UC9: Desenvolver Algoritmos.

CARGA HORÁRIA: **108 HORAS**

Indicadores

1. Planeja o desenvolvimento de software de acordo com as características do projeto e regras de negócio.
2. Desenvolve algoritmo de acordo com as melhores práticas de programação.
3. Desenvolve algoritmos computacionais de acordo com as premissas da linguagem selecionada.
4. Testa algoritmos computacionais de acordo com as orientações técnicas da linguagem selecionada.
5. Valida a estrutura de dados conforme os resultados dos testes dos algoritmos.

Elementos da competência

CONHECIMENTOS

- Regras de negócio: conceitos. Características. Tipos. Requisitos funcionais e não funcionais.
- Plataformas de desenvolvimento: conceitos. Tipos. Características e especificações técnicas.
- Metodologias de desenvolvimento de software: introdução a metodologias de desenvolvimento de software: Conceito e tipos. Metodologias tradicionais. Metodologias interativas: RUP. Metodologias ágeis: XP; SCRUM; FDD; entre outras.
- Lógica de Programação: Conceito de algoritmo. Algoritmos naturais e estruturados. Representações visuais, português estruturado ou linguagem algorítmica. Comandos de entrada, processamento e saída de dados. Variáveis e constantes. Expressões e operadores. Teste de mesa. Estrutura condicional simples e composta. Estrutura de repetição. Vetores. Matrizes.

HABILIDADES

- Resolver problemas lógicos e aritméticos.
- Representar expressões lógicas e matemáticas.
- Interpretar textos técnicos.
- Comunicar-se de maneira assertiva.
- Mediar conflitos nas situações de trabalho.

- Selecionar informações necessárias ao desenvolvimento do seu trabalho.
- Analisar as etapas do processo de trabalho.

ATITUDES/VALORES

- Zelo na apresentação pessoal e postura profissional.
- Sigilo no tratamento de dados e informações.
- Zelo pela segurança e pela integridade dos dados.
- Proatividade na resolução de problemas.
- Colaboração no desenvolvimento do trabalho em equipe.
- Cordialidade no trato com as pessoas.
- Responsabilidade no uso dos recursos organizacionais e no descarte de lixo eletrônico.

1. O que são regras de negócio?

Um negócio funciona por processos que, por sua vez, são formados por atividades relacionadas entre si. As funções das áreas de compras, estoque, logística, finanças, vendas e marketing, por exemplo, compõem um processo de fornecimento de um produto ao cliente.
Dentro destes processos, existem regras que devem ser seguidas durante a execução das atividades, que ajudam a definir COMO as operações devem ser realizadas e gerenciadas, por QUEM, QUANDO, ONDE e POR QUÊ. Podemos dizer que as regras de negócio são limites impostos às operações, de forma que elas sigam corretamente em direção às políticas e aos objetivos da instituição. De maneira geral, as regras de negócio devem:

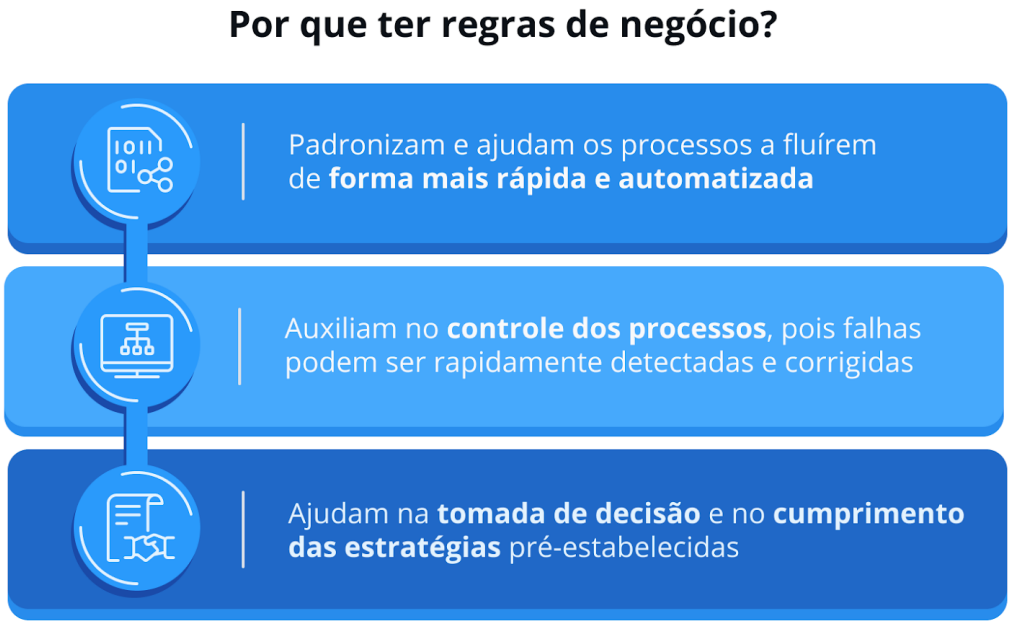
- Ter apenas uma função, ser indivisíveis e simples.
- Ser completas, com início, meio e fim.
- Ser possíveis de mensurar e rastrear.
- Estar em consonância com a legislação.
- Estar atualizadas e sempre revisadas.
- Refletir a política e os valores da organização.
- Ser inteligíveis para os colaboradores e envolvidos no processo.

1.1. O que são regras de negócio em TI?

As regras de negócio podem ser escritas ou efetuadas manualmente, como em manuais de boas condutas, mas a maneira mais eficaz quando tratamos de processos complexos como os de uma empresa, é a automatização. Imagine se uma regra do seu e-commerce é oferecer frete grátis em compras acima de R$100 e você tiver que aplicar esta mudança manualmente em cada entrega. Parece inviável, não é?

Para esta automação, utiliza-se de motores de regras ou softwares de *Business Rules Management Systems* (BRMS ou Sistemas de Gerenciamento de Regras de Negócio), aplicados à Gestão de processos. Estes sistemas são capazes de aplicar a regra em cada etapa do processo, de acordo com sua determinação. Mais à frente, falaremos mais sobre esses sistemas.

1.2. Importância das Regras de Negócio



Por que ter regras de negócio? Além de garantir o funcionamento dos processos, as regras:

- Padronizam e ajudam os processos a fluírem mais rápido, de forma automatizada, sem a necessidade de interferências manuais dos colaboradores. Assim, há um enorme ganho em agilidade, qualidade e eficiência, a partir do corte de etapas repetitivas ou dispensáveis e desperdício de tempo e recursos, reduzindo custos também.
- Auxiliam no controle dos processos, pois caso algo fuja das regras, a falha pode ser rapidamente detectada e corrigida.
- Ajudam na tomada de decisão e no cumprimento das estratégias pré-estabelecidas.

Se aplicadas em conformidade e forem mensuradas, as regras podem potencializar o desempenho da organização, que terá custos reduzidos e conseguirá gerar mais valor ao cliente. Para que isso aconteça, é importante que a comunicação na empresa e o fluxo de atividades sejam claros. Caso uma regra seja aplicada equivocadamente, todo o processo pode ser prejudicado e a falha detectada apenas ao final, quando entregue ao cliente.

1.3. Regras de negócio: exemplos para esclarecer

Daremos alguns exemplos para que você compreenda melhor sobre as regras de negócio. Em geral, as regras se aplicam à operação de negócio, decisões, uso de dados, procedimentos e políticas, regras financeiras e outras.

- Em um controle de qualidade de granja, pode-se dizer que a cada 100 ovos impróprios para consumo, o lote será descartado.
- Em um banco, clientes com faturamento mensal de mais de R$ 25 mil e CPF sem restrições, serão atendidos pelo gerente Premium pessoa física.
- Para conclusão de licitações, devem ser feitos três orçamentos e o vencedor será sempre o de menor preço final.
- Em um processo de seleção de RH, o candidato só pode ser aprovado se tiver mais de 5 anos de experiência na área, diploma de pós-graduação, espanhol fluente e pretensão salarial abaixo de R$ 8.000,00.

1.4. Como aplicar as regras na prática?

Por meio de uma análise dos procedimentos internos, observando regras determinadas por órgãos externos para que haja o *compliance* da empresa, é possível entender o que traz um melhor funcionamento dos processos, sempre com foco nos objetivos da corporação.

dessas normas também colabora para a manutenção de um padrão de qualidade e de um código de conduta que esteja alinhado aos valores corporativos.

A revisão dessas regras deve ser constante, considerando possíveis mudanças na legislação, a introdução de outras regras que não eram vistas em um primeiro momento e principalmente a compreensão de como essas normas impactam a produtividade.

1.5. Como essas regras se aplicam na tecnologia?

Dentro de contexto tecnológico, essas regras são aplicadas no desenvolvimento de softwares, usando as diretrizes para determinar os fluxos na programação. Os programas são desenvolvidos para atender diversas necessidades da empresa, como controle de estoque, emissão de pedidos, contabilidade, recursos humanos, apenas para citar algumas aplicações.

De forma geral, nessas atividades há regras de negócio para regularizar os processos. Por exemplo, um controle de estoque é atualizado com o processamento de notas fiscais de entrada e saída. Essa relação deve ser indicada na programação do software, para que seja feita automaticamente, representando o uso das regras de negócio na tecnologia.

Um equívoco de código na programação desse sistema pode impedir a aplicação da regra no controle de estoque, e não aplicar esse parâmetro importante para a empresa. Com a determinação e consulta das regras do negócio, é possível desenvolver um sistema que atenda a uma demanda específica e, ao mesmo tempo, manter a prática do que é importante para a corporação.

Assim como as regras de negócio devem passar por revisões periódicas, é importante implementar essas revisões também nos softwares.

Podemos tomar como exemplo um programa de contabilidade pensado para aplicar automaticamente uma porcentagem fixa de desconto. A solução pode ficar obsoleta com uma mudança na cobrança de impostos. Por isso, também deve ser adaptado para que esses valores possam ser atualizados, de acordo com as mudanças na regras de negócio.

1.6. Como é feito o desenvolvimento de um software com base nas regras de negócio?

Na hora de programar a solução que será usada para resolver problemas ou agilizar processos internos da empresa, é preciso considerar aspectos importantes para um resultado satisfatório.

Para exemplificar tais aplicações, destacamos aqui os mais importantes:

1.6.1. Definição de requisitos

O software é projetado com base nos requisitos definidos, isso significa que como primeiro passo é importante determinar para que a solução será empregada e quais regras ela deve obedecer em seu funcionamento.

Nessa fase, é importante conversar com os representantes do setor envolvido para entender as necessidades. Por exemplo, se o software será usado pela contabilidade, os desenvolvedores precisam entrevistar o gestor da área para compreender como o programa será usado e quais normas deve obedecer.

1.6.1.1. Requisitos de usuário

Os requisitos provenientes dos *stakeholders* primários são a fonte mais importante de informações para o entendimento das necessidades reais do cliente.

É fundamental que além de competência e conhecimento técnico da engenharia de software, o analista que seja um exímio ouvinte e saiba em cada ocasião qual a melhor maneira de extrair as verdadeiras “dores” dos *stakeholders* do projeto.

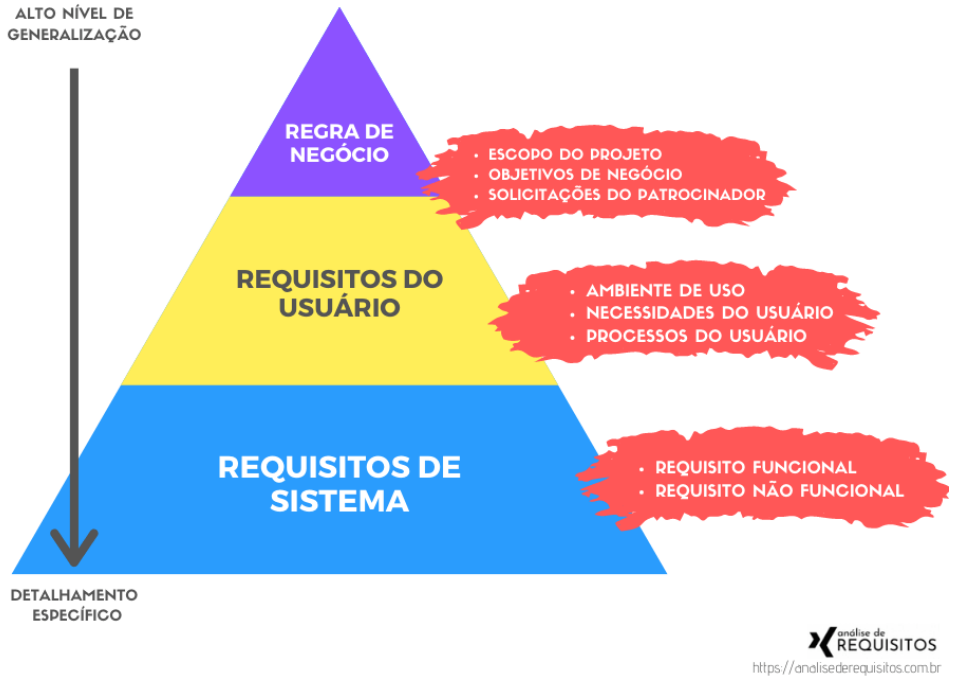
1.6.1.2. Requisitos sistema ou de solução

Essa categoria de requisitos caracteriza-se por descrever as características do produto que atenderão às suas expectativas e necessidades de negócios.

Os requisitos de solução são divididos em dois tipos:

- Requisitos funcionais que descrevem as maneiras como um produto deve se comportar.
- Requisitos não funcionais, também conhecidos como atributos de qualidade, descrevem as características gerais do software

Agora vamos esclarecer esses dois tipos de requisitos: “Requisitos Funcionais” e “Requisitos Não Funcionais”.



1.6.1.2.1. O que é Requisito Funcional?

Requisitos funcionais são todas as necessidades, características ou funcionalidades esperadas em um processo que podem ser atendidos pelo software.

De forma geral, um REQUISITO FUNCIONAL expressa uma ação que deve ser realizada através do sistema, ou seja, um requisito funcional é “o que sistema DEVE fazer“.

Um clássico e simples exemplo de requisito funcional é a funcionalidade “MANTER USUÁRIO” (o verbo “manter” é utilizado na documentação de software para referir-se à uma funcionalidade de cadastro, que contempla a inclusão de um novo item, a alteração de um item, a exclusão de um item além de da leitura de suas informações).

O requisito que detalha a funcionalidade “MANTER USUÁRIO” engloba uma série de outros requisitos menores, por vezes chamados de “*features*“, como no exemplo abaixo:

Manter usuário:

- Cadastrar novo usuário.
- Alterar usuário.
- Excluir usuário.
- Consultar usuário.

O requisito principal de negócio “manter usuário” é responsável por toda a manutenção do cadastro usuários, por isso a utilização do verbo “manter“.

Mas afinal, por que eles são chamados de requisitos funcionais? A categorização dos requisitos citados como requisitos funcionais se deve ao fato de que todos eles são funcionalidades atendidas através de uma ação do software ou comportamento específico do sistema.

Podemos dizer que é considerado um requisito funcional, todo cenário onde o usuário informa um dado, ou um sistema terceiro realiza uma solicitação qualquer durante uma interação com o sistema, que então, responde com determinada ação correspondente.

Ainda como requisitos funcionais, podemos citar algumas funcionalidades muito comuns durante o processo de análise e levantamento de requisitos em um projeto de software.

Esses cinco requisitos que citamos acima são muito comuns em sistemas de controle financeiro, fiscal ou contábil. Todos eles representam uma funcionalidade que o sistema deve executar em decorrência de uma solicitação ou ação do usuário, assim podemos facilmente identificá-los como requisitos funcionais.

Devemos porém lembrar, de que um requisito funcional pode também ser executado como sequência da execução de um requisito anterior, que incluí tal requisito em sua execução.

Quando o sistema executa um requisito funcional?

Para ficar claro esta característica dos requisitos funcionais basta pegar como exemplo o caso de um sistema de vendas online.

Imagine que um sistema possua um requisito chamado “FECHAR VENDA”, durante sua execução ele inclui outros dois requisitos funcionais – ou features: “EMITIR NF-C” e “ATUALIZAR ESTOQUE”.

1.6.1.2.2. Requisitos funcionais X Requisitos não funcionais

Os requisitos funcionais por definição, e conforme explicamos, é uma característica, funcionalidade ou necessidade que o sistema deve contemplar, ou seja, um requisito funcional é ‘tudo aquilo que o sistema “DEVE FAZER”.

Já um requisito NÃO FUNCIONAL, por sua vez pode ser definido como “de qual maneira o sistema deve fazer“. Por outro lado pode parecer muito vago e com pouco sentido, mas é muito simples assimilar o conceito.

Uma forma simples de entender o que é um requisito funcional é ter por base que todo requisito não funcional deve expressar uma premissa ou restrição do sistema.

Dessa forma, requisitos não funcionais devem sempre ser mensuráveis, ou seja, deve ser possível verificar se ele está ou não sendo atendido pelo software.

Exemplo de Requisito não Funcional:

Vamos dar alguns clássicos e básicos exemplos de requisitos não funcionais de software, que são comuns durante o levantamento de requisitos de projeto de desenvolvimento:

- O sistema deve ser multiplataforma – Windows, Linux e macOS.

- O desenvolvimento deve ser em linguagem C++.
- O programa deve funcionar offline.
- O sistema deve respeitar o tempo máximo de 160 segundos durante processamentos.

Os quatro requisitos acima citados, são requisitos NÃO FUNCIONAIS, pois eles indicam condições ou então características de COMO será executada determinada ação pelo sistema. Lembre-se: requisitos não funcionais devem sempre ser mensuráveis!

Tirinha de Dilbert: A definição e o entendimento do escopo de negócio e dos requisitos de software nem sempre são entendidos ou respeitados

1.7. Exercícios de fixação de conteúdo.

De acordo com a explicação, em grupos definam:
- Regras de negócio para um sistema de Pizzaria;
- Definam os stakeholders (usuários);
- Definam os requisitos funcionais;
- Definam os requisitos não funcionais.

2. Introdução às plataformas de *software*

O que há de comum entre um aplicativo para previsão do tempo no iPhone, uma calculadora em um smartphone Android e um game dentro do Facebook?

O fato de que todos foram desenvolvidos sobre plataformas de software. Interessante que são três empresas diferentes, mas que encontram uma forma de agradar seu público abrindo espaço para que desenvolvedores pudessem desenvolver e comercializar suas próprias criações digitais.

Quando pensamos nas mais diferentes tecnologias que estão ao nosso redor é fácil perceber que estamos vivendo a “era da plataforma”. Mas afinal quais são os fatores que definem uma plataforma de software? Quais as vantagens para as empresas e seus clientes quando adotam uma plataforma de software? Por que jovens nerds estão se tornando milionários desenvolvendo aplicativos “sobre” plataformas de software? Antes, porém de responder estas perguntas fundamentais, importante revisar alguns conceitos fundamentais, os quais explicam o sucesso desta inovação.

2.1. Algoritmos

Antes do software existe o algoritmo, um conjunto de instruções que descreve, passo a passo, como uma tarefa será realizada. Estas instruções formam uma sequência lógica, a qual uma vez realizada, leva à conclusão da tarefa. Uma linguagem de programação (Java, C++, Javascript etc) é usada para codificar um algoritmo, este é o software.

2.2. *Software* proprietários e livres

Nos primórdios da computação, praticamente todo software era proprietário. A arte da programação estava restrita às empresas que vendiam programas e aplicativos. Com a chegada do software livre e de linguagens de programação abertas este cenário mudou radicalmente.

Linguagens de programação tais como Java, Python e PHP e sistemas operacionais gratuitos tais como o Linux, começaram a ser usados por talentosos programadores, os quais colocaram em xeque a estrutura fechada do mercado de software. A arte da programação tornava-se acessível para todos e várias organizações surgiram para promover o software livre, entre estas a *The Open Source Initiative* (OSI).

Em pouco tempo, surgiram bancos de dados livres tais como o *Firebird* e o *PostgreSQL*, gerenciadores de conteúdo tais como Joomla! e o Drupal, e tantos outros produtos em diferentes áreas.

2.3. Compartilhamento e colaboração

Um efeito colateral do movimento de software livre foi o de estabelecer duas atitudes de trabalho muito diferentes do que a indústria de software vinha adotando: compartilhar e colaborar.

Se na era do software proprietário o segredo e as patentes eram palavras de ordem, os jovens da nova era do software livre adotaram o compartilhamento aberto e irrestrito das informações, dos códigos fonte e conteúdos.

Além disso, colaborar e trabalhar em equipes multidisciplinares, muitas vezes 100% virtuais também se tomaram práticas comuns. Não por acaso, as empresas e startups que surgiram nesta onda publicam abertamente seus códigos fonte e trabalham em conjunto com os usuários de forma aberta e irrestrita.

2.4. Plataformas de software

A Motorola possui uma das melhores definições para plataforma:

“Plataforma é um conjunto de ativos que podem ser usados para alavancar o reuso e o rápido desenvolvimento de novos produtos.”

No mínimo, ela define o ambiente operacional, a arquitetura em alto nível de todos os produtos desenvolvidos com base nesta plataforma, e um conjunto de políticas de desenvolvimento para aperfeiçoar a plataforma e o desenvolvimento de produtos.”

Quando entramos no ambiente de desenvolvimento de software, os “ativos” são as ferramentas de programação e os “produtos” são os aplicativos. Entre estas ferramentas estão:

1. Kits ou pacotes para o desenvolvimento de aplicativos – SDK (Software Development Kit);
2. Emuladores ou Simuladores, que permitem ao desenvolvedor visualizar e testar seu aplicativo como se estivesse no ar;
3. APIs (Application Program Interfaces) as quais definem padrões e especificam como os diferentes componentes da plataforma se comunicam;
4. Bibliotecas e frameworks, os quais podem ser usados pelos desenvolvedores para agilizar o desenvolvimento dos aplicativos.

Mas as atuais plataformas de software de maior sucesso, e aqui estou me referindo a Apple, Google Android e Facebook, estão indo além, adicionando três importantes componentes:

1. Fator social

Para um aplicativo obter sucesso é preciso que ele tenha volume de acessos ou downloads. Para os jovens desenvolvedores e mesmo para as empresas, uma plataforma que tenha ao redor de si milhões de usuários é um fator crítico.

Ao oferecer um ambiente de compartilhamento, divulgação e mídia espontânea, as plataformas estão potencializando os bons aplicativos.

2. Multiplataforma

Um dos fatores mais importantes de sucesso de um aplicativo está relacionado a facilidade com que o usuário possa acessá-lo e isto significa poder acessá-lo de qualquer plataforma, seja web, tablet ou mobile.

Neste sentido, o que temos hoje é o Facebook como plataforma web, enquanto Apple e Google Android estão estabelecidos nas plataformas móveis e tablets. Mas este cenário deverá mudar nos próximos anos, com o Facebook estendendo sua plataforma para smartphones e tablets.

3. Comercialização

Aplicativos podem ser gratuitos ou pagos. Alguns podem adotar ainda o modelo *Freemium* (modelo de negócios onde o internauta adota um aplicativo gratuitamente, mas para ter mais recursos deve pagar).

Reconhecendo que as integrações e requisitos técnicos para adoção de modos de pagamento são um "gargalo" para os desenvolvedores de aplicativos, as plataformas oferecem modelos prontos para comercialização dos aplicativos.

3. Metodologia de desenvolvimento de Software

Sistemas de software têm desempenhado um papel cada vez mais preponderante no dia-a-dia das pessoas, e em muitas situações o funcionamento correto ou incorreto desses sistemas pode ser a diferença entre a vida e a morte. Entretanto, a construção de sistemas é complexa, pois deve lidar com requisitos intransigentes, restrições de integridade e a necessidade de um vasto conhecimento sobre a aplicação para que as interações esperadas entre o software e o ambiente possam ser adequadamente descritas. Quando os requisitos não são totalmente compreendidos, registrados e comunicados para a equipe de desenvolvimento, muito provavelmente, haverá discrepância entre o que o sistema construído faz e o que ele deveria fazer.

Hoje em dia o software assume um duplo papel. Ele é o produto e ao mesmo tempo o veículo para entrega do produto. Como produto ele disponibiliza o potencial de computação presente em computador, ou mais amplamente numa rede de computadores acessível pelo hardware local. Quer resida em um telefone celular, quer opere em um computador de grande porte (Mainframes) o software é transformador de informações – produzindo, gerando, adquirindo, modificando, exibindo, ou transmitindo informação, que pode ser tão simples como um bit ou tão complexa como uma apresentação multimídia. Como veículo usado para a entrega do produto, o software age como base para controle do computador – sistemas operacionais – para a comunicação da informação e para a criação e o controle de outros programas.

Atualmente, o desenvolvimento de software não ocorre como no passado, o programador solitário foi substituído por uma equipe de especialistas com cada um se concentrando numa parte da tecnologia necessária para produzir uma aplicação. No entanto, os mesmos questionamentos feitos ao programador solitário estão sendo feitos nos dias atuais:

- Por que leva tanto tempo para concluir o software?
- Por que os custos de desenvolvimentos são tão altos?
- Por que não podemos achar todos os erros antes da entrega do software aos clientes?
- Por que continuamos a ter dificuldades em avaliar o progresso enquanto o software é desenvolvido?

Há alguns anos tem-se discutido maneiras de contornar a complexidade do software, visto que a cada dia novas áreas de aplicação têm surgido e exigido mais confiabilidade e precisão dos softwares já existentes e dos que ainda virão a ser construídos. Não há consenso sobre qual a melhor prática para o desenvolvimento de sistemas de software, mais existe um esforço em encontrar soluções para reduzir as dificuldades oriundas da própria natureza do software, da sua complexidade, de necessidades de cumprir seus objetivos e da rapidez com que sofre alterações. A Engenharia de software propõe a adoção da disciplina para lidar com essas dificuldades, tentando reduzir ao máximo a influência delas no processo de desenvolvimento de software.

3.1. Engenharia de software

A Engenharia de software é uma disciplina que reúne metodologias, métodos e ferramentas a ser utilizadas, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolver problemas inerentes ao processo de desenvolvimento e ao produto de software.

O objetivo da Engenharia de software é auxiliar no processo de produção de software, de forma que o processo de origem a produtos de alta qualidade, produzidos mais rapidamente e a um custo cada vez menor. A Engenharia de software segue o conceito de disciplina na produção de software, fundamentado nas metodologias, que por sua vez seguem métodos que utilizam de ferramentas automáticas para englobar as principais atividades do processo de produção.

3.2. Metodologia de desenvolvimento de software

Já faz alguns anos que o desenvolvimento de software deixou de ser sinônimo apenas de código. Hoje em dia, sabe-se que é necessária a utilização de uma metodologia de trabalho.

Mas o que é necessariamente uma metodologia de software? Entende-se por metodologia, como a maneira – forma – de se utilizar um conjunto coerente e coordenado de métodos para atingir um objetivo, de modo que se evite, tanto quanto possível, a subjetividade na execução do trabalho. Fornecendo um roteiro, um processo dinâmico e interativo para desenvolvimento estruturado de projetos, sistemas ou software, visando à qualidade e produtividade dos projetos.

O dicionário define metodologia como um conjunto de métodos, regras e postulados empregados por uma disciplina: um procedimento particular ou conjuntos de procedimentos.

É objetivo de uma metodologia definir de forma clara “quem” faz “o que”, “quando”, “como”, e até mesmo “onde”, para todos os que estejam envolvidos diretamente ou não com o desenvolvimento de software. Deve definir também qual o papel dos técnicos, dos usuários, e o da administração da empresa no processo de desenvolvimento. Com isso, evita-se a situação a qual o conhecimento sobre o sistema é de poucos, comumente apelidados, de “os donos do sistema”. Além disso, deve instruir um conjunto de padrões preestabelecidos, de modo a ser evitar a subjetividade na abordagem, a fim de garantir fácil integração entre os sistemas desenvolvidos. Com isso, o uso de uma metodologia possibilita:

- Ao gerente: controlar o projeto de desenvolvimento de software mantendo o rumo do projeto sobre controle para que não haja desvios de planejamentos de custos e prazos, que, se negligenciados ou mal conduzidos, podem por em risco o sucesso do projeto.
- Ao desenvolvedor: obter a base para produzir de maneira eficiente, software de qualidade que satisfaça os requisitos estabelecidos.

Muitas vezes, o uso de uma metodologia é encarado como cerceamento da criatividade dos técnicos, ou como, acréscimo de burocracia, leia-se documentações, por muitos tidos como desnecessário a construção de software. Uma metodologia não deve limitar a criatividade profissional, mas deve ser um instrumento que determine um planejamento sistemático, que harmonize e coordena as áreas envolvidas. O que limita a criatividade não é a metodologia, mas os requisitos de qualidade e produtividade de um projeto.

Como uma metodologia é um conjunto de métodos, convém definir o que é um método e qual o seu objetivo.

Um método é abordagem técnica passo a passo para realizar uma ou mais tarefas indicadas na metodologia. Ou seja, é (são) o(s) procedimento(s) necessário(s) a ser (em) adotado(s) para atingir um objetivo. Já uma técnica, pode ser compreendida como sendo um modo apropriado de se investigar sistematicamente um universo de interesse ou domínio do problema. Para tanto, utiliza-se de uma notação. Como exemplo de técnica, temos: Análise estruturada, Análise Essencial, Projeto Estruturado, Análise Orientada a Objetos.

A escolha de uma metodologia a ser utilizada no desenvolvimento, deve ser realizada com base na natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramentas a serem utilizadas e dos controles e produtos intermediários desejados.

3.2.1 Metodologias Tradicionais e interativas

Metodologias tradicionais, também chamada de metodologias orientadas a documentação, foram muito utilizadas no passado, quando o contexto de desenvolvimento de software era bem diferente do que temos atualmente. Não existiam ferramentas de apoio e qualquer alteração solicitada no andamento do processo era inviável pois o custo para tal era muito alto.

Algumas destas metodologias tradicionais são utilizadas até hoje, para realização de pequenos projetos, e ainda podem ser encontradas como forma de organização no mercado de trabalho sendo elas: se

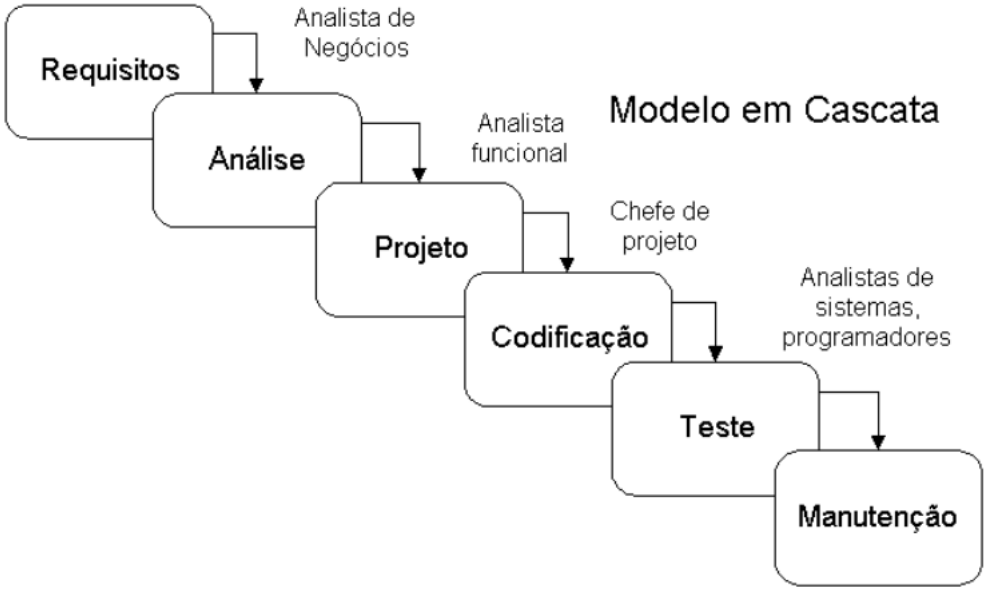
- Cascata;
- Prototipação;
- Espiral.

3.2.1.1 Cascata

O modelo cascata foi uma das primeiras metodologias criadas para organização de desenvolvimento de software. Também chamada de linear ou sequencial por se basear em uma secessão de tarefas / etapas onde uma só é iniciada quando a anterior for concluída. A imagem abaixo ilustra o fluxo seguido pela metodologia em questão.

Nessa metodologia, inicialmente procura-se compreender completamente o problema, a ser resolvido, seus requisitos e suas restrições; depois projeta-se soluções que atendam a todos

os requisitos e restrições. Feito isto inicia-se a implementação do projeto e quando toda a etapa de implementação é concluída verifica-se junto ao cliente se a solução atende aos requisitos estabelecidos e por fim é efetuada a entrega do produto (KROLL e KRUCHTEN, 2003 apud LOURENÇO, 2011).
A abordagem adotada pela metodologia cascata acaba trazendo alguns problemas. Dentre estes problemas merece destaque o fato de que os projetos reais dificilmente seguem o fluxo sequencial, o cliente quase sempre não consegue exprimir todas as suas necessidades além de ser exigida dele muita paciência visto que o software só estará pronto para uso num ponto tardio do cronograma. E o maior dos problemas é que se ocorrer um erro em qualquer uma das etapas o resultado pode ser desastroso e frequentemente caro (PRESSMAN, 2006).
Essa metodologia é uma instrução que vai de requisitos à manutenção.

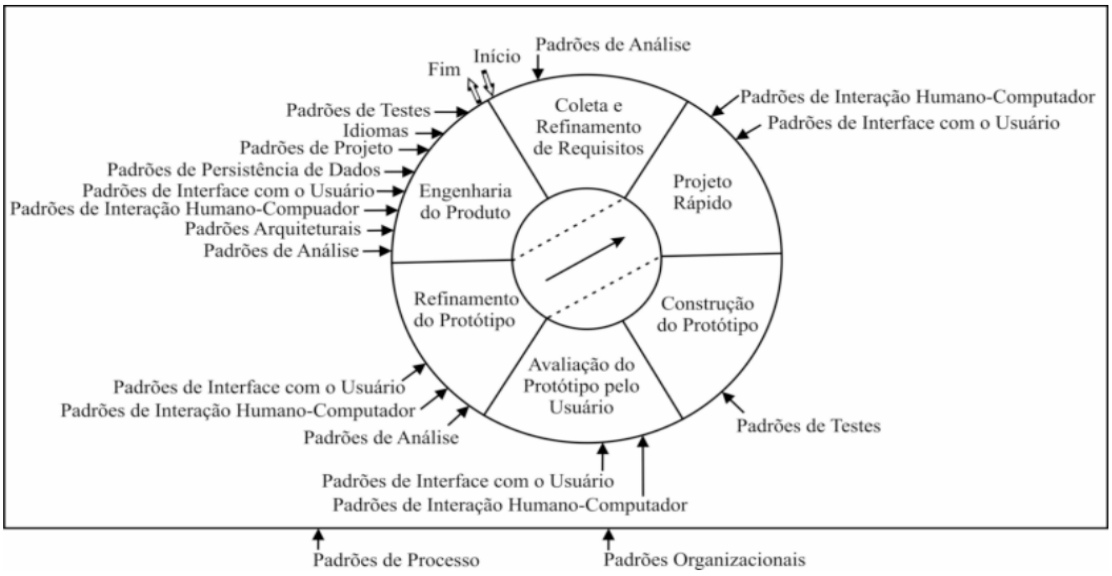


Fonte: https://ead.uepg.br/apl/sigma/assets/editais_antigos/PS0056E0077.pdf

Algumas vezes o cliente define um conjunto de objetivos gerais que não esclarecem consistentemente os requisitos. Outras vezes o desenvolvedor não tem certeza da eficiência de parte do código, da adaptação da aplicação ao sistema operacional ou mesmo da forma que interação homem-máquina deve ter. Visando fornecer melhores soluções à casos assim surgiu a metodologia conhecida como prototipação.

3.2.1.2 Prototipação
Prototipação é uma metodologia surgida posteriormente à Cascata. Ela possibilita a equipe de desenvolvimento a criar uma aplicação protótipo que pode assumir três formas distintas. A primeira delas é um **protótipo em papel ou mesmo no computador** que retrate a interação homem-máquina. A segunda opção é **implementar uma funcionalidade que já está no escopo** do software a ser desenvolvido. Por fim existe a possibilidade de **utilizar-se de um software já pronto** que tenha parte ou todas as funcionalidades desejadas. Esta forma é mais comumente adotada em softwares que apesar de prontos ou parcialmente prontos possuem características que precisam ser incrementadas ou melhoradas em um novo esforço de desenvolvimento (PRESSMAN, 2006).
Geralmente o protótipo serve apenas como um mecanismo para identificar requisitos de software. Isto ocorre porque na maior parte dos casos o primeiro sistema construído não é completamente usável. Normalmente ele possui uma série de problemas que só serão corrigidos em uma versão reprojetada na qual as deficiências sejam corrigidas (BROOKS,1975 apud PRESSMAN, 2006).
Essa metodologia também apresenta pontos negativos, um deles é a ilusão que o cliente tem de acreditar que o protótipo criado já é seu software finalizado ou pronto para ser

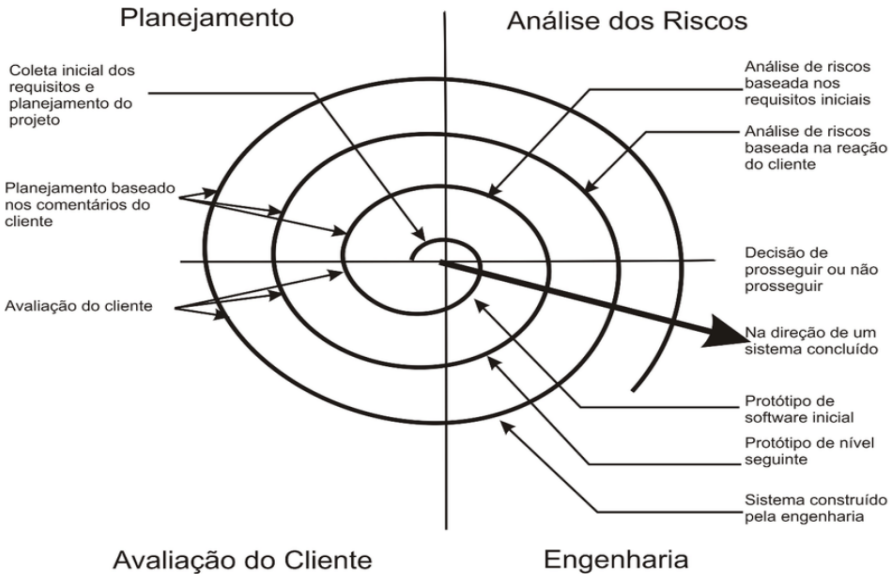
testado e já começa a cobrar solicitando pequenos ajustes e entrega rápida do software. Diante de um quadro assim, muitas vezes, a equipe de desenvolvimento cede e a qualidade final, bem como a manutenibilidade podem ficar comprometidas. Outro ponto negativo é que algumas vezes a equipe de desenvolvimento pode fazer concessões temporárias a fim de colocar o protótipo em funcionamento que acabam permanecendo no software final.
Apesar desses problemas, a prototipação ainda é uma eficiente metodologia de desenvolvimento de software. A fim de se obter sucesso no projeto, tanto cliente quanto desenvolvedor devem chegar a um consenso de que o protótipo servirá apenas para ajudar na definição dos requisitos (PRESSMAN, 2006). Apesar de resolverem muitos dos problemas do desenvolvimento de softwares alguns parâmetros ainda não eram fornecidos pelas metodologias existentes.



Fonte: https://www.researchgate.net/figure/Figura-2-Modelo-de-processo-Prototipacao-Apoiado-por-Padroes-para-prototipos_fig1_224827635

3.2.1.3 Espiral

De acordo com Pressman, 2006, a metodologia espiral foi concebida para englobar as melhores práticas tanto do ciclo de vida clássico quanto da prototipação. Essa metodologia inovou ao trazer também um novo elemento, a análise de riscos. Além disso, foi uma das primeiras metodologias a adotar o conceito de iteração. Sucessivas iterações moldam aos poucos soluções mais completas do software.
Na primeira iteração, os objetivos, alternativas e restrições são definidos e os riscos são identificados e analisados. O cliente avalia o resultado da iteração e baseado nos apontamentos do mesmo a próxima iteração é iniciada. Isso possibilita ao cliente e ao desenvolvedor perceber e reagir a riscos em cada uma das etapas evolutivas. Entretanto a metodologia espiral exige considerável experiência para avaliar os riscos e se baseia nela para obter sucesso. Encara-se que se um grande risco não for detectado indubitavelmente ocorrerão problemas.



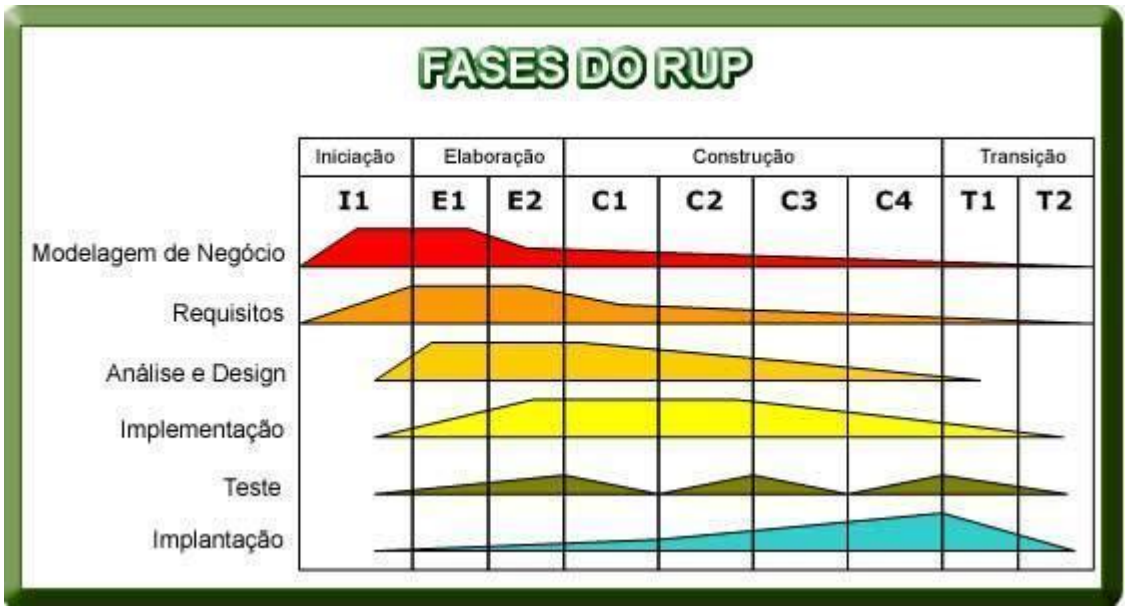
Fonte: https://www.researchgate.net/figure/Figura-21-Modelo-de-processo-de-desenvolvimento-em-espiral-proposto-por-Pressman_fig16_317087042

3.2.1.4 RUP

A metodologia RUP pode ser considerada a metodologia intermediária entre as tradicionalistas e as metodologias ágeis. O RUP organiza o desenvolvimento em 4 fases bem direcionadas, contendo em cada uma delas no mínimo uma iteração, ou seja, um ciclo de vida, são nessas iterações que são mostradas ao cliente o andamento da produção para que ele possa validar e assim liberar a continuação do desenvolvimento. Dessa forma de iterações com que RUP trabalha é que a faz ser considerada uma **metodologia iterativa**.

O principal objetivo do RUP é atender as necessidades dos usuários garantindo uma produção de software de alta qualidade que cumpra um cronograma e um orçamento previsíveis. Assim, o RUP mostra como o sistema será construído na fase de implementação, gerando o modelo do projeto e, opcionalmente, o modelo de análise que é utilizado para garantir a robustez. O RUP define perfeitamente quem é responsável pelo que, como as coisas deverão ser feitas e quando devem ser realizadas, descrevendo todas as metas de desenvolvimento especificamente para que sejam alcançadas.

O RUP organiza o desenvolvimento de software em quatro fases, onde são tratadas questões sobre planejamento, levantamento de requisitos, análise, implementação, teste e implantação do software. Cada fase tem um papel fundamental para que o objetivo seja cumprido, distribuídos entre vários profissionais como o Analista de sistema, Projetista, Projetista de testes, entre outros.



Fonte: https://www.infoescola.com/engenharia-de-software/rup/

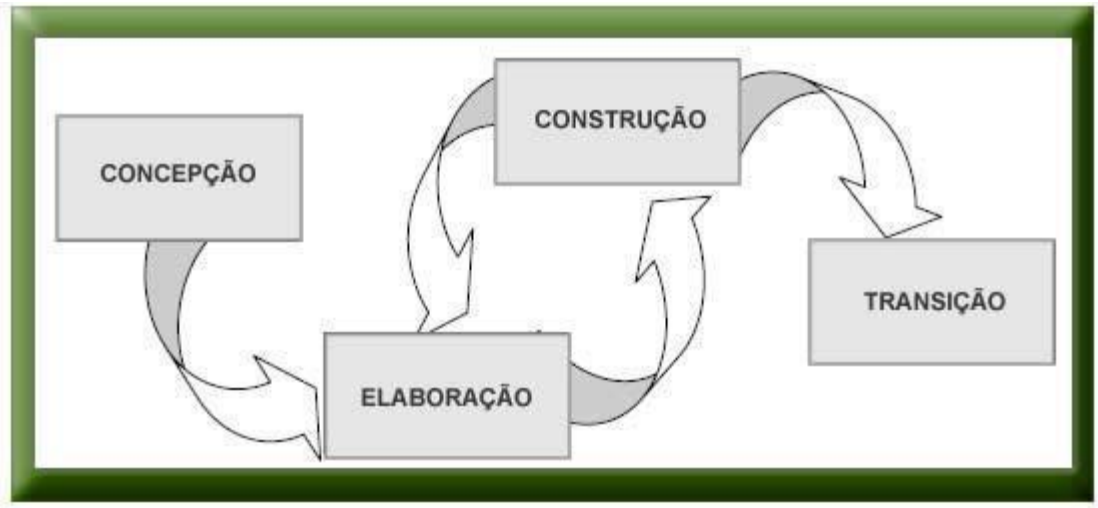
As fases do RUP são:

Fase de Concepção / Iniciação: Esta fase do RUP abrange as tarefas de comunicação com o cliente e planejamento. É feito um plano de projeto avaliando os possíveis riscos, as estimativas de custo e prazos, estabelecendo as prioridades, levantamento dos requisitos do sistema e preliminarmente analisá-lo. Assim, haverá uma anuência das partes interessadas na definição do escopo do projeto, onde são examinados os objetivos para se decidir sobre a continuidade do desenvolvimento.

Fase de Elaboração: Abrange a Modelagem do modelo genérico do processo. O objetivo desta fase é analisar de forma mais detalhada a análise do domínio do problema, revisando os riscos que o projeto pode sofrer e a arquitetura do projeto começa a ter sua forma básica. Indagações como "O plano do projeto é confiável?", "Os custos são admissíveis?" são esclarecidas nesta etapa.

Fase de Construção: Desenvolve ou Adquire os componentes de Software. O principal objetivo desta fase é a construção do sistema de software, com foco no desenvolvimento de componentes e outros recursos do sistema. É na fase de Construção que a maior parte de codificação ocorre.

Fase de Transição: Abrange a entrega do software ao usuário e a fase de testes. O objetivo desta fase é disponibilizar o sistema, tornando-o disponível e compreendido pelo usuário final. As atividades desta fase incluem o treinamento dos usuários finais e a realização de testes da versão beta do sistema visando garantir que ele possua o nível adequado de qualidade.



Fonte: https://www.infoescola.com/engenharia-de-software/rup/

3.2.2 Metodologias Ágeis

As metodologias ágeis vêm sendo utilizadas em muitas equipes e empresas de tecnologia já que com elas é possível ter entregas mais ágeis e assertivas.

Os métodos ágeis surgiram no setor de Tecnologia para resolver problemas comuns a quase toda organização que precisa gerenciar projetos: as etapas de produção muito longas e sem entregas definidas; a falta de clareza e comunicação entre a própria equipe; o desalinhamento entre equipe e cliente e outros. Por isso, eles rapidamente foram também adotados em outros mercados e para projetos além dos de tecnologia, como por exemplo a área de gestão.

Todo projeto precisa ter um início e um fim bem definidos, que é o que realizamos no levantamento de requisitos, mas sua execução pode se estender por períodos de mais de anos – durante os quais muita coisa pode acontecer, havendo a necessidade de reestruturar parte ou todo o projeto. Então, era preciso desenvolver métodos inteligentes e eficientes que conseguissem contornar esses problemas e que pudessem simplificar a forma como os projetos eram executados – gerando impactos positivos em sua finalização mesmo com alterações solicitadas no meio do caminho. Começaram a ser implementados então os métodos ágeis.

Ao contrário dos modelos tradicionais, as metodologias ágeis propõem ciclos de desenvolvimento curtos, com entregas bem definidas e foco na melhoria contínua dos processos e alinhamento da equipe, realizando o levantamento de requisitos e dividindo o todo do projeto em partes, projetando um produto mínimo viável a ser entregue inicialmente e, com o passar do desenvolvimento, esse produto mínimo vai sendo complementado com entregas de mais e mais funcionalidades até chegar ao todo solicitado. Com isso, passou a ser mais simples identificar erros e falhas durante a execução do projeto e as pessoas envolvidas nele ganharam mais flexibilidade e facilidade para fazer adaptações e evitar que determinados problemas afetassem o seu resultado.

O Manifesto Ágil

Como muitas formas de metodologias ágeis começaram a surgir, em 2001 um grupo composto por 17 pessoas se reuniu para debater sobre essas novas abordagens em gerenciamento de projetos e criou o chamado Manifesto Ágil, que, de certa forma, oficializa a existência das metodologias e estabelece princípios que as caracterizam.

A partir desse documento, pode-se dizer que os princípios mais importantes e que orientam a aplicação de um método ágil são:

- **Comunicação:** indivíduos e interação entre eles mais que processos e ferramentas;
- **Praticidade:** Software em funcionamento mais que documentação abrangente;
- **Alinhamento de expectativas e colaboração:** colaboração com o cliente e membros do projeto mais que negociação de contratos;
- **Adaptabilidade e flexibilidade:** responder a mudanças mais que seguir um plano.

Existem muitas vantagens de se trabalhar com metodologias ágeis comparadas as tradicionais e algumas delas são:

- Maior alinhamento entre o time e com os clientes e rápida resolução de possíveis problemas e conflitos;
- Redução de riscos e resultado de alta qualidade;
- Economia de recursos por meio de entregas mais assertivas;
- Agilidade e eficiência nas entregas e na execução do projeto como um todo;
- Flexibilidade para propor alternativas e chegar à melhor solução possível.

Além destes, existem benefícios indiretos que as metodologias ágeis podem trazer, como a melhoria do clima entre áreas e na empresa como um todo; o aumento da credibilidade e confiabilidade da organização no mercado; e muito mais.
Podemos dizer que os conceitos das metodologias ágeis em seu planejamento são:

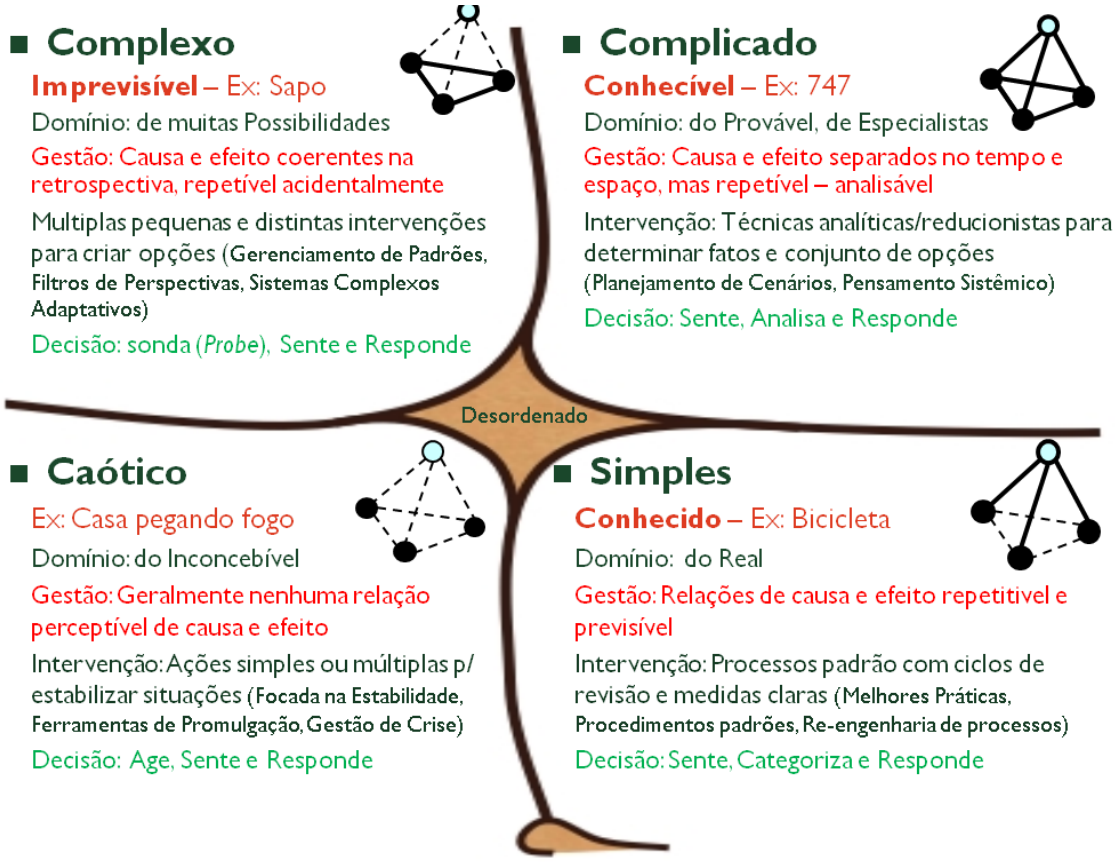
- **Visão:** Antes de iniciar o projeto, a visão do todo. Aborda os problemas principais que o produto deseja resolver. É aconselhável que seja elaborada antes do início do projeto, é aqui que conseguimos as informações (requisitos) necessárias para criar e manter o Backlog do Produto
- **Roadmap:** O plano a ser desenvolvido para alcançar a visão. Aqui é definido informações como quando será disponível o produto, qual o nome da versão a ser publicada, a meta, os requisitos para serem definidas as principais funções; e as métricas para verificar se o objetivo foi de fato atingido;
- **Release:** As validações feitas com o usuário por meio de releases (lançamentos). Aqui teremos uma lista ordenada de tudo aquilo que é necessário para criar o produto, o que estiver fora da lista não será realizado, aqui contêm regras de negócio, requisitos não funcionais, melhorias, correções, testes de arquitetura, etc. Estas releases (chamamos também de backlog), são criadas a partir do documento de visão.
- **Sprint:** Como construiremos os releases de forma incremental;
- **Daily:** Como o progresso será acompanhado diariamente.

Agora iremos abordar algumas das metodologias ágeis do mercado que consideramos as mais utilizadas, mas vale apontar que estas são apenas algumas das inúmeras existentes no mercado e que a melhor a ser utilizada vai depender do projeto a ser desenvolvido, assim quando se inicia um novo projeto é ideal verificar qual das metodologias será mais bem utilizada e irá auxiliar de forma mais assertiva no projeto.

3.2.2.1 Cynefin

O framework Cynefin foi criado em 1999, por Dave Snowden, enquanto ele trabalhava na IBM, para ajudar a companhia a gerenciar seu capital intelectual. Cynefin foi descrito, em detalhes, mais tarde, em um paper que pode ser conferido através do site http://alumni.media.mit.edu/~brooks/storybiz/kurtz.pdf.
A proposta central do framework Cynefin é classificar problemas em cinco domínios diferentes:

1) Claro/Simples;
2) Complicado;
3) Complexo;
4) Caótico e
5) Confuso/Desordem

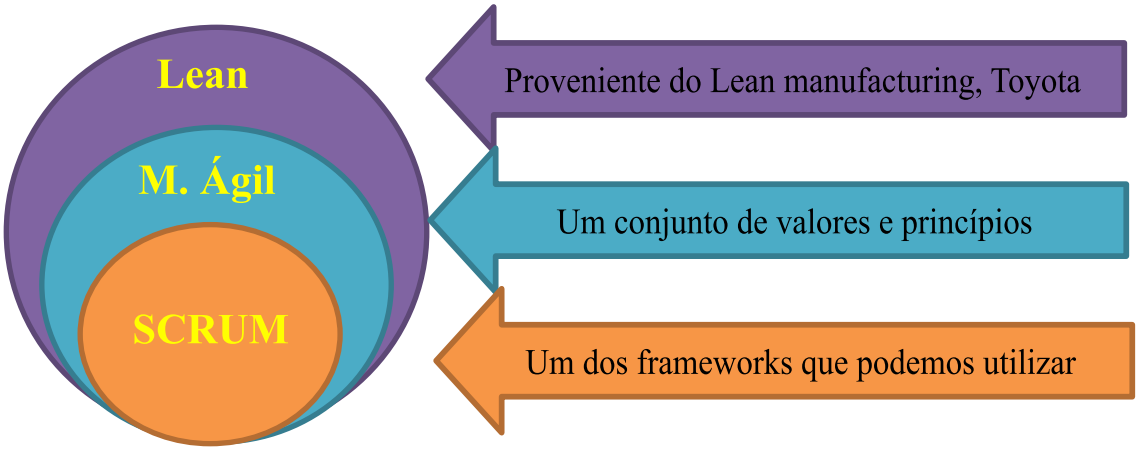


Fonte: https://medium.com/@daviparola/entendendo-o-modelo-cynefin-d%C3%A1-pra-usar-em-qualquer-empresa-inclusive-na-caixa-446ac095ffff

À direita do quadro, encontram-se os contextos denominados de **ordenados** (simples e complicado), onde é possível identificar as causas e efeitos dos eventos que ali ocorrem.
No contexto **simples**, o ambiente é caracterizado pela estabilidade. Os problemas são conhecidos por todos, e a relação entre causa e efeito também. A dificuldade aqui, é categorizar o evento e escolher a resposta apropriada, com base em documentos, procedimentos e manuais de boas práticas.
Nos ambientes onde o contexto é considerado **complicado**, as causas também são conhecidas, mas a dificuldade é encontrar a melhor solução, devido a necessidade de uma análise mais aprofundada. Neste cenário, aparece o trabalho dos especialistas, ajudando na busca pelas melhores respostas para cada situação. Diferente do contexto simples, onde sempre há uma única solução para cada problema, aqui podem existir várias.
Nos contextos chamados de **não ordenados** (complexo e caótico), posicionados à esquerda do quadro, a relação entre causa e efeito não é aparentemente visível, sendo necessário seguir um caminho baseado na intuição e em padrões e metodologias ancorados em tentativa, erro e aprendizado.
O contexto **complexo** é caracterizado pela imprevisibilidade, onde as causas são conhecidas, mas não se conhece os efeitos oriundos das soluções escolhidas. Esses efeitos começam a ser percebidos ao longo do caminho, e por isso, a necessidade do uso de metodologias e padrões que priorizem o aprendizado através da experimentação e a rápida resposta a mudanças.
A palavra que melhor define o contexto **caótico** é sobrevivência. Nesse contexto é impossível determinar qualquer relação entre causa e efeito, simplesmente porque ela muda o tempo todo, e qualquer busca por padrões e respostas corretas é inútil. A ordem aqui é primeiramente agir e tentar sair da situação caótica do ambiente.
No centro do diagrama, no espaço denominado **desordenado**, ficam os ambientes onde não se sabe ou não há consenso sobre o tipo de contexto predominante.

3.2.2.2 Lean

O conceito do Lean foi criado para ser aplicado no “chão de fábrica” na era do Toyotismo, porém com o passar do tempo foi ganhando espaço dentro das metodologias ágeis. Se analisarmos a fundo, podemos dizer que todo manifesto ágil está composto no Lean.
Essa metodologia faz parte do TPS (Toyota Production System) — método usado pela montadora para aumentar a produtividade das suas fábricas — criado nos anos 60 pelo vice-presidente da companhia Taiichi Ohno.



O objetivo da Metodologia Lean é evitar o desperdício, ou seja, utilizar apenas o necessário para realizar determinada atividade. Sendo assim, ela afeta os insumos, o tempo gasto e os níveis de produção. Mas, antes de entender como aplicar esse método na Gestão de Projetos é preciso compreender o que é e como funciona a Metodologia Lean.

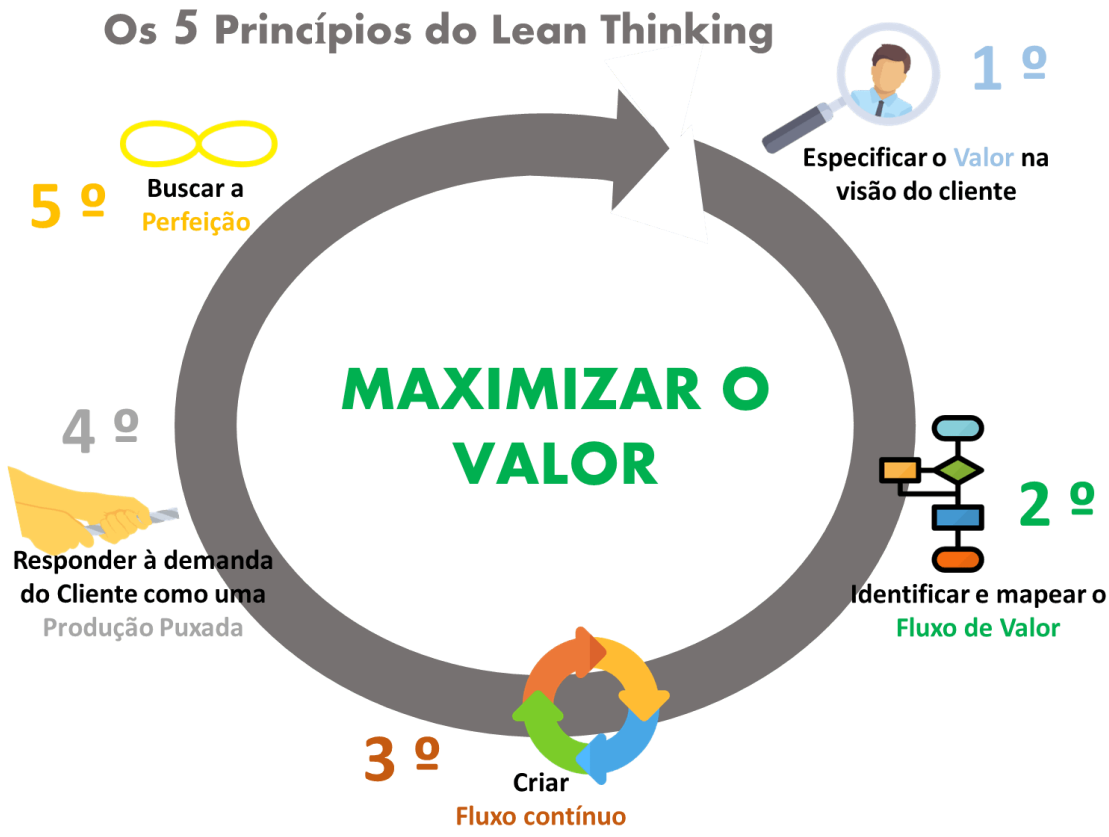
Fonte: https://proj4.me/blog/metodologia-lean

O termo “lean” pode ser entendido como “enxuto”. Isso quer dizer, que a Metodologia Lean é um método que determina o uso de apenas o necessário para a realização de atividades, processos e/ou trabalhos.

Ou seja, essa metodologia prega o fim dos desperdícios, sejam eles de recursos ou de tempo. Para que ela seja aplicada de forma satisfatória é necessário que algumas ações sejam aplicadas ao gerenciamento de processos. São elas:

- Identificar e solucionar aqueles problemas que, constantemente, travam a execução dos projetos;
- Melhorar a comunicação entre os envolvidos, garantido que todos compreendam o fluxo de trabalho e quais são suas responsabilidades dentro dele;
- Priorizar as demandas mais importantes e entregá-las primeiro;
- Oferecer os meios necessários para que a equipe tenha condições de realizar suas atividades.

Como a Metodologia Lean prega o fim do desperdício é comum que ela seja aplicada junto com a Metodologia Ágil, que busca entregar as necessidades do cliente de uma forma mais rápida e eficiente.
As ações se baseiam principalmente na redução de 7 desperdícios: Falta de Qualidade, Espera, Estoques, Movimentação, Transporte, Processos Desnecessários e Superprodução.
A Metodologia Lean nos ajuda a identificar estes desperdícios de uma forma mais clara e nos oferece algumas ferramentas para reduzi-los.
O Pensamento Enxuto é o responsável por guiar a metodologia e ele se baseia em 5 princípios:



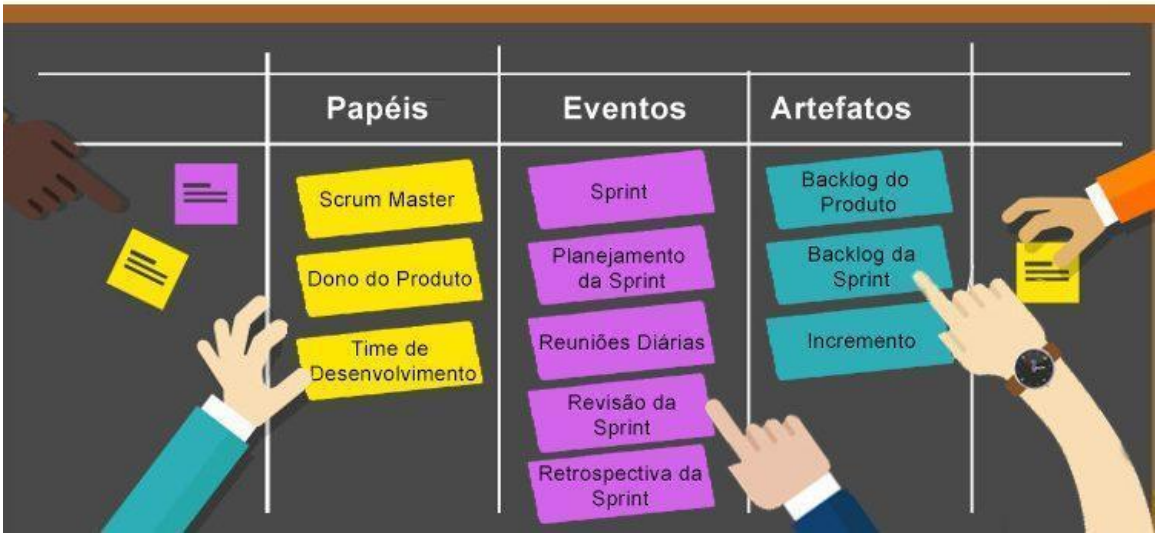
Fonte: https://www.leannasemergencias.com.br/a-comunidade-lean-nas-emergencias/metodologia-lean/

Valor: Este é o principal princípio e talvez o que pareça mais simples, mas exige bastante atenção. Para começar a desenvolver algo precisamos primeiro conhecer, entender e deixar bem definido o que o cliente vê como valor em seu produto ou serviço. Se você entrega algo que não é aquilo que o seu cliente está disposto a pagar, temos um desperdício, pois basicamente aquilo que agrega valor ao cliente é aquilo que ele se dispõe a pagar. Isso

exige estudos frequentes para entender o que agrega valor ao seu cliente e para que você transforme o seu produto ou serviço para melhor adequá-lo.
Fluxo de Valor: Este princípio está bem ligado ao anterior. Agora que você sabe o que é valor para seu cliente, você precisa olhar para seus fluxos e suas respectivas etapas para identificar o que agrega ou não agrega valor ao cliente, pois aquelas que não agregam são consideradas desperdícios e, portanto, devem ser eliminadas.
Fluxo contínuo: Depois que foi identificado o valor e os fluxos foram revistos, é necessário que esse fluxo se torne contínuo, ou seja, sem interrupções, trazendo o conceito de rapidez, menor tempo de processamento e fluidez.
Produção Puxada: Nessa etapa entende-se que deve ser produzido apenas o que o cliente demanda e no momento certo, para reduzir o desperdício de superprodução, ou seja, entregar ao cliente apenas aquilo que é necessário, no tempo certo e com qualidade, sem necessidade de sobras.
Perfeição: Essa é uma etapa que deve ser eterna para que a perfeição sempre seja buscada através da melhoria contínua dos processos, serviços, produtos, pessoas etc. sempre visando a agregação de valor ao cliente.

3.2.2.3 Scrum

Inicialmente o Scrum foi criado para gerenciamento de projetos de fabricação de automóveis e de produtos de consumo, por Takeuchi e Nonaka no artigo “The new product development game”, em janeiro-fevereiro de 1986, pela Universidade de Harvard. Neste artigo, Takeuchi e Nokada notaram que em projetos com equipes pequenas e multidisciplinares produziam melhores resultados, e associaram isto a formação Scrum do Rugby. Em 1995, o Scrum teve sua definição formalizada por Ken Schwaber, que trabalhou para consolidá-lo como método de desenvolvimento de software por todo o mundo, atualmente esta é uma metodologia utilizada não apenas na tecnologia, mas também em diversos segmentos profissionais, inclusive áreas de gestão.
O Scrum não define uma técnica específica para o desenvolvimento de software durante a etapa de implementação, ele se concentra em descrever como os membros da equipe devem trabalhar para produzir um sistema flexível, num ambiente de mudanças constantes. A ideia central do Scrum é que o desenvolvimento de sistemas envolve diversas variáveis (ambientais e técnicas) e elas possuem grande probabilidade de mudar durante a execução do projeto (por exemplo: requisitos, prazos, recursos, tecnologias etc.). Estas características tornam o desenvolvimento do sistema de software uma tarefa complexa e imprevisível, necessitando de um processo flexível e
.



Fonte: https://www.ctrlzeta.com.br/o-que-sao-eventos-por-que-eles-existem-no-scrum/

3.2.2.4 Elementos de Apoio (Artefatos)

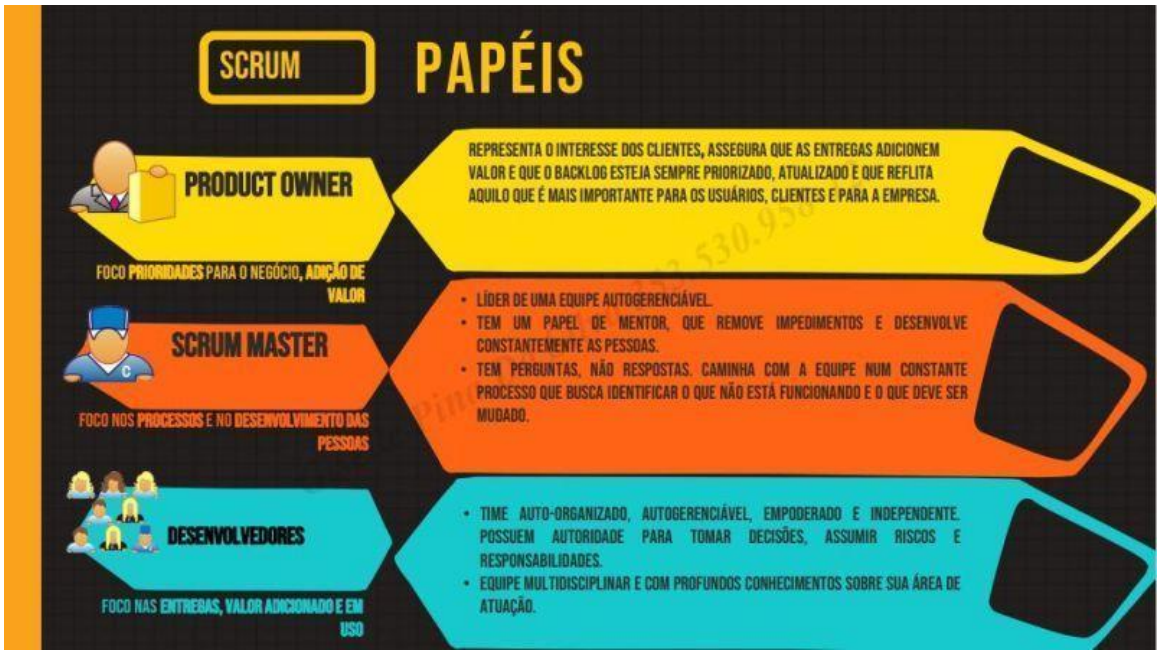
Em sua dissertação de mestrado pela Universidade de São Paulo, com o tema: “Experiências com desenvolvimento ágil”, Filho descreve os elementos de apoio do Scrum, sendo que os únicos elementos que a equipe produz para seguir a práticas de Scrum são cartões com as funcionalidades e gráficos de acompanhamento. Os cartões agrupados formam o Backlog do Produto e outros backlogs. Os gráficos são atualizados frequentemente e devem refletir o estado do projeto. Estes são listados a seguir:

- **Backlog do Produto:** Lista de todos os cartões de funcionalidades que o produto deve possuir e que ainda não foram desenvolvidas;
- **Backlog Selecionado:** Um subconjunto de funcionalidades que o cliente escolheu a partir do backlog do produto para ser implementado na sprint atual e que não pode ser modificado durante a sprint;
- **Backlog da Sprint:** Lista priorizada, obtida a partir da quebra dos cartões do backlog selecionado em tarefas menores;
- **Backlog de Impedimentos:** Lista dos obstáculos identificados pela equipe que não pertencem ao contexto do desenvolvimento;
- **Gráficos de Acompanhamento:** Gráficos que medem a quantidade de trabalho restante são os preferidos em Scrum. É recomendado fazê-los para várias esferas do projeto: para o produto, para a release e para o sprint.

3.2.2.5 Papéis e Responsabilidades

Já na dissertação de mestrado de Franco, com o tema “Um modelo de gerenciamento de projetos baseado nas metodologias ágeis de desenvolvimento de software e nos princípios da produção enxuta”, também pela Universidade de São Paulo, os papéis no Scrum que possuem tarefas e propósitos diferentes durante o processo e suas práticas são apresentados a seguir:

- **Cliente:** participa das tarefas relacionadas à definição da lista de funcionalidade do software sendo desenvolvido ou melhorado, elaborando os requisitos e restrições do produto desejado.
- **Gerente:** é encarregado pela tomada das decisões finais, utilizando as informações visuais disponibilizadas graficamente pelos padrões e convenções a serem seguidas no projeto. Ele também é responsável por acordar, junto aos Clientes, os objetivos e requisitos do projeto.
- **Equipe Scrum:** é a equipe de projeto que possui autoridade de decidir sobre as ações necessárias e de se organizar para poder atingir os objetivos preestabelecidos. A Equipe Scrum é envolvida, por exemplo, na estimativa de esforço, na criação e revisão da lista de funcionalidade do produto, sugerindo obstáculos que precisam ser removidos do projeto.
- **Scrum Master:** é responsável por garantir que o projeto esteja sendo conduzido de acordo com as práticas, valores e regras definidas no Scrum e que o progresso do projeto está de acordo com o desejado pelos Clientes. O Scrum Master interage tanto com a Equipe Scrum, como com os Clientes e o Gerente durante o projeto. Ele também é responsável por remover e alterar qualquer obstáculo ao longo do projeto, para garantir que a equipe trabalhe da forma mais produtiva possível.
- **Responsável pelo Produto:** é oficialmente responsável pelo projeto, gerenciamento, controle e por tornar visível a lista de funcionalidade do produto. Ele é selecionado pelo Scrum Master, Clientes e Gerente. Ele também é responsável por tomar as decisões finais referentes às tarefas necessárias para transformar a lista de funcionalidades no produto, participando na estimativa do esforço de desenvolvimento necessário e é responsável pelo detalhamento das informações referentes à lista de funcionalidade utilizada pela Equipe Scrum.



Fonte: https://www.linkedin.com/in/andre-dentzien-6383b095/?originalSubdomain=br

3.2.2.6 Entregas Contínuas (Eventos)

A metodologia proposta pelo modelo Scrum aplica um sistema de entregas contínuas. Nesta metodologia com os backlogs definidos (que são os requisitos funcionais do sistema), uma sprint programada
(tempo predeterminado no qual será dividido o trabalho para efetuação de uma entrega, tendo como padrão o prazo dentre duas a quatro semanas), reuniões diárias (de 10 minutos para acompanhar se o projeto está de acordo com o planejamento) e, ao final de cada sprint, uma reunião de retrospectiva e planejamento do próximo sprint.

3.2.2.7 Planejando as interações (entregas)

Na metodologia Scrum para se planejar uma interação (uma entrega) deve-se seguir a seguinte sequência de atividades, segundo Araújo e Galina:

- Discutir uma estória;
- Decompor uma estória em tarefas;
- Desenvolvedor deve aceitar a responsabilidade por cada tarefa;
- Após todas as estórias terem sido discutidas e todas as tarefas aceitas, os desenvolvedores individualmente estimam as tarefas aceitas para ter certeza de que eles não estão se comprometendo com algo que não podem cumprir.