical architect* (arquiteto técnico) e cliente, entre outros".

Uma das principais ferramentas de tecnologia disponibilizadas atualmente no mercado para apoiar a manutenção de regras de negócio são soluções de *Business Rules Management Systems* (BRMS), que, combinadas à adoção de um *Business Process Management System* (BPMS) e de uma arquitetura orientada a serviços (SOA), podem fornecer infraestrutura de negócio muito mais ágil às mudanças do mercado. Falaremos mais sobre BPMS nos próximos capítulos.

Um Sistema de Gerenciamento de Regras de Negócios (BRMS) permite que políticas organizacionais e as decisões operacionais associadas a essas políticas sejam implementadas, monitoradas e mantidas separadamente do código principal de um aplicativo. Exteriorizando regras de negócios e fornecendo ferramentas para gerenciá-las, um BRMS possibilita aos especialistas em negócios definir e manter as decisões que orientam o comportamento de sistemas, reduzindo a quantidade de tempo e esforço necessários para atualizar os sistemas de produção e aumentar a capacidade da organização de responder às mudanças no ambiente de negócios.

2.2 Análise

Nesta fase, ocorre o estudo e o entendimento sobre a lógica de funcionamento do sistema. Para tanto, a criação dos modelos é essencial como resultado dos estudos, por meio da utilização de ferramentas, técnicas e métodos.

De acordo com Blaha e Humbaugh (2005), a fase de análise pode ser subdividida em análise de domínio, ou análise de negócio, e análise da aplicação.

A análise de domínio modela os objetos do mundo real como uma venda que possui um cliente e o(s) produto(s), que são objetos de negócio, portanto de domínio. Já as telas necessárias para apresentá-los são consideradas objetos da aplicação – no exemplo, a própria tela em que se registra a venda. Com a orientação a objetos, a modelagem das classes de domínio define a aderência do sistema em relação à necessidade do usuário.

Muitos deixam essa fase para ir direto ao resultado, que é a implementação. Infelizmente, os resultados acabam se tornando negativos e divergentes do que foi definido na fase de levantamento de requisitos. Essa fase permite a criação de modelos que viabilizarão a estruturação lógica do *software*.

Artefatos construídos nessa fase são Diagramas da UML, entre eles, o diagrama e a documentação de casos de uso e o diagrama de classe de domínio.

A UML é uma linguagem visual para modelagem de sistemas orientados a objetos. Esse assunto será tratado nos próximos capítulos.

A validação e a verificação dos modelos construídos de acordo com o levantamento de requisitos realizado compõem o fator mais importante que deve ocorrer nessa fase, pois assegurará que as necessidades do usuário serão atendidas.

Nessa fase, ocorrem as modelagens do sistema e do banco de dados. Como usamos a orientação a objetos, na prática, sempre surge uma dúvida: começar pelo modelo de banco de dados ou pela modelagem das classes. Segundo o conceito da orientação a objetos, a princípio devemos desenvolver a modelagem das classes e, posteriormente, a do banco de dados.

As ferramentas *cases* permitem a construção dos modelos de banco de dados com a modelagem das classes, mas este é um caso que deve ser discutido muito com a sua equipe, pois os modelos de banco de dados podem não ser aderentes à realidade das regras de negócio. Nem sempre é recomendável, cada cenário deve ser muito bem discutido e estudado.

2.3 Projeto

Na fase de projeto, na teoria, há duas atividades: projeto de arquitetura, também chamado projeto de alto nível; e projeto detalhado, conhecido como projeto de baixo nível.

Um dos artefatos é o projeto de arquitetura que, na teoria, é desenvolvido pelo arquiteto de *software*, mas nem todas as equipes têm um profissional somente com essa função, podendo ser construído por um analista ou um programador sênior.

O projeto da arquitetura visa distribuir as classes de objetos relacionados do sistema em subsistemas e seus componentes, distribuindo também esses

componentes pelos recursos de *hardware* disponíveis. Na UML, os diagramas de componentes mostram a estrutura do sistema de *software* que descreve os componentes do *software*, suas interfaces e suas dependências. É possível utilizar diagramas de componentes para modelar sistemas de *software* em um alto nível ou para mostrar componentes em um nível de pacote mais baixo.

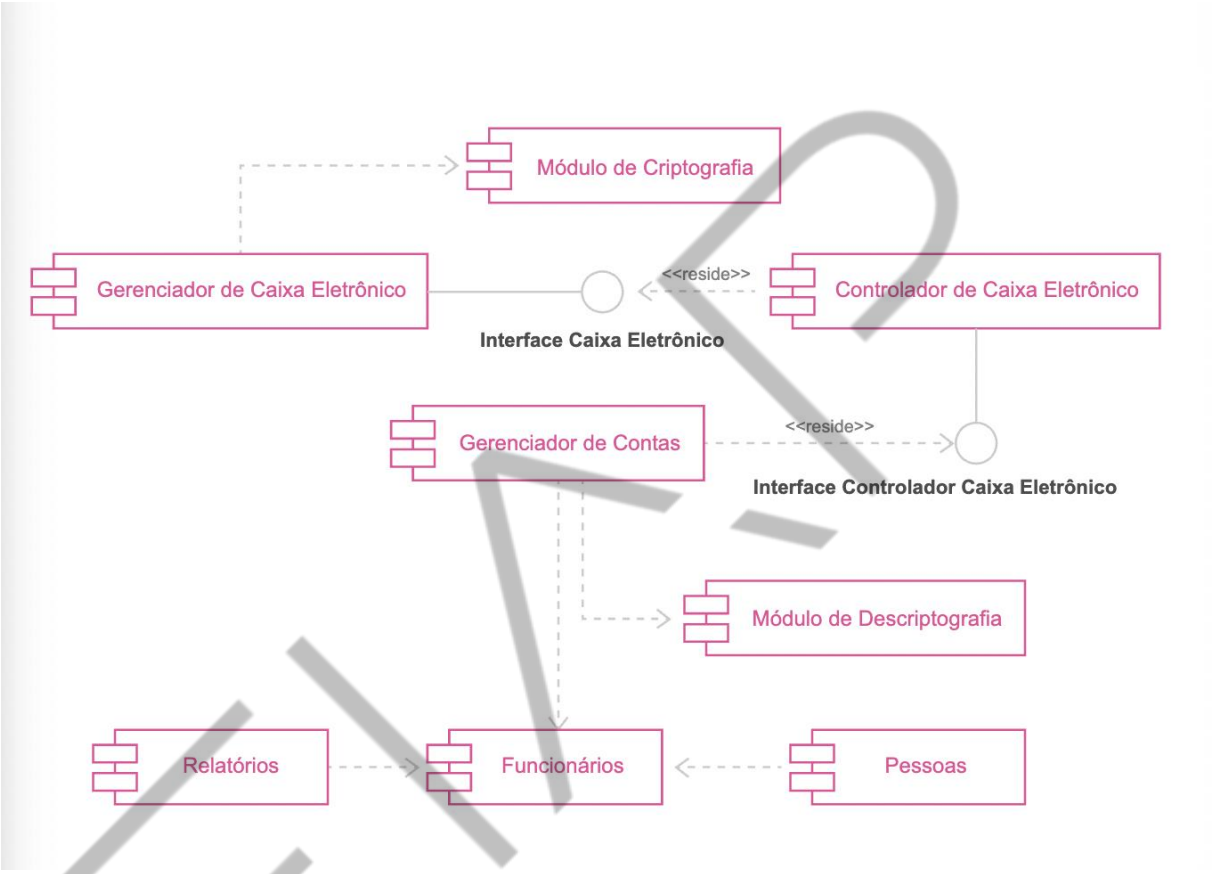

Figura 17 – Diagrama de componentes
Fonte: Elaborado pelo autor, adaptado por FIAP (2017)

Outro artefato da fase de projeto são as interfaces gráficas ou propriamente as telas do sistema, que já podem apresentar um resultado em relação à navegabilidade e usabilidade, por exemplo: em projetos de sistema web, podemos construir as interfaces e, nessa fase, apresentar para o usuário o resultado do sistema por meio do *front-end* navegável.

Até o momento não tínhamos nenhum artefato tão próximo à realidade final do usuário. Com a construção das telas, o usuário pode até ter a expectativa de que o sistema esteja pronto, o que, nesse momento, ainda não é um fator positivo. Em contrapartida, ele contribui para a validação, o que minimiza erros da implementação,

que é a próxima fase, para que seja bem fundamentada no resultado esperado pelo usuário.

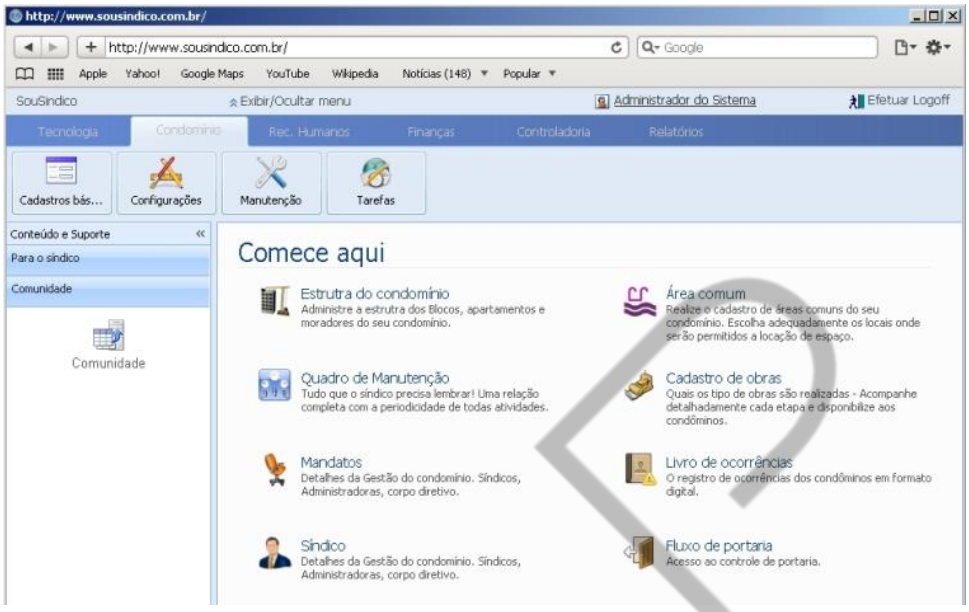

Figura 18 – Tela Web
Fonte: site Sou Síndico (2018)

Embora a fase de projeto seja, na teoria, após a análise, as duas, na prática, ocorrem em conjunto e acabam se misturando para que os resultados sejam alcançados.

2.4 Implementação

É a fase que traduz por meio de códigos de programação a modelagem do sistema. Como falamos de orientação a objetos, nessa fase surgem classes de implementação conforme a arquitetura e os padrões determinados nas melhores práticas de desenvolvimento.

A implementação é o transporte da lógica de funcionamento do sistema para um código de programação que gera resultados propriamente ditos. Nas fases anteriores, em que só havia modelos lógicos e interface de tela, agora passa a existir o sistema com suas funcionalidades, que devem atender às necessidades do usuário verificadas na fase de levantamento de requisitos.

2.5 Testes

É a atividade essencial que garantirá a qualidade do produto, pois estamos prestes a concluir o *software* para entregar ao usuário.

Engenheiros de *software* buscam qualidade (e desenvolvem atividades de garantia e de controle de qualidade) aplicando métodos e medidas técnicas sólidas, conduzindo revisões técnicas formais e efetuando um teste de *software* bem planejado (PRESSMAN, 2011).

A verificação é definida como um conjunto de atividades que asseguram que o produto será construído de maneira correta: “Estamos construindo corretamente o produto?”.

A validação é o conjunto de atividades que garantem que o produto correto está sendo construído: “Estamos construindo o produto certo?”.

O principal objetivo dessa fase é assegurar que o *software* cumpra com as especificações e atenda à necessidade e à realidade do usuário.

Nessa fase, ocorrem inúmeros testes, os quais podem ser realizados em dois âmbitos: pelo desenvolvedor e por profissionais de teste.

Os casos de teste definem quais são as informações de entrada estabelecidas pelo profissional que está realizando o teste manualmente ou por meio de *software* de teste e quais são os resultados esperados. Os casos de testes são determinados por funcionalidade.

Os testes são realizados com base no documento de casos de testes, que, por sua vez, utilizam a documentação dos casos de uso desenvolvidos na fase de análise.

Os principais artefatos dessa fase são os relatórios de testes com as evidências de erros ou que não foram detectados.

Podemos dividir os testes em:

- Teste de Função/Teste Unitário (exemplo: Ler Código do Produto).
- Teste de Funcionalidade (exemplo: Registrar Venda).
- Teste de Módulos (exemplo: Controle de Vendas).
- Teste de Integração (exemplo: Controle de Vendas x Controle de Estoque).

O teste unitário é uma modalidade que se concentra na verificação da menor unidade do projeto de *software*. Previne contra os erros (*bugs*) do *software* quando os códigos foram mal escritos.

O XP, como boa prática, propõe que primeiro é preciso projetar e escrever as classes de testes, depois as classes com regra de negócios. O Scrum define, como boa prática, que tanto o desenvolvimento quanto os testes devem obedecer a uma Definição de Pronto, que será detalhada no próximo capítulo, e que garantiria qualidade ao estabelecer diretrizes e patamares operacionais mínimos que permitam a um *software* estar pronto para ser entregue.

Alguns testes são funcionais e operacionais. Os testes operacionais podem ser Alfa quando ocorrem no ambiente do desenvolvedor e Beta quando ocorrem no ambiente do usuário.

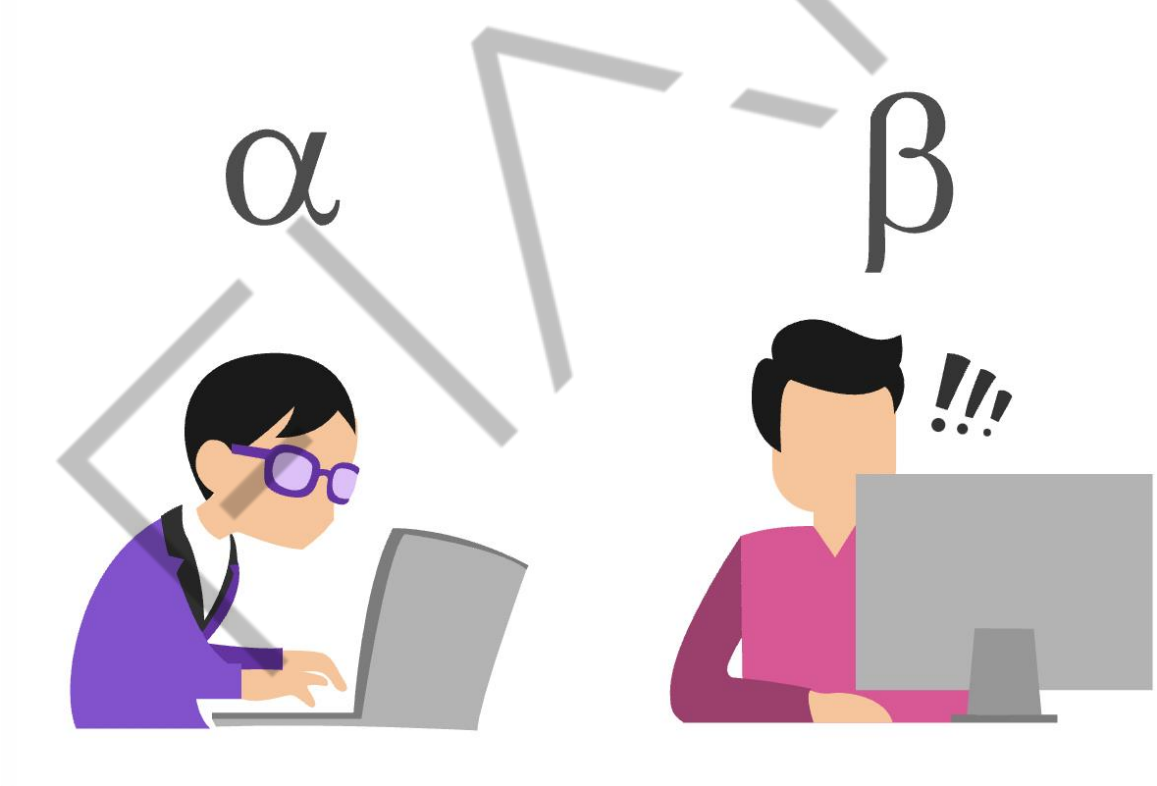

Figura 19 – Testes operacionais Alfa e Beta
Fonte: FIAP (2017)

Os testes funcionais podem ser de caixa branca e de caixa preta. O teste caixa branca é aquele em que os casos de teste devem ser gerados de maneira que, ao

serem executados conforme o fluxo do programa, passem por todos os comandos existentes.

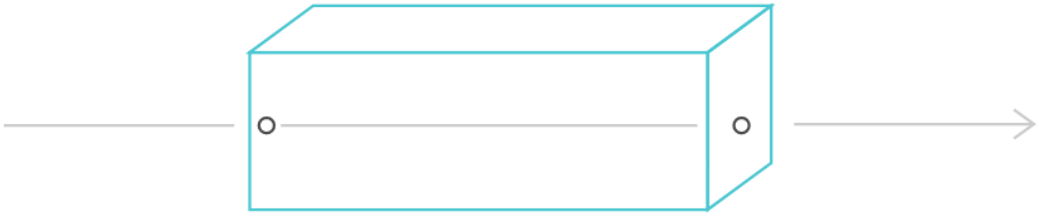

Figura 20 – Teste caixa branca
Fonte: FIAP (2017)

O teste caixa preta é aquele no qual os casos de testes gerados a partir da entrada de dados visa ao resultado com base nessas entradas.

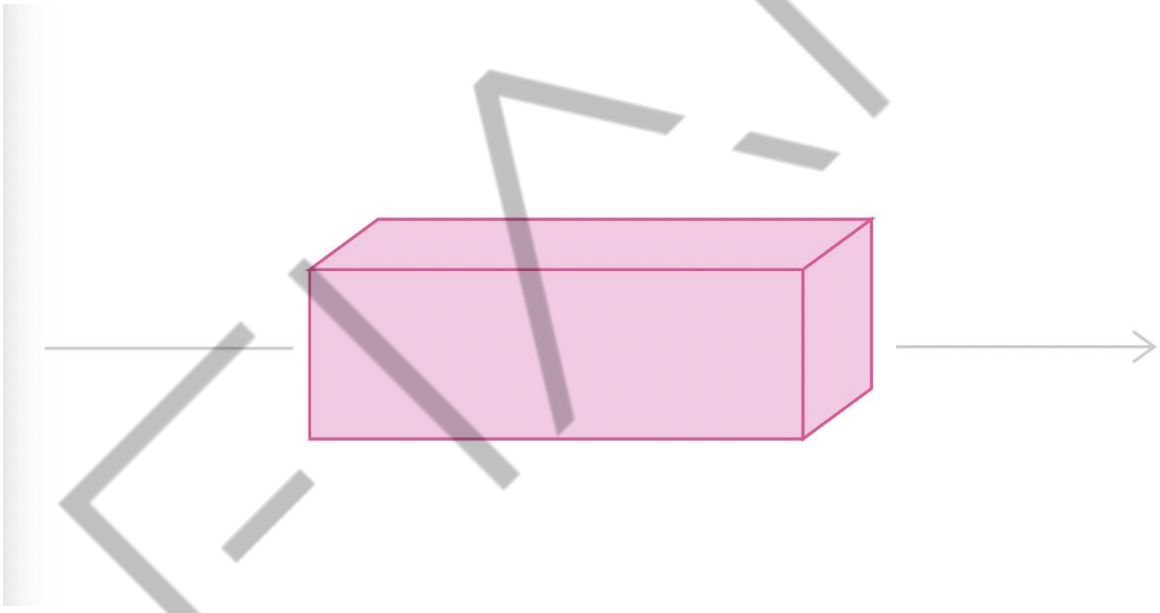

Figura 21 – Teste caixa preta
Fonte: FIAP (2017)

Garantir a qualidade é primordial para conseguir sucesso e credibilidade de qualquer projeto de *software*.

2.5.1 Implantação

Enquanto implementação é a fase de desenvolvimento, a implantação corresponde à fase em que o sistema irá para o ambiente de produção. Nesse

momento, a equipe de infraestrutura instala o sistema no ambiente real do usuário, onde ele executa suas atividades rotineiras. Por exemplo: caixa de supermercado, caixa eletrônico de um banco etc.

Os usuários são treinados para utilizar o sistema adequadamente. Um artefato criado para que o usuário tenha suporte em caso de necessidade é o manual do usuário, que explica quais ações ele deve executar nas funcionalidades do sistema.

2.5.2 Manutenção

Com a utilização do sistema, surgem novas necessidades ou detecta-se a existência de alguns *bugs* (erros) que não foram detectados inicialmente.

Para realizar a manutenção, é preciso ter a documentação do sistema. Nem sempre quem implementou é a pessoa que realizará a manutenção. Caso seja necessária uma mudança muito grande, deverá ser feito um novo projeto.

2.6 Modelos de desenvolvimento

O modelo de desenvolvimento representa a sequência das etapas do processo de desenvolvimento de *software*.

Um ciclo de vida corresponde a um encadeamento específico das fases para construção de um sistema (BEZERRA, 2014).

Entre alguns dos modelos de desenvolvimento, os dois mais comumente discutidos são: cascata, ou clássico, e o iterativo incremental.

2.6.1 Modelo cascata ou clássico

O modelo cascata, representado na Figura “Modelo de desenvolvimento cascata ou clássico” abaixo, demonstra que somente ao final do projeto o sistema será implantado, não existem versões disponibilizadas ao usuário antes da entrega final do projeto. Este modelo define que é possível ter todos os requisitos especificados antes das demais fases.

É um dos primeiros modelos de desenvolvimento e foi utilizado durante muitos anos em razão do crescimento da necessidade de *software.* A espera por um resultado final acabou se tornando inviável.

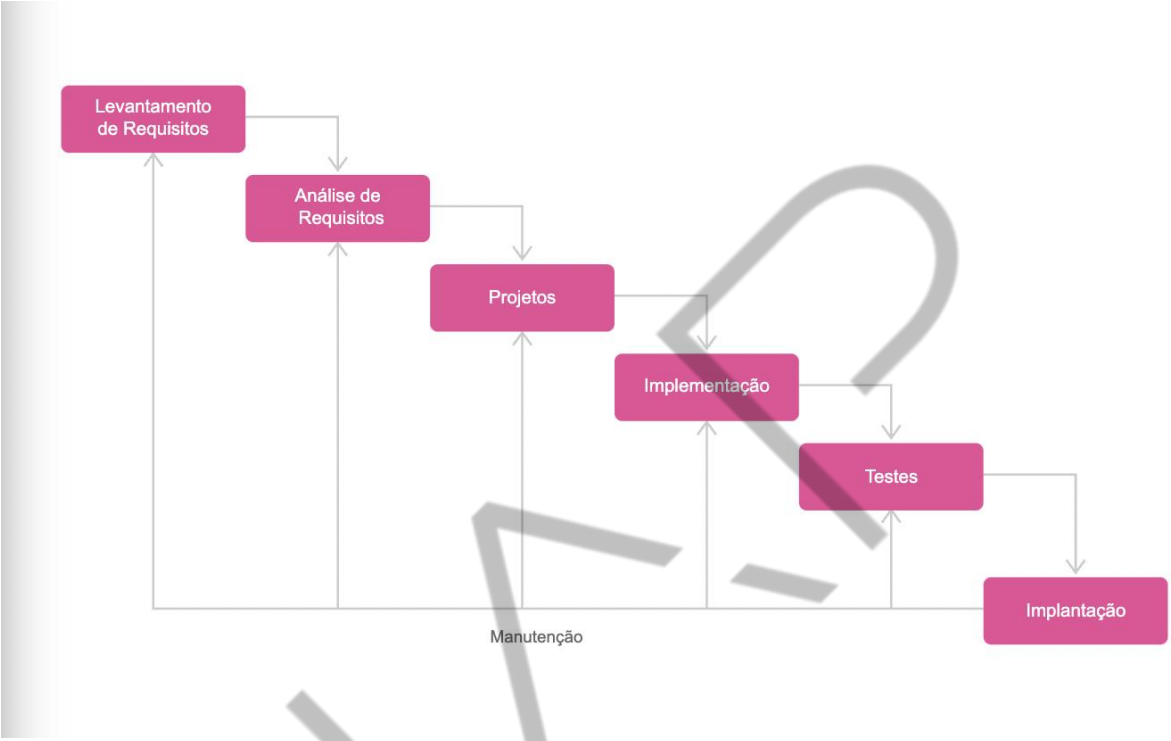

Figura 22 – Modelo de desenvolvimento cascata ou clássico
Fonte: FIAP (2017)

2.6.2 Modelo iterativo e incremental

O conceito de iterativo neste modelo está relacionado à repetição das etapas em ciclos diferentes e incrementais. A cada nova entrega, é realizado um incremento do *software*, ou seja, adquire novas funcionalidades.

É o modelo mais utilizado no mercado em decorrência da necessidade do usuário-cliente de receber porções do *software* antes de sua entrega por completo.

É separado por ciclos nos quais representa um conjunto de funcionalidades do sistema que podem ser entregues separadamente e devem ser interligadas a cada entrega, o que só é possível caso o sistema permita essa divisão. Tal abordagem difere do modelo cascata, no qual as fases são realizadas uma única vez.

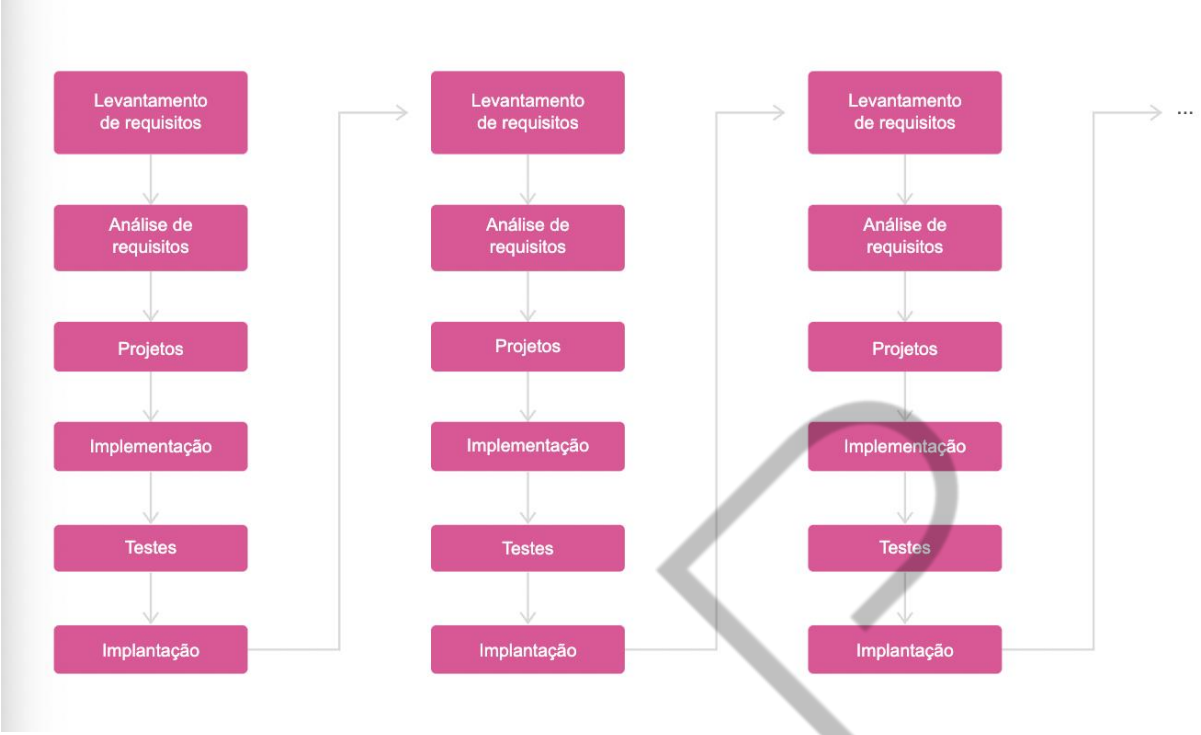

Figura 23 – Modelo Iterativo e Incremental
Fonte: FIAP (2017)

O modelo iterativo permite flexibilidade em ambientes de projeto onde existem indefinições de requisitos iniciais e a determinação do foco nos pontos mais críticos do projeto.

Outras vantagens desse modelo são a detecção, o mais cedo possível, de inconsistências entre os requisitos e a implementação, a produção de resultados tangíveis a cada iteração e a identificação de melhorias contínuas no processo (KRUTCHEN, 2000).

Na prática, é um modelo que apresenta resultados significativos para o usuário a cada entrega, com a proposta das boas práticas do XP – *Extreme Programming.* No primeiro ciclo, pode ser entregue o "*Business Value*" ao usuário-cliente, a funcionalidade que mais agregue valor ao seu negócio, demonstrando a importância do entendimento de sua necessidade em relação ao projeto de *software*.

3 PRÁTICAS ÁGEIS

No Scrum, as necessidades do negócio determinam as prioridades do desenvolvimento de um sistema e devem ser levantadas pelo Product Owner em conjunto com *stakeholders*, a fim de se obter a cumplicidade no trabalho. Além disso, as equipes se auto-organizam a fim de definirem a melhor maneira de entregar as funcionalidades de maior prioridade em um tempo que seja adequado ao time e ao cliente.

As entregas ocorrem a cada uma ou quatro semanas, quando todos podem ver o produto realmente funcionando, permitindo que seja avaliado, tendo um rápido feedback e decidindo se ele deve ser liberado ou se precisa continuar a ser aprimorado por mais um ciclo, chamado de Sprint.

Como você pode observar, vários paradigmas devem ser quebrados para se atingir o que é preconizado.

Portanto, não se começa a utilizar Scrum de um dia para o outro. É preciso apresentar aos envolvidos como ele funciona, suas cerimônias e suas restrições ao time, cliente e ao gerente de projeto, pois todos devem mudar de atitude frente à nova forma de executar um trabalho. E isso não é uma tarefa trivial. Além de um treinamento inicial, é preciso executar projetos pequenos como experimentações para que todos conheçam e se acostumem com a nova abordagem e vejam o resultado real de sua utilização.

> “Quando um projeto está atrasado, adicionar pessoas a ele servirá apenas para atrasá-lo ainda mais. Devemos considerar o tempo que perdemos em gestão e comunicação quando temos pessoas demais trabalhando em um projeto.
>
> Ao calcular o tempo de desenvolvimento de qualquer coisa, temos que dobrá-lo. O programador precisa de ‘tempo para pensar’ além do ‘tempo para programar’.”
>
> Frederick Brooks, *The Mythical Man-Month*: Essays on Software Engineering. 1. ed. Boston: Addison-Wesley, 1975.

A afirmação de Brooks (1975) reitera outro fundamento do Scrum, que **é manter inalterada a equipe e o tempo de um ciclo**, pois só assim é possível aprender com os erros e melhorar no ciclo seguinte, sem postergar prazos e gerar esforços adicionais que acabam por desmotivar e prejudicar o desempenho do time.

3.1 Características do *framework*

O principal objetivo do Scrum é permitir o controle das atividades de construção que utilizam técnicas ágeis para gerar informações gerenciais sobre o andamento e a evolução do projeto.

Em uma visão genérica inicial, podemos considerar as seguintes características principais do *framework* Scrum:

- Os requisitos formam o chamado ***Product Backlog***, que é a lista de tudo o que precisa ser feito no projeto ou produto.
- O produto evolui em ciclos curtos e de duração fixa chamados de ***Sprints***, cujos conceitos de sustentação são o desenvolvimento iterativo-incremental e o *timebox* (duração fixa).
- Durante a execução do Sprint, cada equipe utiliza a técnica de desenvolvimento que melhor se adapta ao cenário do projeto.
- As equipes se auto-organizam, ou seja, todos decidem o que e como fazer.
- Há um processo de melhoria contínua: ao final de cada Sprint, há uma série de coisas boas que devem ser repetidas e coisas ruins que não devem tornar a se repetir – lições aprendidas.

Vamos detalhar duas dessas características que fundamentam o conceito que deve ser seguido para a elaboração e execução dos Sprints e de todas as cerimônias do Scrum.

Importante: cerimônias são as reuniões que acontecem durante o ciclo Scrum de desenvolvimento, como as reuniões de planejamento, reuniões diárias, de revisões e de lições aprendidas que veremos à frente.

3.2 Desenvolvimento iterativo e incremental

É o modelo desenvolvido e caracterizado pela construção em vários ciclos de entregas constantes de produtos, permitindo a identificação de problemas nas fases iniciais do projeto e as tomadas de ações, visando corrigir o curso do desenvolvimento no tempo adequado e de maneira eficiente (KRUTCHEN, 2000). No Scrum, uma iteração é equivalente a um Sprint – ou seja, um Sprint é um ciclo de análise, desenvolvimento, teste e entrega, com uma duração específica que não deve ser alterada.

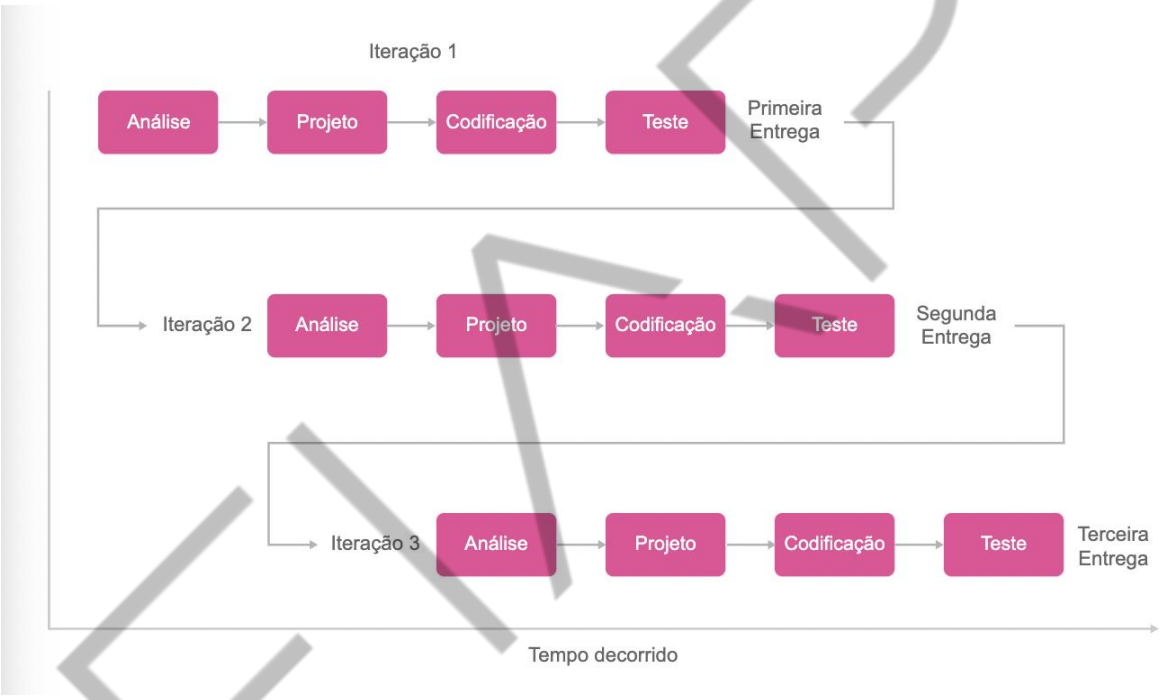

Figura 24 – Desenvolvimento iterativo
Fonte: Pressman (2005)

Como você pode observar na Figura “Desenvolvimento iterativo”, no modelo iterativo, a primeira iteração reúne os requisitos principais do produto, que são validados pelo cliente. Um plano é desenvolvido para a próxima iteração, visando satisfazer as necessidades do cliente. Esse processo é repetido após cada iteração até que o produto completo seja produzido. O modelo iterativo objetiva a elaboração de um produto a cada iteração (PRESSMAN, 2005).

O modelo iterativo permite flexibilidade em ambientes de projeto nos quais há indefinições de requisitos iniciais e a determinação do foco nos pontos mais críticos do projeto.

Outras vantagens desse modelo são a detecção, o mais cedo possível, de inconsistências entre os requisitos e a implementação, a produção de resultados tangíveis a cada iteração e a identificação de melhorias contínuas no processo (KRUTCHEN, 2000). É possível “juntar” vários incrementos resultantes de diversas iterações para compor uma versão ou um release. Assim, não é obrigatório que, ao fim de cada iteração, o software produzido já seja colocado em produção. O Product Owner, em conjunto com clientes e demais *stakeholders*, pode decidir que a melhor estratégia comercial é lançar uma versão apenas após duas, três ou mais iterações. Entretanto, a melhor prática é realizar entregas frequentes para que os usuários já possam ter o benefício de usar o *software* e, assim, validá-lo na prática.

3.3 Desenvolvimento *timebox*

O desenvolvimento *timebox* é uma prática que auxilia a manter o foco nas principais características do produto, evidencia o senso de restrição de tempo à equipe do projeto e reduz o tempo de construção.

O ponto central dessa prática é enquadrar os principais requisitos do projeto ao tempo disponível, enquanto os demais requisitos são incorporados em outros *timeboxes* com menor prioridade. A prioridade desses requisitos é definida conjuntamente pelo cliente e pela equipe do projeto, permitindo a redução do tempo de desenvolvimento.

A aplicação dessa prática requer a utilização conjunta com a prática de prototipação, além de necessitar do envolvimento significativo do usuário final e de revisões constantes da equipe do projeto. Após a construção, o sistema é avaliado pelo cliente, podendo ser aceito completamente ou retornar para ajustes de funcionalidades ou de qualidade, conforme ilustrado pela Figura “Prototipando o produto”.

Figura 25 – Prototipando o produto
Fonte: Pressman (2005)

A definição dos *timeboxes* deve ser parte da fase de planejamento, em que o cliente participa diretamente da definição e limitação dos principais requisitos que são atendidos em cada *timebox*, bem como da validação dos protótipos e das tomadas de decisões.

As principais condições de sucesso dessa prática é que a data final estabelecida para cada *timebox* não seja alterada, o cliente esteja de acordo com os requisitos definidos e a limitação rígida do escopo seja atendida. Qualquer mudança nessas condições deve ser tratada como um novo *timebox* para não comprometer os prazos estabelecidos.

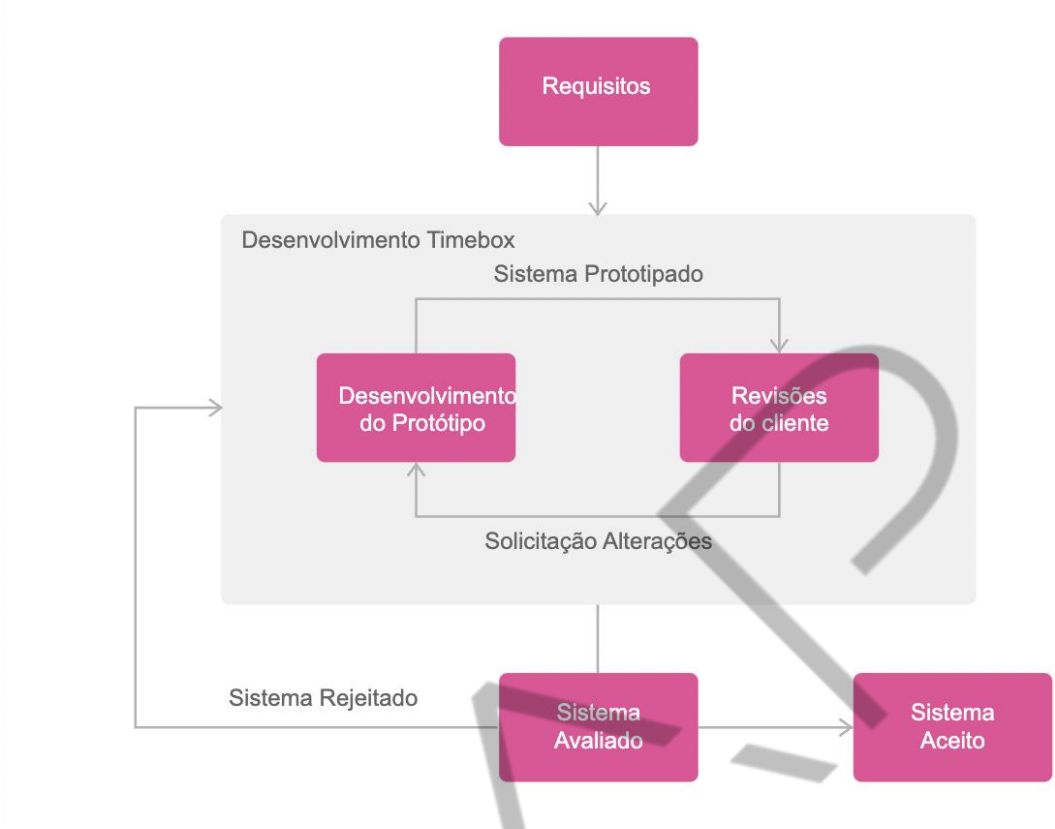

Figura 26 – Desenvolvimento *timebox*
Fonte: McConnell (1995), adaptado pela FIAP (2017)

O primeiro método criado para a gestão de projetos ágeis foi o Extreme Project Management (XPM).

No Scrum, o Sprint representa tanto as iterações quanto define o seu *timebox*, pois tem duração fixa e determina quando deve haver planejamento, desenvolvimento e entrega de um incremento.

3.4 Teste seus conhecimentos

Considere o estudo de caso a seguir. Preencha as lacunas que correspondem às características deste relato de desenvolvimento de *software*:

A empresa **Casa do Software Ltda.** tem várias equipes de desenvolvimento e, entre elas, a Laranja. No ano passado, essa equipe desenvolveu um aplicativo de locação de veículos para a **LoccaCar Ltda.**, em um projeto que durou 12 meses.

O desenvolvimento foi dividido em módulos e, em cada um deles, o escopo foi levantado, documentado, projetado, testado e implementado; somente com o módulo aprovado se avançava para o seguinte.

O app permite ao usuário cadastrar seus dados obrigatórios e até mesmo agendar uma locação de veículo; apenas usuários com e-mail e cartão de crédito verificado poderão locar um veículo.

Os testes foram realizados em um ambiente de desenvolvimento e em *smartphones* específicos para essa tarefa. O teste em modo depuração e passo a passo garantiu que cada procedimento codificado fosse implementado de maneira adequada e que estivesse funcionando perfeitamente.

Escolha, dentre as opções abaixo, as mais adequadas para o preenchimento de cada lacuna do quadro abaixo:

Caixa branca; Modelo Iterativo e Incremental; Cadastrar dados do cliente; Apenas usuários com e-mail e cartão de crédito verificados poderão locar um veículo; Alfa; Beta; Modelo Cascata ou Clássico; Locar veículo; Usuários não cadastrados poderão locar um veículo.

Modelo de desenvolvimento		
Testes		
Testes operacionais		
Requisito funcional		
Regra de Negócio		

Caso seja necessário, leia nossas dicas:

Dica – Modelo de desenvolvimento: o modelo interativo e incremental é separado por ciclos, nos quais representa um conjunto de funcionalidades do sistema que podem ser entregues separadamente e devem ser interligadas a cada entrega, o que só é possível caso o sistema permita essa divisão. Tal abordagem se difere do modelo cascata, no qual as fases são realizadas uma única vez.

Dica – Testes: o teste caixa branca é aquele em que os casos de teste devem ser gerados de maneira que, ao serem executados conforme o fluxo do programa, passem por todos os comandos existentes.

O teste caixa preta é aquele no qual os casos de testes gerados a partir da entrada de dados visa ao resultado com base nessas entradas.

Dica – Testes operacionais: os testes operacionais podem ser Alfa, quando ocorrem no ambiente do desenvolvedor, e Beta, quando ocorrem no ambiente do usuário.

Dica – Requisito funcional: os requisitos funcionais representam as funcionalidades do sistema. É o resultado esperado para o usuário, o principal artefato da fase de levantamento de requisitos.

Dica – Regra de Negócio: as regras de negócio representam os procedimentos, restrições, normas, cláusulas, critérios, políticas, ou seja, "como" executar o processo de negócio. De modo geral, representam a relação da informação para a execução do processo de negócio e determinam as condições para que os fatos sejam válidos.

Respostas:

Modelo de desenvolvimento	Modelo Interativo e Incremental	
Testes	Caixa Branca	
Testes operacionais	Alfa	
Requisito funcional	Cadastrar dados do cliente	Locar veículo
Regra de Negócio	Apenas usuários com e-mail e cartão de crédito verificado poderão locar um veículo.	

REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 3. ed. Rio de Janeiro: Campus, 2014.

BLAHA, M.; RUMBAUGH, J. **Modelagem e Projetos Baseados em Objetos com UML 2**. Rio de Janeiro: Elsevier, 2005.

BOOCH, G.; RUMBAUGH, J.; JACBSON, I. **UML Guia do Usuário**. 2. ed. Rio de Janeiro: Campus, 2005.

DAVENPORT, T. H. **Reengenharia de Processos**: como inovar na empresa através da Tecnologia da Informação. Rio de Janeiro: Campus, 1994.

GRAHAM, I. **Business Rules Management and Service Oriented Architecture**: A Pattern Language. Chichester: John Willey & Sons, 2007.

HAMMER, M.; CHAMPY, J. **Reengenharia Revolucionando a Empresa em Função dos Clientes, da Concorrência e das Grandes Mudanças da Gerência**. Rio de Janeiro: Campus, 1994.

HARRINGTON, H. J. **Aperfeiçoando Processos Empresariais.** São Paulo: Makron Books, 1993.

IBM. **Business Rules Management System**. Disponível em: <https://www-01.ibm.com/*software*/websphere/products/business-rule-management/whatis/>. Acesso em: 10 dez. 2020.

OMG. **Semantics of Vocabulary and Business Process**. Disponível em: <http://doc.omg.org/formal/08-01-02.pdf>. Acesso em: 10 dez. 2020.

PRESSMAN, R. S. **Engenharia de "*Software*".** 7. ed. São Paulo: Makron Books, 2011.

RUMBAUGH, G.; BLAHA, M. **Modelagem e Projetos Baseados em Objetos com UML2**. 2. ed. Rio de Janeiro: Campus, 2005.

SBC. **Sociedade Brasileira de Computação**. Disponível em: <http://www.sbc.org.br/>. Acesso em: 10 dez. 2020.

SOMMERVILLE, I. **Engenharia de *"Software".*** Tradução de Maurício de Andrade. 9. ed. São Paulo: Pearson, 2011.