

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DAINF - DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ANA FLÁVIA YANAZE MORANOBU
EDUARDO VANDERLEI DOS SANTOS JUNIOR

**UTILIZAÇÃO DE SCRUM E EXPERIÊNCIA DE USUÁRIO
NO DESENVOLVIMENTO DE SOFTWARE EM UMA
APLICAÇÃO PARA HEMOBANCO**

PROPOSTA DE TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2021

ANA FLÁVIA YANAZE MORANOBU
EDUARDO VANDERLEI DOS SANTOS JUNIOR

**UTILIZAÇÃO DE SCRUM E EXPERIÊNCIA DE USUÁRIO
NO DESENVOLVIMENTO DE SOFTWARE EM UMA
APLICAÇÃO PARA HEMOBANCO**

Proposta de Trabalho de Conclusão de Curso apresentado
ao Curso de Bacharelado em Sistemas de Informação
da Universidade Tecnológica Federal do Paraná, como
requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof^a Dra. Maria Cláudia Figueiredo
Pereira Emer
DAINF - Departamento Acadêmico de In-
formática -UTFPR

Coorientador: Prof^a Dra Claudia Bordin Rodrigues da
Silva
DADIN - Departamento Acadêmico de De-
senho Industrial -UTFPR

CURITIBA
2021

SUMÁRIO

1 – INTRODUÇÃO	9
1.1 Objetivo Geral	10
1.2 Objetivos Específicos	10
2 – REVISÃO DE LITERATURA	12
2.1 Metodologias Ágeis	12
2.2 Exemplos de Metodologias Ágeis	14
2.2.1 Extreme Programming (XP)	14
2.2.2 Lean Software Development	16
2.3 Scrum	16
2.3.1 Papéis / Time	17
2.3.1.1 <i>Product Owner</i> (PO)	17
2.3.1.2 <i>Scrum Master</i>	17
2.3.1.3 Time de Desenvolvimento	17
2.3.2 Eventos	18
2.3.2.1 Sprint	18
2.3.2.2 Planejamento da Sprint	18
2.3.2.3 Reunião Diária	18
2.3.2.4 Revisão da Sprint	19
2.3.2.5 Retrospectiva da Sprint	19
2.3.3 Artefatos	20
2.3.3.1 Backlog do Produto	20
2.3.3.2 Backlog da Sprint	20
2.3.3.3 Incremento	20
2.3.3.4 Definição de “Pronto”	20
3 – METODOLOGIA	21
4 – Resultados	22
5 – EXEMPLOS	23
5.1 Exemplo de seção	23
6 – CONSIDERAÇÕES FINAIS	26
Referências	27

1 INTRODUÇÃO

Atendendo à demanda global intensamente crescente de software, várias técnicas de gestão de desenvolvimento de aplicações surgem com (Abrahamsson et al. (2017)) o foco em atender as necessidades do consumidor final (Silva et al. (2011)). Naturalmente com uma maior demanda, novos obstáculos surgiram no terreno da tecnologia tais mudanças resultaram na troca de metodologias mais convencionais como o Cascata (Safwan et al. (2013)), pelas Metodologias Ágeis, que se destacam por - resumidamente - propoem adaptabilidade, entrega do produto final com maior velocidade diante os métodos convencionais e mais entregas durante o desenvolvimento (Melo et al. (2013)). Tal escopo e conceito ágil foi estudado profundamente por (Boehm e Turner (2004)), que identificaram a base do *agile*, *i.e* Métodos Ágeis. Os autores também elencaram quatro elementos: Aplicação, Gestão, Técnica e Pessoas, *i.e* Recursos. Tais elementos segundo os autores convergem as necessidades que fazem o conceito ser executado com êxito. (Boehm e Turner (2003)) (Boehm e Turner (2004)) (Boehm e Turner (2005))

Portanto, o conceito de *agile* pode se manifestar nos mais diferentes métodos para desenvolvimento de software, entre quais o *Scrum* é um dos mais maduros e adotado (Dingsøyr et al. (2012)), sendo um dos poucos métodos sobreviventes no mercado com tantas mudanças (Abrahamsson et al. (2017)). *Scrum* pode ser estruturado, para fins introdutórios, em Eventos, Artefatos e Papéis/Time (Schwaber e Sutherland (2017)), (Cooper e Sommer (2016)). A classe Eventos engloba reuniões para resolver conflitos, realizar planejamentos, retrospectativas e acompanhamento. A Classe de Artefatos engloba os conceitos importantes para o *framework*. E por fim, a classe de Papéis/Time que representa o dono do produto, o *Scrum Master* e o time de desenvolvimento. Tais conceitos serão discutidos a frente neste texto.

Dentro dos conceitos de Metodologias Ágeis para desenvolvimento de software o *Scrum* apresenta uma postura sucinta e coesa (Dingsøyr et al. (2012)). No entanto, mesmo sendo um ótimo guia para o desenvolvimento do produto final (Schwaber e Sutherland (2017)), metodologias ágeis na área de computacional, em sua maioria, tendem em ter seu foco voltado para o caminhar da escrita do código e de funcionalidades em si (*e.g.* (Beck e Gamma (2000))) e deixando de lado um elemento importante que é a Experiência do Usuário (UX) (Ferreira, Sharp e Robinson (2011)).

Segundo Jokela (2012), Usabilidade (9241-210 (2010)) e Boa Experiência do Usuário são fatores centrais para classificar bons produtos e sistemas. Ambas tem como foco o usuário final e sua relação com o produto. Porém (Lárusdóttir, Cajander e Gulliksen (2012)), diz que há duas maiores escalas para mensurar a Usabilidade são elas a Eficiência e a Eficácia (9241-210 (2010)), enquanto a Experiência do Usuário (UX) abrange muito mais que isso. Ela tem como objetivo a satisfação do usuário na sua mais extensa pluralidade:

desde os sentimentos hedônicos presentes antes da apresentação de um produto até as expectativas durante a utilização. (Hassenzahl (2003)), que transpõem o foco voltado a tópicos e métricas da Usabilidade e visa mais os sentimentos e emoções.

No terreno do desenvolvimento computacional, o *Scrum* pode ser considerado um *framework* ágil com uma grande popularidade e procura (Hernández, Kreye e Eppinger (2019)), já a Experiência do Usuário (UX) segue em crescimento dentro deste cenário, sendo abordada em diversas discussões (Law (2011)) dentro do mercado de software que se encontra cada vez mais competitivo. Surge assim a necessidade cada vez maior que os produtos e serviços da área computacional ampliem suas conjunturas.

Apesar dos conceitos de Experiência do Usuário (UX) e *Scrum* serem muitas vezes aplicadas em conjunto no mercado (Kikitamara e Noviyanti (2018)) e ser em especial promissora a união. Não há clareza ou padrões a serem seguidos na hora de aplicá-las (Kuusinen, Mikkonen e Pakarinen (2012)), (Ferreira, Sharp e Robinson (2011)), (Silva et al. (2011)). Isso causa um atrito, incertezas e questionamentos sobre a integração dos dois elementos, e impede clareza e coesão nesse processo, como não conseguir profissionais e usuários devido alguns elementos temporários estabelecidos pelo *Scrum*, como as durações dos ciclos de trabalho (*Sprint*) pode impedir trabalhos como a pesquisas com usuários (Lárusdóttir, Cajander e Gulliksen (2012)).

1.1 Objetivo Geral


Dado o cenário descrito, este trabalho de conclusão de curso visa apontar *Qual a melhor maneira de conciliar Scrum com Experiência de Usuário (UX) no desenvolvimento de uma aplicação.*

Baseando-se na análise de casos, levantamento bibliográfico e simulação de desenvolvimento afim de adquirir um rico estado da arte sobre o objeto de pesquisa, este projeto visa assim desenvolver uma proposta para responder à questão de pesquisa utilizando a classificação em ((KIKITAMARA; NOVIYANTI, 2018)).

1.2 Objetivos Específicos

- Eleger uma maneira de conciliar Scrum e Experiência do Usuário para o desenvolvimento de uma aplicação
- Empregar Scrum durante o desenvolvimento como forma de gerenciamento do projeto, de forma a adequar o método para desenvolvimento
- Empregar Experiência de Usuário no desenvolvimento para entender as necessidades de quem irá utilizar a aplicação
- Simular as etapas do *Scrum* em conjunto com a Experiência do Usuário (UX) no desenvolvimento de uma aplicação real.

- Avaliar a forma de aplicação conjunta do Scrum e Experiência de Usuário utilizada neste estudo, no que diz respeito a prazo de entrega do produto, qualidade e atendimento dos requisitos

O tema, escopo, desenvolvimento e estudo do projeto será baseado em uma aplicação para Hemobancos, contexto a ser retratado de melhor forma durante o desenvolvimento deste projeto. 

2 REVISÃO DE LITERATURA

Essa seção apresentará a revisão de literatura que servirá como base teórica deste trabalho e um estado da arte sobre os temas abordados no trabalho.

2.1 Metodologias Ágeis

Na segunda metade da década de 1990, houve um crescimento significativo dos conceitos de IDD (*Iterative and Incremental Development* (Silva (2019))). Conforme o mercado crescia as necessidades se expandiam, alguns desenvolvedores perceberam que tais metodologias não eram mais ideais para seu contexto. Assim, eles acabaram criando novos princípios para o processo de desenvolvimento, misturado com antigos. Esses novos princípios acabaram trazendo mais próximos o desenvolvimento dos *stakeholders* (partes interessadas e também satisfazendo mais ambos. Com, isso, as pessoas que criaram esses princípios sentiram a necessidade de expandir sua ideias e seu *framework* para o mundo. Ainda assim, essa expansão foi muito restrita e não houve grande crescimento dessas ideias (Alliance ()).

Um grupo de 17 pessoas no ano de 2001 em Utah - EUA, se reuniu para debater seus princípios e ideias, trazendo semelhanças e diferenças e o que de fato funcionava. Houve diversos pontos em que essas pessoas discordaram, porém nos pontos que houve assentimentos, este grupo pôde criar o que hoje é conhecido como Manifesto for Agile Software Development, i.e. *Agile Manifesto* (Alliance ()), (al. (2001)) e, posteriormente, criaram a *Agile Alliance*.

O Manifesto Ágil (al. (2001)), então, tornou-se um modelo para boas práticas de software a serem seguidas, que tem como foco um conjunto de valores discutidos por seus idealizadores. São eles:

- **Indivíduos e iterações acima de processos e ferramentas.**

Processos e ferramentas não são o centro do desenvolvimento de sistemas. É possível criar sistemas sem as ferramentas e os processos, mas não sem pessoas. Pessoas e suas relações devem ser as melhores, pois geram melhores resultados. Isso não significa que ferramentas e processos devem ser deixados de lado. (Silva (2019))

- **Software em funcionamento acima de documentação abrangente.**

Ao se deslocar de um ponto a outro é melhor usar as informações mais atualizadas sobre o caminho e alterar a rota sempre que necessário do que usar o mesmo mapa. Não existe uma receita para desenvolvimento e sim adequações para que o produto saia como esperado. (Silva (2019))

- **Colaboração com o cliente acima de negociação de contratos.**

Muitos projetos dão errado por se manterem focados nas solicitações feitas na hora do

contrato. Demandas do cliente sofrem alterações durante o tempo, sempre devemos trabalhar com os clientes para atender suas necessidades.(Silva (2019))

- **Responder à mudanças antes de seguir o plano**

Planos seguem sequências de passos para atingir um alvo, mas o futuro é incerto o alvo de amanhã pode ser diferente do de hoje. Prestar atenção nas mudanças ajuda que o alvo seja mais próximo possível do que o cliente deseja.

Além disso, o Manifesto contém 12 princípios que refletem os mencionados valores acima. (Ver figura abaixo)

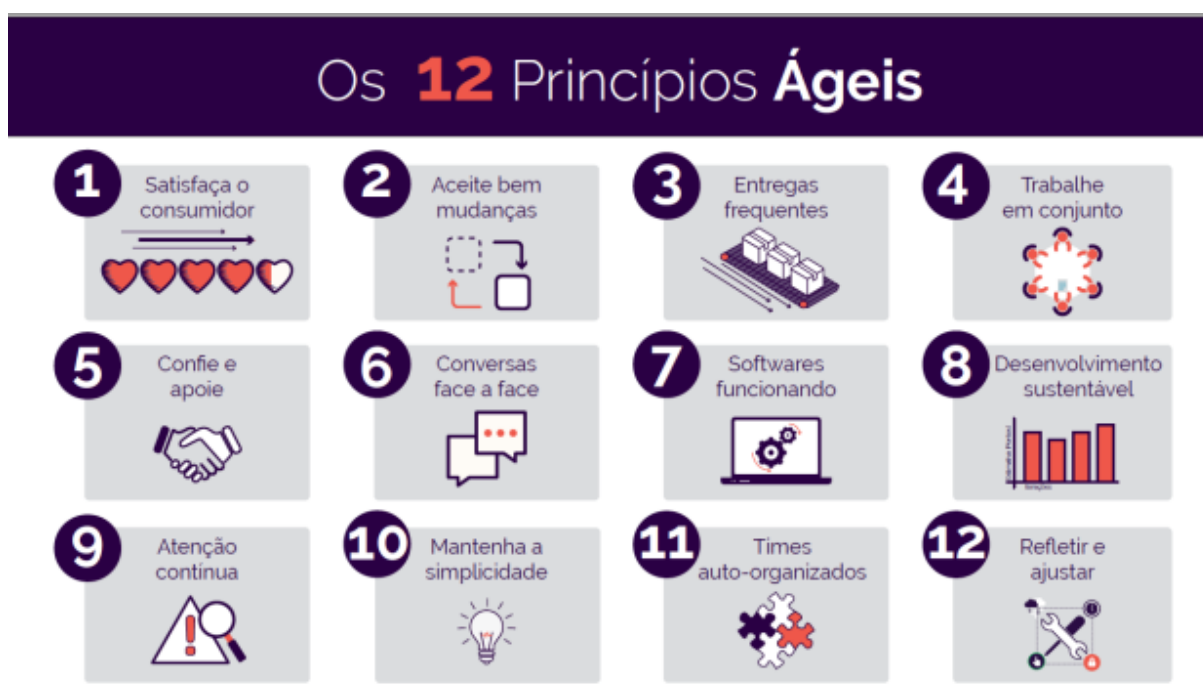


Figura 1 – 12 princípios Ágeis.

Fonte: <https://claudemirgarcia.wordpress.com/2017/04/05/os-12-principios-ageis>

Os 12 princípios são:

- Nossa maior prioridade é satisfazer o cliente através de entrega antecipada e contínua de software.
- Receber mudanças de requisitos, mesmo tardiamente no projeto.
- Entregar software estável com frequência, entre algumas semanas e alguns meses, com preferência à escala de tempo mais curta.
- A equipe de negócios e a de desenvolvimento devem trabalhar juntos diariamente durante o projeto
- Construa projetos em torno de indivíduos motivados.
- O método mais eficiente e eficaz de transmitir informações para e dentro de um desenvolvimento em equipe é conversa pessoal.
- A versão estável do software é a medida prioritária de progresso.

- Processos ágeis promovem o desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem poder manter um ritmo constante indefinidamente.
- Atenção contínua à excelência técnica e um bom design aumentam a agilidade.
- Simplicidade - a arte de maximizar a quantidade de trabalho não feito - é essencial.
- As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizadas.
- Em intervalos regulares, a equipe reflete sobre o que fazer para se tornar mais eficaz, então afina e ajusta seu comportamento de acordo.

2.2 Exemplos de Metodologias Ágeis

2.2.1 Extreme Programming (XP)

O *Extreme Programming* (XP) é baseado em 5 valores definidos por Beck e Gamma (2000), São eles abaixo listados resumidamente:

- **Comunicação**

Comunicação informal e verbal é incentivada entre clientes e fornecedores, possibilitando criar narrativas para transmitir informações importantes. Documentação volumosa é desencorajada para evitar desentendimentos entre as partes envolvidas.

- **Simplicidade**

As simplicidade devem ser considerada na hora de planejar as necessidades imediatas. Se o software precisar ser melhorado futuramente ele deve ser refabricado. Evitando perdas no tempo de desenvolvimentos de soluções que não foram solicitadas.

- **Feedback**

Segundo Beck e Gamma (2000), o *feedback* é oriundo de três partes do software, do cliente e da equipe de desenvolvimento. Testes nos software e testes no software com o cliente, trazem informações sobre o status do desenvolvimento, com essas informações a equipe de desenvolvimento deve levar as cliente atualizações sobre o cronograma e alterações necessárias no software.

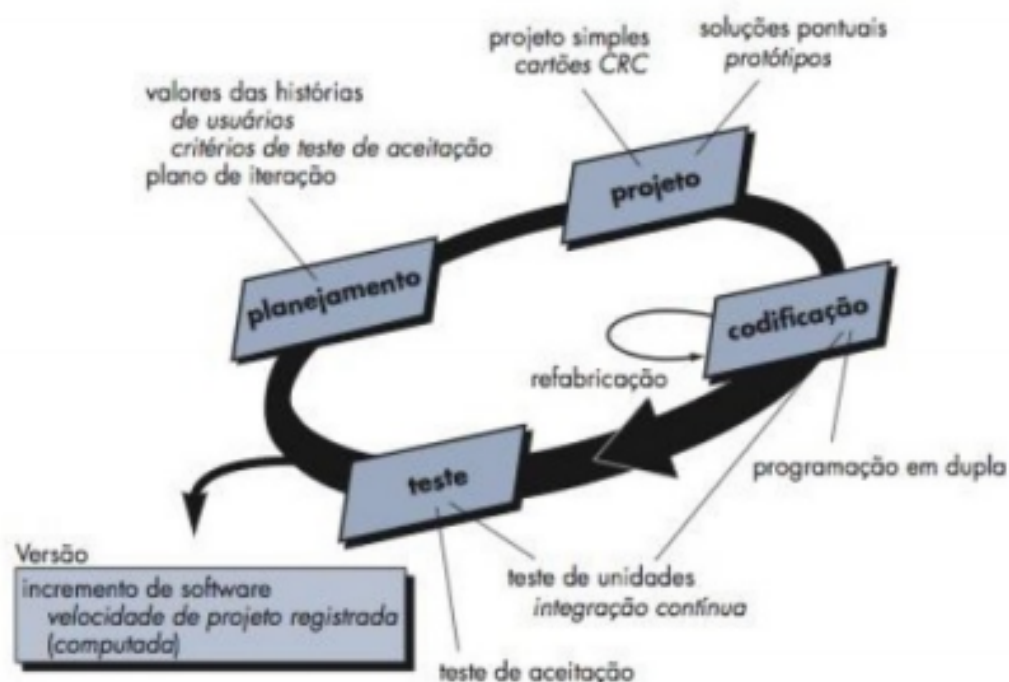
- **Coragem**

Para seguir o que é imposto pela metodologia é requerido da equipe de desenvolvimento coragem. Com o intuito de poupar tempo e esforço ao longo prazo.

- **Respeito**

Por fim, deve-se ter respeito por todos os envolvidos inclusive com o projeto. Conforme o software ganha forma a equipe vai adquirindo respeito pelo trabalho realizado.

Como um conjunto de regras e práticas PRESSMAN (2011) cita planejamento, projeto, codificação e testes. Que se relacionam como indica a imagem abaixo:

Figura 2 – *Extreme Programming (XP)*

Fonte: Pressman 2011

Na fase de planejamento deve-se compreender sobre o ambiente do produto, ouvir as histórias de usuários, avaliar fatores e entender o que é requisitado de funcionalidade pelo cliente.(PRESSMAN (2011))

Cada história é colocada em uma ficha onde é atribuída um valor de prioridade pelos membros da equipe. Após isso cada membro da um tempo em semanas para desenvolver cada história, caso ultrapasse 3 semanas é solicitado que seja quebrado em histórias menores. Esse processo possibilita que seja dado um retorno para o cliente com um possível cronograma do projeto.(PRESSMAN (2011))

Na fase de projeto a simplicidade é encorajada. É dado a cada história um guia de desenvolvimento, conforme vão sendo desenvolvidas funcionalidades extras são retiradas (PRESSMAN (2011)). Quando há um problema de projeção para uma história, é recomendado a criação de um protótipo para amenizar riscos no desenvolvimento real. (PRESSMAN (2011)).

Após cada história ser desenvolvida ela não é diretamente codificada. Elas são submetidas a testes para poderem ser adicionadas na versão final do software. Assim o desenvolvedor foca apenas naquilo que deve ser implementado. Após o código finalizado novamente é testado para correção de pequenos erros, assim evitando grandes problemas próximo a entrega.(PRESSMAN (2011))

Na fase de codificação ocorre a programação em pares. Recomendado que duas pessoas trabalhem juntas para a criação dos códigos, assim garante-se que tudo que é criado vai ser revisado.(PRESSMAN (2011))

2.2.2 Lean Software Development

A metodologia *Lean*, que traduzindo para o Português significa "Enxuta", vem do processo de produção industrial e mais tarde adotada pelo desenvolvimento de sistemas (Poppendieck M.; Cusumano (2012)). A intenção desse método é a entrega de pequenos pacotes de montagem para envio final aos revendedores.

Segundo Petersen K.; Wohlin (2011), a agilidade é inerente ao *Lean*, tendo foco em qualidade, excelência, gestão de pessoas e frequentes entregas ao cliente. E essas características acabam gerando um ciclo de melhoria contínua.

Lean pode ser utilizada em todo processo de desenvolvimento que foca o desperdício mínimo. Podemos usar como exemplo montadoras japonesas de carro que atingiram a liderança do mercado com tempo de menor de engenharia, comparado as rivais europeias e estadunidenses.

As abordagens *Lean* enfatizam a reutilização de recursos principais, fases curtas e simultâneas de produção, reduzindo tempo das entregas (Poppendieck M.; Cusumano (2012)). Unindo isso à princípios ágeis de *feedback* contante e incentivo a alterações. Poppendieck M.; Cusumano (2012) elencam sete princípios para o desenvolvimento enxuto, sendo eles:

1. Otimizar tudo;
2. Eliminar desperdícios;
3. Construir qualidade;
4. Aprender constantemente;
5. Entregar rapidamente;
6. Envolver todos; e
7. Melhorar Continuamente.

2.3 Scrum

Seguindo os valores e princípios citados na seção 2.1, o Scrum é um método ágil criado por dois dos idealizadores do Manifesto, Ken Schwaber e Jeff Sutherland (Beck et al. ()), (Beck e Gamma (2000)). Sendo seu nascimento anterior ao próprio Manifesto, o Scrum é hoje uma das metodologias ágeis mais maduras e mais aplicadas no mercado (Dingsøyr et al. (2012)). Mais de um terço (1/3) dos profissionais de T.I. em países europeus desenvolvidos, como a Islândia, usavam esse processo em seus projetos já em 2009 (Larusdottir, Haraldsdottir e Mikkelsen (2009)), e essa popularidade continua em ascensão no mercado tecnológico (Dingsøyr et al. (2018)). Este estudo irá se basear no Guia Oficial do Scrum, desenvolvido e mantido pelos idealizadores do Scrum, mencionados acima e referenciado em Schwaber e Sutherland (2017). O Scrum, segundo o Guia, é dividido em três grupos de conceitos: Papéis/Time, Eventos e Artefatos. Esses serão detalhados nas subseções a seguir.

2.3.1 Papéis / Time

O time consiste de um *Product Owner*(PO), Scrum Master e o Time de Desenvolvimento. Aqui, o time é auto gerenciável e multi funcional. Feedback também é um item muito importante ao time de Scrum, o que é obtida com a entrega incremental e iterativa do produto pelo time.

2.3.1.1 *Product Owner*(PO)

Também chamado de Dono do Produto, é o responsável por gerenciar o backlog do produto de forma clara, ordenando seus itens e garantindo que esse seja transparente e visível para todos os interessados. Dessa forma, esse papel guia o que necessita ser feito a seguir e maximiza o resultado e valor do produto. A execução desse papel varia de organização para organização, não sendo estabelecidas outras boas práticas para tal função no Guia Oficial do Scrum.

2.3.1.2 *Scrum Master*

Como função principal, o Scrum Master é responsável por colocar em prática de forma efetiva o Scrum como definido no Guia Scrum. Para quem não está dentro do time, ele é responsável por fazer com que haja interação dessas pessoas com o projeto da maneira mais eficiente possível.

Seu trabalho para com o Product Owner garante que todos os objetivos sejam entendidos por todos do projeto, além do escopo do produto e o seu domínio. Ele também deve garantir que o PO saiba gerenciar o backlog de forma clara e efetiva. Já com o Time de Desenvolvimento, o Scrum Master fica responsável por auxiliar a destrinchar quaisquer impedimentos que surjam no decorrer do processo, garantir sua aplicação de forma eficaz, facilitar eventos Scrum, treinar o time para alcançar autogerenciamento e interdisciplinaridade e também facilitar a clara comunicação entre o PO e o Time de Desenvolvimento.

2.3.1.3 Time de Desenvolvimento

São profissionais que realizam o trabalho para entregar, a cada iteração, um incremento do produto considerado "Pronto" para aquele momento. No Scrum, esse time é responsável por organizar e gerenciar seu próprio trabalho, de forma auto organizada para transformar os itens de *Backlog* em Produto. O Scrum não reconhece ou estabelece subtítulos e subtimes dentro do time em seu guia. O Time, para a metodologia, é aquele que, como um todo, independentemente da titularidade de seus integrantes, deve transformar o *Backlog* do Produto em incremento.

Segundo o Guia, o tamanho ideal do Time de Desenvolvimento é "*pequeno o suficiente para se manter ágil e grande o suficiente para completar um trabalho significativo dentro da Sprint*".

2.3.2 Eventos

São chamados Eventos em Scrum certos eventos *time-boxed*, com duração máxima e prescritas, realizadas com pré-determinada regularidade. Foram criadas para diminuir a necessidade de reuniões não previstas. Todo evento contém duração máxima. (Ver figura 3, que ilustra os eventos do *Framework*).

2.3.2.1 Sprint

A Sprint é o coração do Scrum. Ela é um *container time-boxed* de um mês ou menos, com duração fixa, para outros eventos. A partir dela, são realizados planejamentos para o alcance de objetivos, reuniões diárias, estabelecidos os trabalhos feitos de forma flexível pelo Time Scrum, a revisão da Sprint e retrospectiva. Cada Sprint tem uma meta e é considerada um pequeno projeto, que no final o Time chega a um subproduto estável e entregável.

Seu **cancelamento** pode ser realizado única e exclusivamente pelo Product Owner. Quando isso ocorre, qualquer subproduto deve ser revisado e readequado ao gerenciamento do projeto.

2.3.2.2 Planejamento da Sprint

Como mencionada anteriormente, a partir da Sprint é necessária a elaboração de outro evento: o Planejamento da Sprint. Ela é nada mais do que um evento de no máximo oito horas, realizado antes do início da Sprint pelo Time Scrum, no qual duas questões devem ser respondidas: *O que pode ser entregue como resultado do incremento da próxima Sprint?* e *Como o trabalho necessário para entregar o incremento será realizado?*.

A partir do Backlog do Produto, o Time de Desenvolvimento deve prever as funcionalidades a serem desenvolvidas durante esse período. A partir da capacidade do Time, então, são estimados os esforços, previstas funcionalidades e avaliada a viabilidade de um entregável. Neste evento, também, é decidido de forma clara e concisa como o Time de Desenvolvimento pretende trabalhar para completar o objetivo da Sprint e realizar o entregável previsto.

2.3.2.3 Reunião Diária

Também chamada de *Daily Meeting*, o evento acontece internamente entre o Time de Desenvolvimento e tem como propósito informar a todos dentro dele sobre o status

e andamento do projeto, de forma a deixar transparente quaisquer dificuldades, pretensões e trabalho realizado. Sua estrutura é estabelecida pelo time e realizada, como sugere o nome, diariamente. A duração desta reunião deve ser curta (máximo de 15 minutos), de forma a não atrapalhar o desenvolvimento do produto, e sim somar.

2.3.2.4 Revisão da Sprint

Ao final da Sprint, é realizada a sua revisão. Esse evento inspeciona o incremento e adapta o Backlog do Produto caso necessário. Aqui, as partes interessadas e o Time Scrum trabalham para alinhar seus interesses e objetivos. Esta é uma reunião de no máximo 4 horas de duração, para uma Sprint de um mês na qual é exposto quais itens do Backlog foram "Prontos" e quais não foram durante a Sprint, esclarecido dúvidas sobre o incremento, a revisão de problemas durante a Sprint e revisão da linha do tempo, orçamento e outros aspectos do projeto como um todo.

O resultado desse evento é um Backlog do Produto revisado, que servirá de horizonte para definição de itens a serem feitos na próxima Sprint.

2.3.2.5 Retrospectiva da Sprint

Após a Revisão da Sprint e antes do Planjamento da Sprint, em no máximo três horas, o Time Scrum inspeciona seu próprio desempenho para com o projeto, identifica e ordena os principais itens que foram bem e suas potenciais melhorias e, principalmente, cria um plano de ação para sempre melhorar seu desempenho.

SCRUM FRAMEWORK

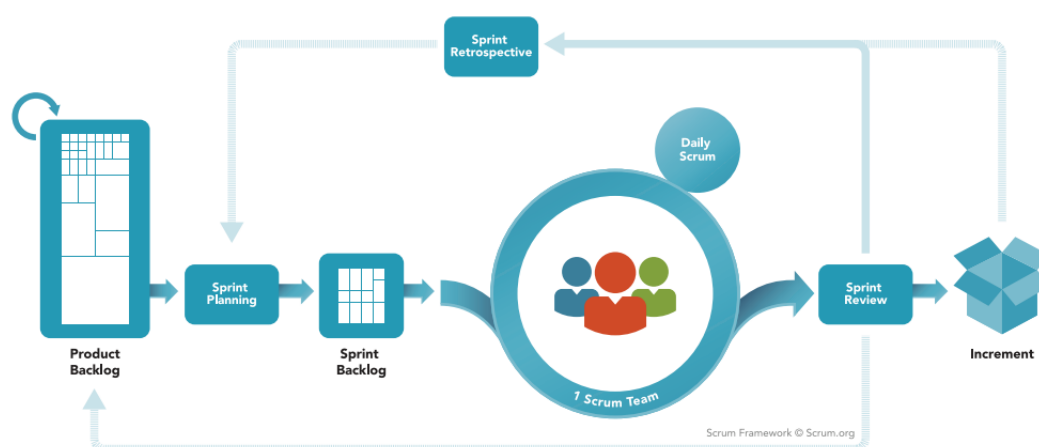


Figura 3 – "The Scrum Framework Poster"
Fonte: *scrum.org*

2.3.3 Artefatos

Os artefatos são informações chaves sobre os elementos do Scrum, para que todos tenham entendimento esclarecido e transparente sobre alguns aspectos já mencionados neste artigo. São eles:

2.3.3.1 Backlog do Produto

De forma simples, é uma lista ordenada de tudo que julga-se necessário pelo Product Owner para a implementação e entrega produto final. Tal artefato pode ser mudado e reorganizado diversas vezes durante o projeto, de forma que o topo da lista sejam coisas prioritárias para o desenvolvimento do produto. O Time de Desenvolvimento e o Scrum Master podem ajudar a apontar aspectos importantes para elaboração do Backlog do Produto, mas apenas o Product Owner julga o que realmente é necessário para tal.

2.3.3.2 Backlog da Sprint

A partir da definição de uma meta para a Sprint, é filtrado itens do Backlog do Produto pelo Time de Desenvolvimento e transformados em Backlog da Sprint. Assim como o artefato anterior, este é flexível e pode ser modificado no decorrer da Sprint, o evento time-boxed que este artefato está contido. Esse Backlog é rastreado inteiramente pela equipe de desenvolvimento, e deve ser adequado às necessidades da equipe de acordo com a meta estabelecida para a Sprint.

2.3.3.3 Incremento

Atendendo de forma transparente ao Backlog da Sprint em execução, o Incremento é um subproduto, um entregável que atende ao objetivo da Sprint. No decorrer do projeto, representa cada vez mais um subproduto perto do objetivo final.

2.3.3.4 Definição de “Pronto”

É necessário que todos tenham o mesmo entendimento do que é considerado “Pronto”, para evitar divergências durante o desenvolvimento. Essa definição garante que um produto ou incremento está pronto para ser deliberado. Geralmente é acordado antes do início da primeira Sprint.