PGDF

PDS-PGDF

Processo de Desenvolvimento de Software

Procuradoria-Geral do Distrito Federal

Procuradora-Geral
Ludmila Lavocat Galvão

Procurador-Geral Adjunto do Contencioso
Idenilson Lima da Silva

Procurador-Geral Adjunto do Consultivo
Hugo de Pontes Cezario

Procurador-Geral Adjunto da Fazenda Distrital
Bruno Paiva da Fonseca

Secretário-Geral
Marcos Gustavo de Sá e Drumond

Procurador-Corregedor
Giullianno Caçula Mendes

Comitê Gestor de Tecnologia da Informação (CGTI/PGDF)

Secretário-Geral - Coordenador
Marcos Gustavo de Sá e Drumond

Subsecretário-Geral de Apoio Técnico, Operacional e Científico
Ricardo Clemente da Costa Júnior

Subsecretária-Geral de Administração
Jordana Cavalcante Barros

Subsecretária-Geral de Tecnologia da Informação
Lorenza D'Onofrio Carneiro

Procuradora-Chefe da Procuradoria Especial de Gestão Estratégica, Estudos e Inovação
Izabela Frota Melo

Procurador Indicado pelo Procurador-Geral Adjunto do Contencioso
Thiago Moisés Elmiro Freitas

Procurador Indicado pelo Procurador-Geral Adjunto do Consultivo
Alexandre Moraes Pereira

Procurador Indicado pelo Procurador-Geral Adjunto da Fazenda Ditrital
Maria Auxiliadora Garcia Durán Alvarez

1.1 Elaboração

Elielson Felipe Crisóstomo Liess - Gerente de Banco de Dados e Qualidade

Jônatas da Silva Conceição - Gerência de Processamento de Dados Operacionais

Kaio Bruno Alves Rabelo - Diretor de Soluções em Tecnologia da Informação

1.2 Revisão da Versão 1.0

Barbara Oliveira Schultz Barbosa - Gerente de Processamento de Dados Operacionais

Bruno Cézar Gomes de Sá e Silva - Diretor de Projetos e Governança em TI

Diego Cesar Bessa - Gerente de Requisitos de Negócio e Desenvolvimento

Riane de Oliveira Torres Santos - Subsecretária-Geral de Tecnologia da Informação

1.3 Revisão da Versão 2.0

Riane de Oliveira Torres Santos - Subsecretária-Geral de Tecnologia da Informação

Kaio Bruno Alves Rabelo - Diretor de Soluções em Tecnologia da Informação

Diego Cesar Bessa - Gerente de Requisitos de Negócio e Desenvolvimento

Eduardo Amaral de Paula – Gerente de Projetos e Processo de Software

Jonathan Michael Pinheiro da Silva – Gerente de Integração e Entrega Contínuas

1.4 Diagramação do Documento

Francisco Antonio Lopes de Farias – Gerência de Projetos e Processo de Software.

Jônatas da Silva Conceição - Gerência de Processamento de Dados Operacionais

Sumário

1.5 Lista de Figuras

PARTE I

Apresentação

A Procuradoria-Geral do Distrito Federal (PGDF) é, nos termos do artigo 110 da Lei Orgânica Distrital, o órgão central do sistema jurídico do Distrito Federal. Exerce, privativamente, a competência de orientação jurídica da Administração direta e indireta, além de representar o ente político judicial e extrajudicialmente, entre outras competências descritas no art. 4º da Lei Complementar nº 395, de 31 de julho de 2001, e suas alterações.

Nessa qualidade, esta Casa de Advocacia Pública deve estar alinhada aos interesses do Estado, especialmente no tocante à defesa do GDF, em ambiente tecnológico compatível com a eficiência pretendida pelo Estado e pela sociedade e exigida pelos órgãos do Poder Judiciário nos níveis federal e distrital.

Para o desenvolvimento de seu mister constitucional, é indispensável a incorporação de um planejamento adequado da Tecnologia da Informação (TI), com a finalidade de preparar a instituição para a inexorável informatização da Justiça, bem como para o adequado funcionamento gerencial desta Casa.

Outrossim, as organizações públicas têm passado por um grande processo de modernização para atender às demandas atuais e emergentes da sociedade. A fim de dar suporte a este movimento, é necessário investir em soluções de software que possam garantir a sustentação e o aperfeiçoamento das atividades dos órgãos públicos. A adoção de soluções modernas

e com qualidade depende de um processo estabelecido que possa assegurar que o software a ser desenvolvido atenda às necessidades da instituição. Para que isso ocorra é necessária a definição de todos os processos envolvidos na contratação, no desenvolvimento interno ou na produção colaborativa de uma solução de software.

Diante deste grande desafio, a Subsecretaria-Geral de Tecnologia da Informação da Procuradoria-Geral do Distrito Federal (SUTIC) propõe um processo desenvolvimento de software com a finalidade de estabelecer marcos metodológicos para as atividades criativas e intersubjetivas que perfazem a construção de um software. Construído de forma colaborativa, este trabalho não propõe procedimentos fixos, mas um arcabouço conceitual a partir do qual as áreas técnicas e negociais poderão se amparar na superação das diversas dificuldades práticas.

A opção por um processo ágil, mesmo no ambiente da administração pública, dificilmente poderia hoje se apresentar como inovadora – e isto não apenas porque já se passaram duas décadas desde que o primeiro manifesto ágil veio ao palco. Amplamente predominante no mercado privado, o ágil já vem sendo utilizado na prática por diversas entidades públicas e, além disto, recebeu seu guia formal com a publicação pelo Sistema de Administração dos Recursos de Tecnologia da Informação - SISP, em 2015, do "Guia de Projetos de Software com Práticas de Métodos Ágeis para o SISP", de que muito nos valemos na construção deste documento.

Importante ressaltar que a difusão da agilidade em contextos tão diversos, ademais, permitiu ao ambiente acadêmico o estudo sistemático de suas forças e fraquezas,

como pode ser conhecido na ampla literatura que a temática já estimulou. Assim, procuramos também ouvir algo deste debate que pudesse nos ajudar a não apenas “aplicar” a agilidade, mas, muito antes, criar uma metodologia ágil adaptada à realidade específica da PGDF. O trabalho que se segue também pode ser visto como um conjunto de hipóteses e sugestões práticas, ou seja, um conjunto de ferramentas para uso de todos os envolvidos no processo de criação de software.

Introdução

O Processo de Desenvolvimento de Software (PDS) é tanto um guia para planejamento de demandas quanto um guia para desenvolvimento.

É um suposto da agilidade que um software nasce de forma incremental: por meio da exploração incessante de um dado domínio, problemas vão sendo definidos e soluções testadas. Como consequência, inexiste separação discreta entre planejar e executar. São atividades correlatas para as quais o PDS propõe um conjunto de regras, padrões, tarefas e modelos com o objetivo de alcançar um software seguro e de qualidade, feito com produtividade e criatividade.

Dessa forma, o PDS trata de projetos de software, um serviço abrangente de tecnologia da informação, que inclui desde a mais básica das atividades de desenvolvimento (a codificação) até o planejamento da arquitetura e dos recursos necessários ao funcionamento adequado do sistema. O escopo deste Processo, portanto, não menos que seu conteúdo, é sensível às novas necessidades digitais e à evolução concreta dos projetos de TI.

1.6 Principais Objetivos

- Desenvolver e manter um catálogo de soluções funcionais, seguras e valorosas;
- Garantir que os projetos de software alcancem soluções correspondentes ao problema definido;
- Atender os requisitos de negócio e de segurança do projeto;
- Cumprir prazos sem comprometer a qualidade;
- Planejar e gerenciar os projetos de software de forma ordenada, clara e racional;
- Documentar os projetos segundo critérios de rastreabilidade, atualização e utilidade;
- Utilizar com economicidade, eficiência e eficácia o repertório de soluções de software;
- Auditar e testar permanentemente o catálogo de soluções e seus incrementos;
- Garantir a adequação das soluções de software com a arquitetura corporativa;
- Fomentar a atuação colaborativa, participativa e criativa dos envolvidos no processo de desenvolvimento.

1.7 Documentos de Referência

O PDS segue a estratégia de governança de TI da PGDF, em conformidade com o Plano Diretor de Tecnologia da Informação 2021-2023 (PDTI). Inspira-se, em sua formatação, no "Guia de Projetos de Software com Práticas de Métodos Ágeis para o SISP", versão 1.0 (2015) e no "PDS-BC Ágil - Processo Ágil

de Desenvolvimento de Software do Banco Central", versão 3.0.0 (2020).

1.8 Abrangência e Período

O PDS se aplica a todas as unidades administrativas envolvidas em projetos de software na PGDF.

Deve ser observado por todos os colaboradores da PGDF, quaisquer que sejam seus vínculos formais, bem como por prestadores de serviço em razão de contrato administrativo. O Processo pode, contudo, conforme avaliação da SUTIC, ser adaptado às necessidades específicas de cada projeto.

O PDS deve ser revisto, de ofício, após 2 (dois) anos de sua publicação, ou a qualquer tempo, caso haja necessidade de adequação tecnológica e considerando, também, a harmonização com o ambiente corporativo da PGDF.

Metodologia

Para o desenvolvimento do PDS, foi formado um grupo de estudos na Diretoria de Soluções (SUTIC/DISOL) e suas gerências especializadas (GEDEN, GEPOD e GEBAN), incluindo a Gerência de Planejamento e Informação da SUTIC. Este grupo, com o apoio da Diretoria de Gestão de Projetos e Governança em TI (DIGOV/SUTIC) e supervisão da Subsecretaria-Geral de Tecnologia da Informação, analisou as metodologias de desenvolvimento de sistemas em uso no mercado e na Administração Pública, sem descuidar de insumos científicos que as publicações acadêmicas oferecem. O presente documento resume e formaliza as conclusões destes estudos.

Projeto de Software

Como define o Guia Ágil SISP (p. 13), um projeto de software é "um serviço disponibilizado pela Área de Tecnologia da Informação para atender várias necessidades do órgão". *Grosso modo,* os projetos se dividem em "novo desenvolvimento", "customização e implantação" e "manutenção evolutiva". Esta diferença responde à pergunta sobre o que fazer quando se identifica que uma necessidade de negócio demanda como solução um software.

Uma vez delimitadas as funcionalidades mais básicas que podem fazer frente a esta necessidade, há, pelo menos, três caminhos: (1) ou desenvolvemos, de modo integral, um novo sistema que as implemente; (2) ou implantamos e customizamos um sistema já existente, mas não internalizado, que já as tenha implementado; (3) ou as desenvolvemos num sistema já implementado no ambiente interno.

Terceirização dos Serviços de TI

Está entre os Princípios e Diretrizes, item PD21, do PDTI, "ampliar a capacidade operacional da área de TI, através da terceirização de tarefas executivas e das parcerias com órgãos externos". Ciente disto, este PDS se preocupou em conceber um processo capaz de integrar atores externos, que atuem como terceirizados em atividades de desenvolvimento e manutenção de sistemas.

Envolvidos na Construção de Projetos de Software

Atividade multifacetada e complexa, a execução de projetos de software se estrutura ao redor de uma díade básica entre um requisitante (o "Negócio") e uma equipe de desenvolvimento (a "TI"), ou seja, alguém que precisa de um software e alguém que o provê.

No ambiente da Administração Pública, quando toma parte um ator terceirizado, esta díade se torna uma tétrade, com o envolvimento de uma área administrativa, que zela pela correta execução de um contrato público, e de uma contratada, responsável por atividades específicas dentro do projeto de software:

- **Área Requisitante do Projeto de Software:** unidade do órgão ou entidade que demande a construção de um projeto de software. É responsável pela definição dos requisitos, priorização de funcionalidades, homologação negocial e apoio constante aos projetos de software;
- **Área de Tecnologia da Informação (SUTIC):** unidade setorial responsável por gerir a Tecnologia da Informação do órgão ou entidade (IN04 SLTI/MP, 2014).
    - **Unidade Setorial de Sistemas (DISOL)**: setor de TI responsável pelo desenvolvimento, medição e avaliação de qualidade de software, principalmente no aspecto técnico;
    - **Unidade Setorial de Infraestrutura de TI (DISEG):** setor responsável pela infraestrutura de TI. Cria e suporta os ambientes de TI que compõem o parque tecnológico da casa.

- **Contratada de Desenvolvimento de Software:** empresa contratada pelo órgão, responsável pelas atividades técnicas de desenvolvimento de software, tais como análise, projeto, implementação, testes e implantação;
- **Área Administrativa:** unidade setorial e seccional de Serviços Gerais – SUAG – com competência para planejar, coordenar, supervisionar e executar as atividades relacionadas aos processos de contratação e procedimentos administrativos de contratos (IN04 SLTI/MP, 2014). Apoia na gestão e fiscalização do contrato de desenvolvimento de software com práticas de métodos ágeis, sugerido neste guia

Escopo do PDS

Este PDS abrange todas as etapas tradicionalmente delimitadas na engenharia de software, desde o levantamento dos requisitos de uma necessidade de negócio (o problema a ser solucionado), passando pelo desenvolvimento e teste até a sustentação de uma solução funcional e capaz de suprir uma necessidade final.

Incontornável o pressuposto, contudo, de que um software possui, quando comparado a outras áreas da engenharia, certas qualidades peculiares, que tornam indesejável uma abordagem estritamente prescritiva. Em outras palavras, não há um método universal e necessário, aplicável a todo e qualquer projeto de software. O que há na literatura mais se assemelha a um conjunto de boas práticas, que nada mais são que um reservatório de modelos organizacionais capazes de orientar os atores imersos na construção de soluções num ambiente tecnológico volátil.

Neste sentido, apresentamos um Processo de Desenvolvimento de Software capaz de atuar como um guia na construção de uma cultura ágil na PGDF, conciliando contexto organizacional e diretrizes estratégicas da PGDF com aquilo que preconiza a literatura sobre metodologias ágeis. Em face do desafio concreto de cada projeto devem os atores envolvidos definir as especificidades do processo de trabalho.

PARTE II

Processo de Desenvolvimento de Software – PDS

1.9 Princípios e Valores do Ágil

Um processo de desenvolvimento ágil busca um pragmatismo orientado a valores e princípios. Mais que mera adoção do Manifesto Ágil (Agile Manifest, 2001), é o caso de se construir uma metodologia pautada no problema, específica ao objeto, capaz de trazer à luz soluções transformadoras. Estimar indivíduos e suas interações, software operativo, colaboração com o cliente e responsividade à mudança são valores que têm demonstrado, tanto na vida prática quanto na literatura especializada, um alto grau de sucesso na busca por qualidade no desenvolvimento de software. Assim, o processo aqui proposto é enformado por estas ideias, sem descuidar das especificidades do setor público.

Os projetos de software no setor público possuem um terceiro interessado, verdadeiro titular do *interesse público*, que, mesmo quando não se pode fazer ouvir diretamente, está presente na forma dos comandos legais e normativos que

determinam obrigações de fazer ou não fazer. Um processo ágil com *compliance*, portanto, é uma frase-síntese do que aqui se persegue.

A agilidade talvez melhor se defina com recurso ao campo semântico da leveza que da velocidade. É dizer, processo ágil, quando se ressalta a própria metodologia em seu modo de fazer. É aquele processo que deixa a cargo dos atores apenas aquele peso estritamente necessário para que cada tarefa seja bem-sucedida. Por que especificar nas minúcias um requisito que não será desenvolvido tão cedo? O peso desta especificação, nota-se, não é apenas o da elaboração do próprio documento: é também o de todas as decisões que, tomadas de modo prematuro, numa fase do projeto em que os atores talvez não tenham todas as informações necessárias, precisarão ser carregadas até o fim.

Tomar a decisão necessária no momento correto, deixando para a etapa seguinte aquelas decisões sobre as quais podemos colher mais informações, conhecendo melhor as consequências. O processo ágil é fundamentalmente sobre isto: uma estruturação das tomadas de decisão que, gradualmente, levarão ao sucesso de um projeto.

Deste modo, o processo aqui apresentado está organizado ao redor de três grandes eixos: os atores; os fluxos de interação e suas cerimônias; e melhores técnicas e práticas. Quais as etapas de um processo de desenvolvimento de software? Quem são os atores envolvidos? Qual o papel deles em cada etapa? Que decisões devem ser tomadas e em que momento? Quais as melhores técnicas para explicitação das decisões? São algumas das perguntas que responderemos a seguir.

1.10 Os atores

Um projeto de software requer, de modo nuclear, um requisitante, que precisa de um sistema, e um desenvolvedor, capaz de criá-lo. É cara à Agilidade a intuição de que os indivíduos e suas interações são mais importantes que processos e artefatos, razão pela qual convém não seguir pelo caminho de uma complexificação de papéis e responsabilidades. Não vamos nos afastar deste esboço basilar: o software surge do diálogo constante entre alguém que possui um problema, *o negócio*, e alguém disposto a auxiliar na construção da solução, *a TI*.

Figura 1 - Atores do projeto de software

Naturalmente, a diversidade das atividades de negócio e das tecnologias impõe que uma maior quantidade de atores se engaje nos projetos de software. O grande desafio deste PDS é precisamente reduzir o ruído na comunicação que surge da enorme distância que pode se interpor entre, de um lado, o usuário que demanda uma funcionalidade de sistema e, na outra ponta, o desenvolvedor que a codifica.

E aqui o *Scrum*[1] propõe um arcabouço organizacional que aparece

Figura 2 - Scrum

[1] Os termos técnicos destacados no texto estão definidos no glossário anexo.

como solução possível. Leve e pouco prescritivo, seu nome deriva de uma jogada do rugby onde os jogadores se abraçam e se movem em sincronia com o objetivo de repor a bola em jogo. Não apenas a analogia com a jogada reforça a centralidade do trabalho em equipe, como também a cultura do rugby inspirou alguns dos traços definidores do framework: times multidisciplinares e auto-organizados atuando em busca de uma meta compartilhada. A equipe de projeto do S*crum* (figura 3) acrescenta um novo ator, *o scrum master*, que, assim como o capitão no time de rugby, é responsável por nutrir um ambiente de projeto capaz de gerar valor por meio de soluções adaptativas para problemas complexos (SCRUM GUIDE, 2021).

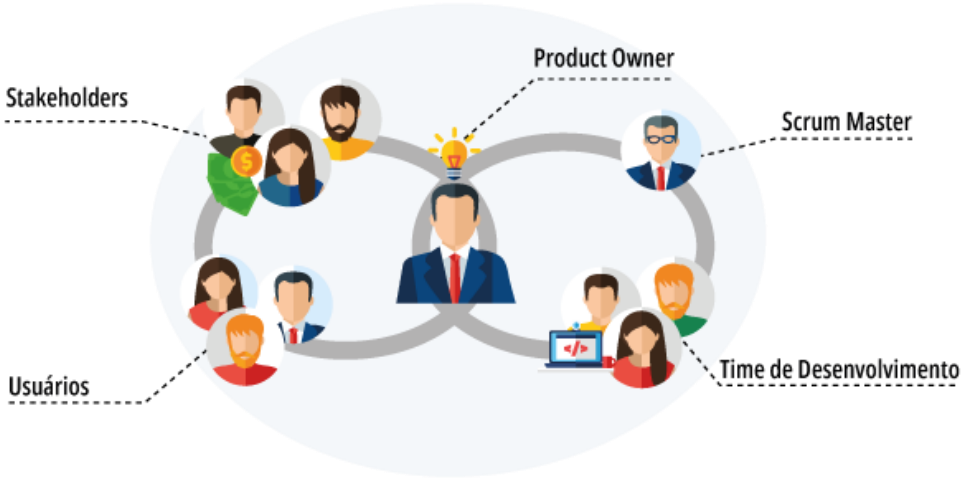


Figura 3 - Equipe de projeto segundo o Scrum

O *Scrum* é intencionalmente simples: o *Product Owner* (PO), representante do negócio, quebra um problema complexo em tarefas menores e ordenadas num *backlog* do produto; o time de desenvolvimento transforma, durante uma *sprint*, cada um dos itens do *backlog* num incremento de software; todos os envolvidos no projeto inspecionam os resultados alcançados e os ajustam para a próxima *sprint;* repete-se o ciclo (figura 4). Os atores são funcionais a este ciclo incremental de projeto, que

possui, ademais, algumas cerimônias basilares que serão explicitadas no tópico Atividades.

Figura 4 - Ciclo Scrum

Dentro desse contexto, o Processo da PGDF precisa diferenciar, contudo, o desenvolvimento interno, onde todos os atores são internos à organização, do desenvolvimento com terceirização, onde as etapas de execução ficam a cargo de um ator externo, contratado para este fim. Neste segundo cenário, a equipe de projeto requer um novo ator, o líder técnico, responsável por auxiliar o PO na busca pela consistência tecnológica da nova solução. Nos projetos exclusivamente internos, o líder técnico poderá assumir parte das atribuições do *scrum master,* conforme necessidade específica do projeto.

Os quadros abaixo descrevem os papéis ou perfis dos responsáveis por atividades. Um papel é responsável por uma ou mais atividades e algumas atividades possuem mais de um responsável. Os papéis podem ser exclusivamente da PGDF ou de uma Empresa Contratada, em caso de terceirização. Os papéis

e suas atividades estão definidos segundo o *Scrum* e a Instrução Normativa 04 do MP, consideradas as especificidades da PGDF.

Papel
Product Owner e Fiscal Requisitante do Contrato de Desenvolvimento de Software.
Descrição
Sua responsabilidade primeira é maximizar o valor do produto, garantindo que exista convergência entre os objetivos de negócio e os objetivos de projeto. Ele é o representante de toda comunidade de usuários e interessados no projeto. Para o bom andamento e continuidade dos projetos, deve o PO designar um substituto para eventuais afastamentos e impedimentos.
Tarefas e Atribuições
• Construir a Visão do Produto; • Planejar o Roadmap; • Elaborar o Backlog do Produto; • Escrever as histórias de usuário que expressam os itens do Backlog; • Participar das reuniões de refinamento, onde as histórias de usuário serão transformadas em critérios de aceitação, auxiliando o time de desenvolvimento a compreender as necessidades de negócio; • Esclarecer as dúvidas do time de desenvolvimento; • Facilitar a comunicação entre usuários e o time Scrum; • Gerenciar o Backlog do produto, o que inclui a criação e comunicação dos itens, inclusive seu ordenamento (priorização); • Homologar os incrementos de software entregues. • Planejar a release; • Validar os incrementos de software; • Homologar o release; • Avaliar, aceitar ou rejeitar serviços de Ordem de Serviço (OS); • Preparar e realizar treinamentos.
Lotação
Área Requisitante do Projeto

Papel
Gestor do Contrato de Desenvolvimento de Software

Descrição
Servidor com capacidade gerencial, técnica e operacional relacionada ao objeto da contratação.
Tarefas e atribuições
• Abrir Ordem de Serviço; • Tratar faturamento de Ordem de Serviço; • Monitorar e controlar obrigações advindas de cláusulas contratuais; • Fechar Ordem de Serviço; • Relatórios de conformidade.
Lotação
Servidor da Área Requisitante do Projeto ou da Área de TI

Papel
Fiscal Técnico do Contrato de Desenvolvimento de Software
Descrição
Servidor com capacidade gerencial, técnica e operacional relacionada ao objeto da contratação
Tarefas e atribuições
• Avaliar, aceitar ou rejeitar serviços de OSs; • Receber serviços de OSs.
Lotação
Servidor da Área de TI

Papel:
Fiscal Administrativo do Contrato de Desenvolvimento de Software
Descrição:
Servidor com capacidade administrativa relacionada ao objeto da contratação.
Tarefas e atribuições:
• Tratar faturamento de Ordem de Serviço; • Monitorar e controlar obrigações advindas de cláusulas contratuais.
Lotação:
Servidor da Área Administrativa

Papel

Líder técnico
Descrição
É o representante da Subsecretaria de Tecnologia da Informação nos projetos de software, responsável por facilitar a interação entre o time de desenvolvimento e o ambiente tecnológico-organizacional da PGDF
Tarefas e atribuições
• Perseguir a implementação, a nível de projeto, do processo de qualidade de software da PGDF; • Avaliar a consistência das soluções em desenvolvimento no interior do ambiente arquitetural da PGDF; • Zelar pelas boas práticas de segurança da informação pertinentes ao projeto; • Facilitar a obtenção de informações requisitadas pela área de execução contratual; • Auxiliar na perseguição do cronograma e objetivos de projeto; • Atuar na remoção de impedimentos técnicos ao projeto, mantendo atualização constante do mapa de riscos; • Buscar a aderência dos processos de trabalho à gestão estratégica de projetos da PGDF, inclusive quanto à atualização da documentação de projeto.
Lotação
Servidor da Área de TI

Papel:
Analista de Qualidade
Descrição:
Servidor responsável por monitorar e auditar o processo de qualidade.
Tarefas e atribuições:
• Padronizar processos de gestão de testes e de requisitos; • Definir padrões técnico-arquiteturais; • Monitorar e auditar a execução dos processos de desenvolvimento.
Lotação:
Servidor da Área de TI

Papel
Analista de Métricas
Descrição
Servidor responsável por executar ações de aferição de sistemas.
Tarefas e atribuições
• Executar ações de aferição dos incrementos de software segundo os processos de qualidade definidos.
Lotação
Servidor da Área de TI

Papel
Analista de Infraestrutura de TI (Servidor da Área de TI)
Descrição
Servidor responsável pela infraestrutura de TI
Tarefas
• Preparar e manter ambientes de TI; • Implantar Software.
Lotação
Servidor da Área de TI

Papel
Scrum Master
Descrição:
"Servo-mestre" do time *Scrum*, o papel deste ator é auxiliar o time e a organização na busca pela efetividade do projeto.
Tarefas e atribuições:

- Auxiliar o time Scrum na remoção de impedimentos ao progresso, na manutenção do foco no valor dos incrementos que atendam à definição de "Pronto" e na adesão às cerimônias e melhores práticas do Scrum;
- Auxiliar o Product Owner na definição do objetivo do produto e no gerenciamento do backlog do Produto, inclusive auxiliando na escrita de histórias de usuário;
- Ajudar no planejamento do produto e na comunicação entre os envolvidos no projeto;
- Auxiliar a organização na adoção da plataforma Scrum e na difusão da cultura ágil;
- Realizar Reuniões de Planejamento da Sprint;
- Executar a Sprint;
- Realizar Reuniões de Demonstração da Sprint;
- Realizar Reuniões de Retrospectiva da Sprint;
- Atualizar Gráfico de Burndown.

Lotação:

Servidor da Área de TI ou da Contratada

Papel
Equipe de Desenvolvimento
Descrição
São os atores dedicados a criar qualquer aspecto de um incremento utilizável de software.
Tarefas e atribuições:

- Planejar a Sprint;
- Perseguir a qualidade da solução, como definida no conceito de "Pronto";
- Adaptar-se diariamente em busca do objetivo da Sprint;
- Manter um ambiente profissional de responsabilidade mútua;
- Desenvolver a solução;
- Realizar Reuniões de Planejamento da Sprint;
- Executar a Sprint;
- Realizar Reuniões de Demonstração da Sprint;
- Realizar Reuniões de Retrospectiva da Sprint;
- Corrigir não conformidades da Ordem de Serviço.

Lotação:

- Servidor da Área de TI
- Contratada

Papel
Stakeholders
Descrição:
São aqueles atores que possuem poder de decisão no direcionamento do projeto, mas não estão diretamente envolvidos na sua execução.

Papel
Usuários
Descrição
São aqueles atores que não possuem poder de decisão nem estão envolvidos no projeto, mas que utilizam e são afetados pela solução em desenvolvimento. Neste grupo estão inclusos tanto servidores e procuradores quanto o público em geral.

Papel
Equipe de Projeto
Descrição
Os atores diretamente engajados na construção da solução, incluindo o requisitante, o *scrum master*, o líder técnico e os desenvolvedores.

1.11 História de Usuário

O software automatiza um algoritmo que pode ser demonstrado como solução para um dado problema. Tradicionalmente o requisito de software é definido como a declaração formal pela qual determinada solução pode ser aceita para um problema especificado. Como os algoritmos possuem uma estrutura lógica, nada mais natural que também os requisitos possuíssem uma linguagem que, buscando clareza e consistência, fossem também expressos com um mínimo de formalidade. A linguagem natural seria, por natureza, ambígua.

Ocorre que o raciocínio lógico, ferramenta poderosa, possui, contudo, limites quando se trata de captar os problemas do cotidiano humano. A comunidade ágil, ao colocar os indivíduos no centro do processo de desenvolvimento, logo percebeu que uma outra ferramenta, o raciocínio narrativo, era mais hábil não apenas para expressar os problemas, mas também para estimular o diálogo e o contato direto entre as pessoas. A contação de histórias permite uma compreensão da perspectiva dos usuários da solução, tornando a todos mais empáticos e capazes de ver além do requisito "frio".

A ferramenta história de usuário se difundiu, deste modo, tanto por sua simplicidade estrutural quanto pela capacidade de servir como um ponto de apoio, em linguagem natural, para as interações da equipe. Dentre outras possibilidades, uma história de usuário assume a forma:

"Enquanto uma [persona], eu quero [isto], para fazer [aquilo]".

Toda história de usuário expressa um papel, um desejo e um objetivo. Por exemplo, "enquanto [procurador do Distrito Federal], eu quero [um alerta de pendências], para [evitar perda de prazos]". Note-se que a história de usuário, em verdade, *não é o requisito de negócio*. Ela apenas transmite uma necessidade, a partir da qual os requisitos deverão ser explorados. Pode ser compreendida antes como um compromisso entre os envolvidos em um projeto para debater um requisito (SMART, 2015: p. 95):

> "Histórias não são uma forma escrita de requisitos; contar histórias, com palavras e imagens, através da colaboração, é um mecanismo que constrói uma compreensão compartilhada.
>
> Histórias não são os requisitos; elas são discussões sobre resolver problemas para

> nossa organização, nossos clientes e nossos usuários. Elas conduzem a acordos sobre o que construir" (PATTON & ECONOMY, 2014: p. 13).

Recomendamos a utilização de histórias de usuário como unidade básica em um projeto de software por ser uma técnica que permite a construção cooperativa dos requisitos ao mesmo tempo em que coloca a perspectiva do usuário (ou seja, o problema) no centro do processo criativo. Como consequência, é incontornável que o Negócio, na figura de seu representante, o PO, assuma a responsabilidade primária pela escrita das histórias de usuário. Elas são o ponto de partida na definição dos critérios de aceitação de uma funcionalidade.

Uma história de usuário precisa ser clara e compreensível para todos os envolvidos. Ela não deve apresentar um alto nível de detalhamento nem deve se comprometer previamente com uma solução. As tarefas de detalhar e propor uma solução devem ser compartilhadas por toda a equipe de um projeto, pois demandam o engajamento conjunto de conhecimentos técnicos e negociais. Na clássica formulação que Wake (2003), corroborada na clássica obra de Cohn (2004) captou no acrônimo em inglês "INVEST", uma boa história de usuário deve ser independente, negociável, valorosa, estimável, pequena e testável:

- Independente (*independent*): tanto quanto possível, as histórias devem poder ser implementadas em qualquer ordem, sem sobreposições.
- Negociável (*negotiable*): as histórias não são uma especificação escrita em pedra. Os detalhes da implementação serão discutidos e criados em conjunto por todos os envolvidos no projeto.

- Valorosa (*valuable*): toda história deve entregar algum valor de negócio, de modo que mesmo especificações não-funcionais devem poder ser percebidas pelo negócio como produzindo valor.
- Estimável (*estimable*): uma boa história pode ser estimada com precisão suficiente para que o negócio possa ordená-las segundo o valor que percebe.
- Pequena (*small*): as histórias procuram representar uma quantidade de trabalho que não ultrapassa algumas semanas de trabalho.
- Testável (*testable*): se a equipe não sabe como testar o requisito, provavelmente há detalhes não esclarecidos que devem tornar mais difícil que se alcance valor com a funcionalidade em desenvolvimento.

É recomendável, durante a execução das sprints, a utilização da técnica das reuniões de refinamento. Nelas o PO e a equipe de desenvolvimento buscam, por meio de práticas como a *especificação por exemplos (*ADZIC, 2011) ou o *mapeamento de exemplos* (BDD), construir em conjunto a solução para a necessidade de negócio apresentada na história de usuário. As reuniões devem consumir apenas o tempo necessário para que, de uma história, sejam identificados os critérios de aceitação segundo os quais a entrega será testada.

Recomendações para o uso de histórias de usuário (PATTON & ECONOMY, 2014; ADZIC, 2011):

1. Histórias de usuário devem focar no que o negócio quer – e apenas isso.
2. Comunicação antes de documentação: documentação compartilhada não é compreensão compartilhada.
3. O objetivo das histórias de usuário é a compreensão compartilhada: o diálogo importa mais que a escrita.

4. Boas histórias são sobre quem e o porquê, não sobre o quê.
5. Exemplos são uma boa ferramenta para evitar problemas de comunicação:
    - Se você não consegue dar bons exemplos, provavelmente você não tem informação suficiente para implementar o software.
    - Exemplos devem ser precisos e evitar ambiguidades: idealmente, devem ser sempre verificáveis.
    - Exemplos devem ser concretos: evite equivalências abstratas ou variáveis indefinidas; busque replicar a descrição da funcionalidade.
    - Exemplos devem ser completos: o conjunto de exemplos busca demonstrar na prática o escopo inteiro da funcionalidade.
    - Pense em caminhos alternativos para a funcionalidades.
    - Exemplos devem ser realistas, preferencialmente com dados reais (testagem *data-driven*).
    - Exemplos devem ser fáceis de entender.
    - Evite tentar explorar todas as possibilidades e foque nos exemplos que aprimoram a compreensão da funcionalidade.
    - Procure por conceitos implícitos e os explicite.
    - Use protótipos para aprimorar a compreensão dos requisitos não funcionais.

1.12 Desenvolvimento Orientado a Testes de Aceitação (ATDD)

Os critérios de aceitação de uma história de usuário devem ser expressos em linguagem natural para que possam ser claramente compreendidos por todos os envolvidos. Recomenda-se a utilização de uma linguagem declarativa como o *GHERKIN* nesta etapa. Com uma sintaxe simples e intuitiva, ele expressa um critério de aceitação da seguinte forma:

História de usuário: "enquanto [procurador do Distrito Federal], eu quero [um alerta de pendências], para [evitar perda de prazos]".

Exemplo 1: se [data final para manifestação no PJe] for menor que [7 dias úteis] e não houver [pendência encerrada] após a [data inicial para manifestação no PJe] o [procurador titular da ação] deve ser alertado [na tela inicial do sistema de peticionamento eletrônico].

Exemplo 2: se [data final para manifestação no PJe] for menor que [3 dias úteis] e não houver [pendência encerrada] após a [data inicial para manifestação no PJe] a [chefia da especializada] deve ser alertada [na tela inicial do sistema de peticionamento eletrônico].

Cenário: alerta de iminente perda de prazo:
Dado que: sou um [procurador do Distrito Federal] [titular] do [processo nº X];
Quando: [data final para manifestação no PJe] for menor que [7 dias úteis] **e** não houver [pendência encerrada] após a [data inicial para manifestação no PJe];
Então: o [procurador titular da ação] deve ser alertado [na tela inicial do sistema de peticionamento eletrônico].

Cenário: alerta de iminente perda de prazo:
Dado que: sou um [procurador do Distrito Federal] [titular] do [processo nº X];
Quando: [data final para manifestação no PJe] for menor que [3 dias úteis] **e** não houver [pendência encerrada] após a [data inicial para manifestação no PJe];
Então a [chefia da especializada] deve ser alertada [na tela inicial do sistema de peticionamento eletrônico].

Uma história de usuário será especificada por todo o conjunto de exemplos considerados relevantes pelos envolvidos. Cada exemplo será derivado em um cenário de teste, nos quais irão se basear os testes de aceitação do novo incremento de software.

O desenvolvimento orientado a testes de aceitação, ou ATDD, é uma metodologia ágil que procura utilizar técnicas de teste de software de forma intensa na persecução dos princípios ágeis de foco no valor de negócio e feedbacks rápidos e contínuos.

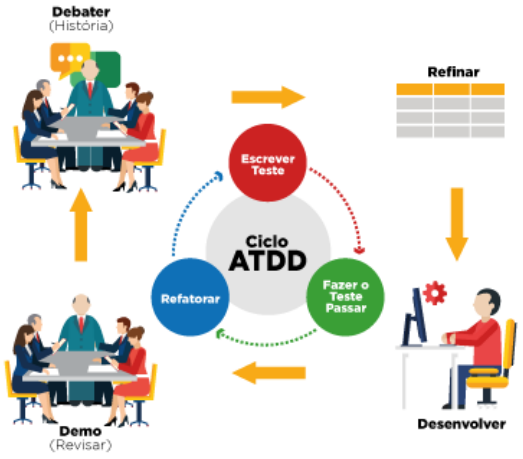


Figura 5 - Ciclo ATDD

O processo de desenvolvimento preconizado pelo ATDD, como demonstrado na figura 5, segue alguns passos básicos (AXELROD, 2018: p. 375): (1) definição de cenários e critérios de aceitação para histórias de usuário; (2) escrita de um teste para cada cenário; (3) escrita de uma aplicação para passar nos novos testes escritos sem falhar nos testes antigos; (4) entrega da aplicação e *feedback* do usuário. Há uma preocupação constante em garantir não apenas que o desenvolvimento seja norteado pelo valor de negócio que se procura suprir, mas que o

comportamento do software entregue atenda às especificações do problema a ser solucionado.

Na metodologia ATDD aqui recomendada, um incremento de software não testado não deve jamais ser considerado pronto. Só se pode atribuir o conceito de "pronto" àquele incremento que não apenas atende os critérios de aceitação de sua própria história de usuário (teste de aceitação) como também todo o conjunto de testes já escritos para aquele software (testes de integração e regressão). O teste constante e contínuo, que perpassa todo o ciclo de desenvolvimento, possui outros impactos desejáveis, como a redução do custo de manutenção de uma aplicação, uma vez que as partes mais relevantes de um código estarão cobertas por um plano de testes evoluído em conjunto com o software, e o estímulo à refatoração, uma vez que os desenvolvedores possuem a proteção de um plano de testes para aperfeiçoar partes funcionais da aplicação.

1.13 Testes e Processo de Qualidade

O teste é tanto uma etapa do ciclo de desenvolvimento quanto uma atitude coletiva dos envolvidos em um projeto de software. Esta atitude nasce da compreensão do objetivo último de qualquer processo de testes: a qualidade do produto. O propósito da testagem é aumentar a capacidade do time de desenvolvimento em entregar um produto com qualidade (AMMANN & OFFUTT, 2017: p. 37), motivo pelo qual se deve evitar uma separação de trabalhos estanque entre testadores e desenvolvedores.

De fato, projetos ágeis tendem a considerar o teste como uma responsabilidade compartilhada entre todo o time (SMART, 2015: p. 229), em contraste com a visão mais convencional que se vale de uma fase de testagem à parte, com um time dedicado.

Desnecessário aqui definir um formato. São as particularidades de cada projeto que o definirão. Contudo, vale frisar, é essencial que em todas as etapas do ciclo de vida de um software exista a conjugação de três perspectivas: o negócio, o desenvolvimento e o teste (os "três amigos", figura 6).

Figura 6 - "Reunião dos três amigos"

O processo de qualidade busca pautar o processo de desenvolvimento nana promoção do valor de negócio. Isto é realizado por meio de atividades de verificação e validação.

A primeira diz respeito à correta implementação do requisito, a segunda, por sua vez, questiona se o software entregue satisfaz seu usuário. A verificação é composta por atividades técnicas em um sentido mais estrito, que utilizam o rol de testes definidos na engenharia para avaliar a implementação de um artefato de software; já a validação se vale também de técnicas de coleta de informação do usuário e realiza, neste sentido, uma exploração conjunta do domínio de software, mais que de artefatos particulares. A validação, além disto, revela se a própria especificação de requisitos está correta.

Para qualquer incremento de software se deve partir da definição dos critérios de aceitação para, então, construir o plano de testes específico que deverá orientar o desenvolvimento. Recomenda-se, portanto, a utilização de práticas como o *test-driven development* (TDD), que preconiza uma escrita de código desde o início voltada a passar no plano de testes especificado. **O requisito é o teste.**

Neste sentido, as tarefas de verificação, executadas de forma constante no interior da mesma *sprint* que o desenvolvimento, são derivadas das características do incremento de software, ou seja, do valor do negócio. É a funcionalidade implementada que determina a necessidade de se realizar um teste de performance, de carga, de stress, dentre outros. Não obstante, cada *sprint* do processo começa e termina em uma necessidade de negócio (Figura 7), primeiro expressa como desejo (a história de usuário), depois substanciada como capacidade (uma funcionalidade em produção).

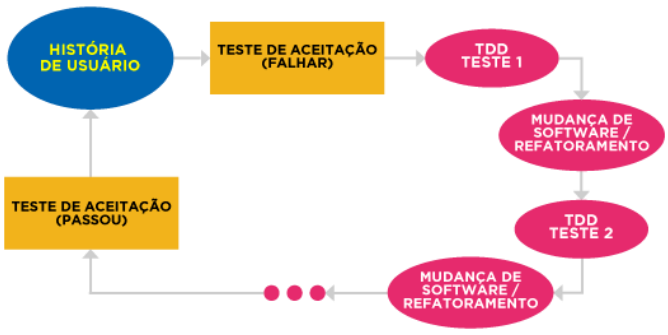


Figura 7 - Centralidade da História de Usuário

A escrita de código centrada num objetivo claro e expresso por um critério de aceitação em formato de teste sustenta e torna transparente para os desenvolvedores a compreensão compartilhada dos objetivos de negócio. Como demonstra a figura 8, a necessidade de alcançar um código bem-sucedido em um teste de aceitação estimula a escrita prévia de

testes unitários, contagiando, assim, todo o processo com uma cultura de qualidade.

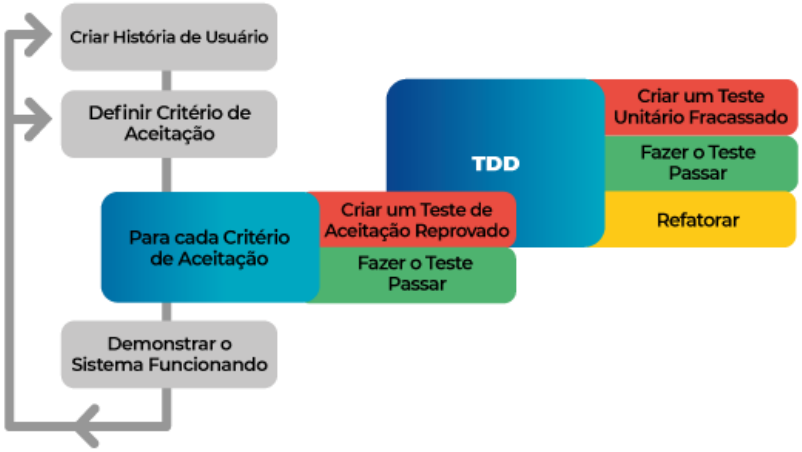


Figura 8 - Do ciclo ATDD à prática TDD

Por outro lado, testes funcionais e testes, de qualidade de código (análise estática) são recomendáveis na composição de todos os planos de testes. O teste funcional possui uma centralidade metodológica por sua capacidade de validar o software desenvolvido, com consequências para todo o projeto. O teste de qualidade de código, por sua vez, por verificar a consistência do incremento quanto à adesão às melhoras práticas de codificação, torna o software mais legível e conciso, favorecendo a qualidade arquitetural como um todo ao longo do tempo.

O processo de qualidade ágil supõe uma definição mais estrita da correção do software, que passa a se referir apenas a um conjunto específico de testes, não ao software como um todo. Esta modesta definição intenta ser, fundamentalmente, útil e operativa no interior de um processo de desenvolvimento:

> Todos os métodos ágeis possuem uma pressuposição subjacente segundo a qual no

> lugar de *definir todos* os comportamentos com requisitos ou especificações, nós *demostramos alguns* comportamentos com testes específicos. O software é considerado correto se ele passa em conjunto particular de testes (AMMANN & OFFUTT, 2017: p.98).

Como os casos de teste são as especificações do sistema, a documentação técnica do sistema – entendida aqui como as descrições do comportamento esperado – devem ser especificações executáveis, referidas de modo direto às partes do código. A automação dos testes e a integração contínua são resultados esperados e desejáveis, que se assentam sobre esta prática de documentação que busca a capacidade de **demonstrar** um comportamento.

Fluxo do PDS

O PDS é um fluxo racional e ordenado dos passos necessários à construção de soluções em sintonia com o valor de negócio. Ele se inicia com a idealização, ainda a nível estratégico, de uma necessidade a ser suprida ou um anseio a ser alcançado pela PGDF.

O primeiro passo do fluxo, como se pode ver nas figuras 9 e 10, é a sedimentação desta idealização em um documento de visão, que expressa os grandes marcos de valor, abrindo um projeto.

Seguindo o princípio ágil da construção incremental, constantemente pautada e validada pelo requisitante, é definido um mapa com aqueles passos menores capazes de, com relativa autonomia, trazer valor ao negócio. Cada um destes passos, funcionalidades capazes de resolver problemas, são agrupados em *releases*, que, por sua vez, são construídas em ciclos interativos menores, as *sprints*.

O fluxo mostra a conjugação de esforços da equipe de projeto – notadamente o PO, o líder técnico, o *scrum master* e a equipe de desenvolvimento – ao longo de quatro fases: planejamento das *sprints*, construção, teste de qualidade e tarefas de transição. Quanto a isto não há diferença entre o desenvolvimento estritamente interno (figura 9) e aquele com contratação de um ator externo (figura 10). Sempre será necessário que um requisitante expresse uma necessidade, a ser solucionada gradualmente, com tarefas menores, por uma equipe de desenvolvimento, apoiada por *scrum master* e um líder técnico.

As especificidades da gestão de contratos públicos, por sua vez, comandam a elaboração de um artefato que formaliza o ateste de qualidade pela equipe técnica antes da disponibilização para homologação de um incremento de software: um relatório que afere a conformidade estrutural e a aderência às regras de negócio do incremento entregue pela Contratada.

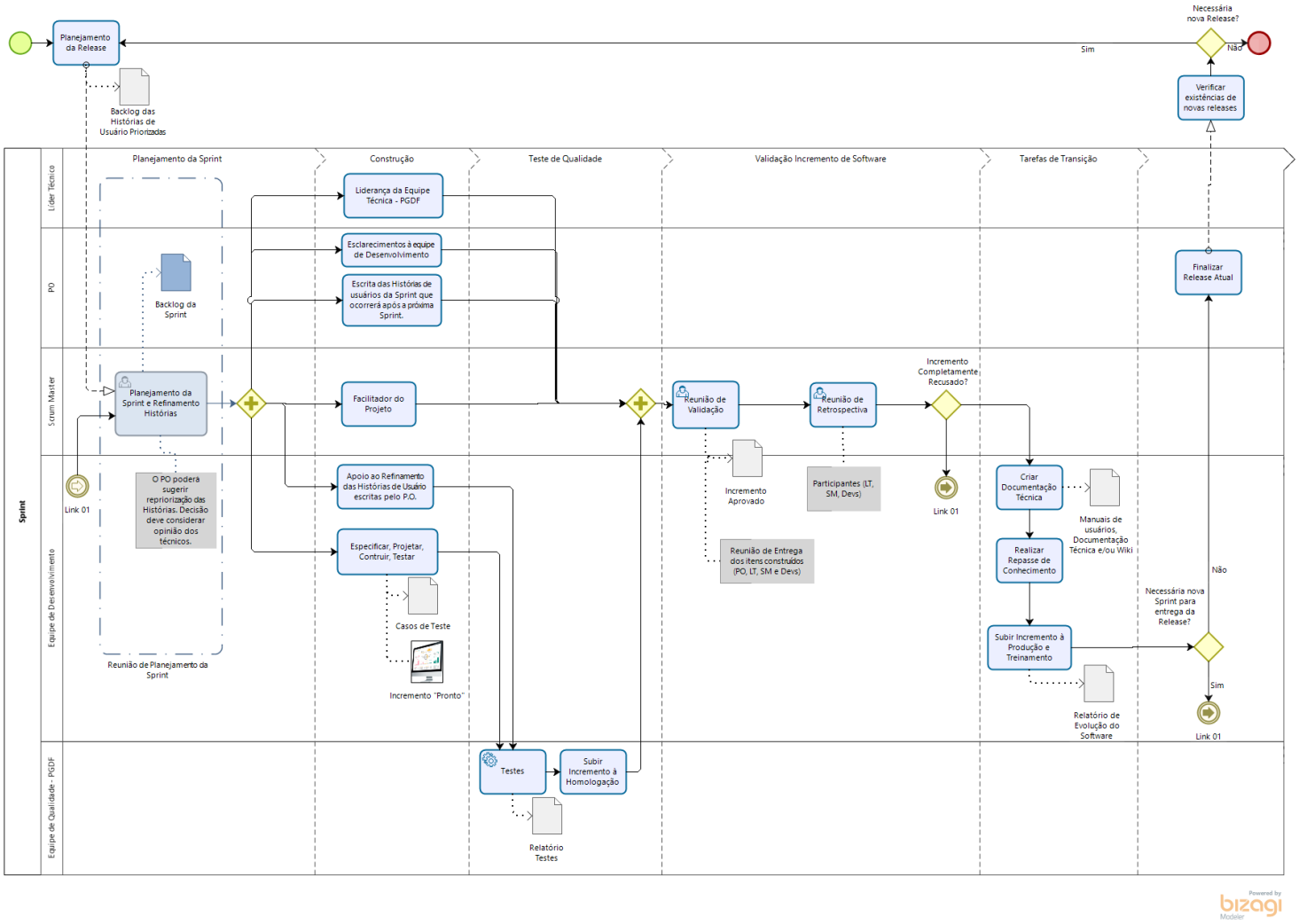


Figura 9 - Fluxo para desenvolvimento interno

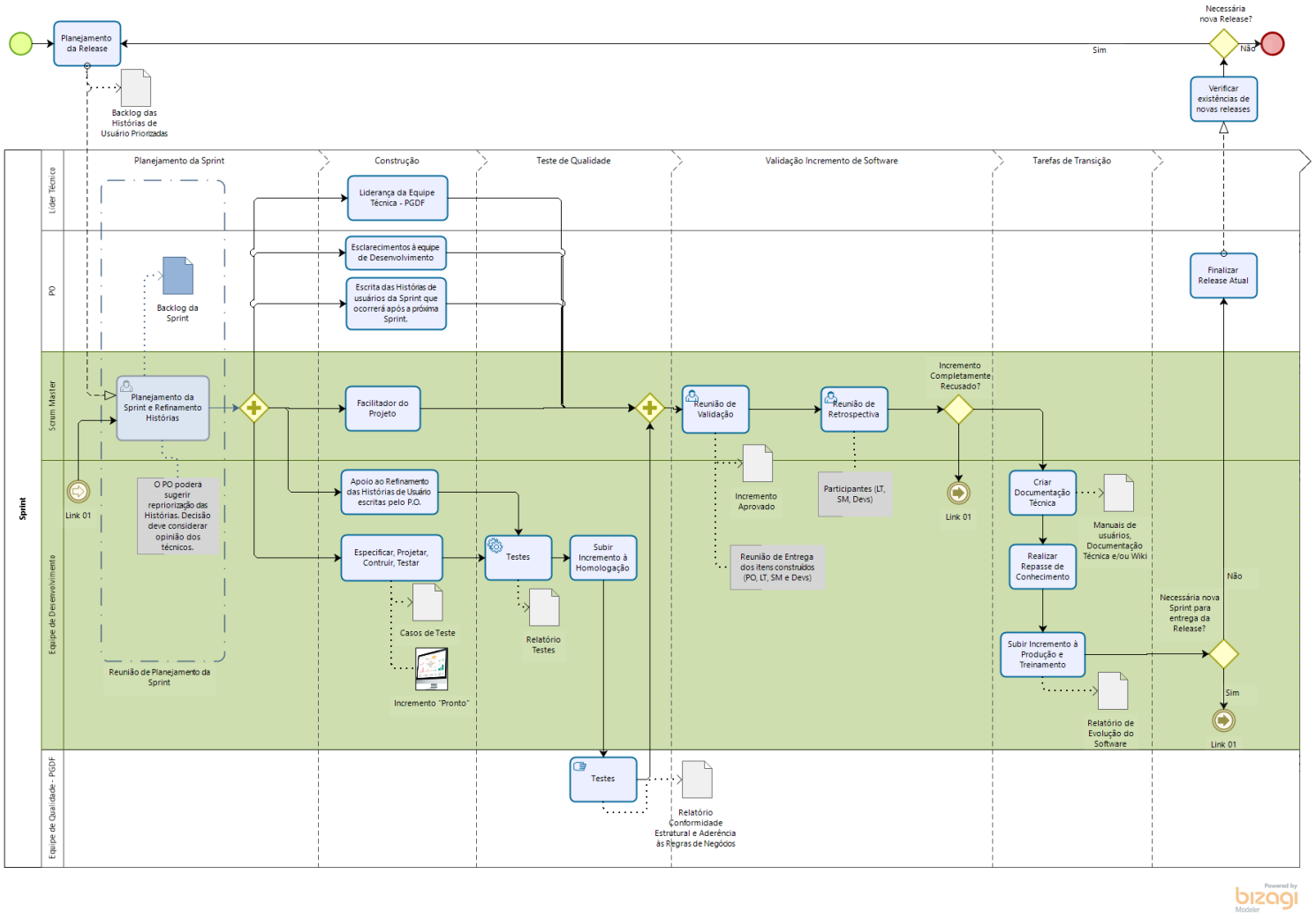


Figura 10 - Fluxo para desenvolvimento com terceirização

PARTE III

Atividades do PDS

A seguir são apresentadas (10.1) e detalhadas (10.2) as atividades que operacionalizam o fluxo do PDS. A visão de fluxo por atividades auxilia a compreensão dos fluxos apresentados no item 9.

1.14 Fluxo de Atividades do PDS

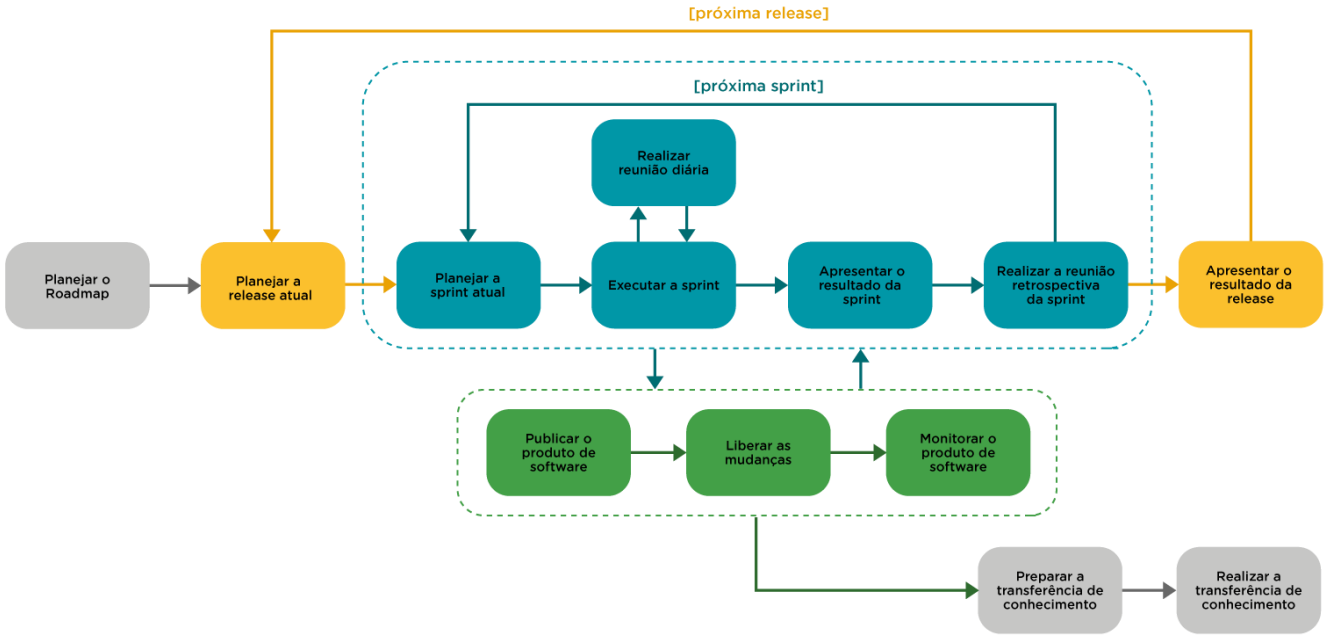


Figura 11 - Fluxo de Atividades do PDS

1.15 Detalhamento das atividades do PDS

1.15.1Atividades de Iniciação

Atividade:
Planejar o Roadmap (sprint zero)
Descrição:
Planejamento do mapa com os marcos de entrega mais significativos para o alcance do valor do negócio.
Entradas:
• Descrição dos problemas do negócio • Documento de visão do projeto
Tarefas:
1. O product owner deve definir (técnica impact mapping), em conjunto com a equipe de desenvolvimento, os objetivos das releases do produto: a. O product owner deve alinhar os objetivos das releases aos problemas do negócio; b. O product owner e a equipe de desenvolvimento devem planejar entregas de releases frequentes; c. O product owner deve priorizar as releases pelo valor que agregam ao negócio; d. O product owner deve considerar na definição dos objetivos das releases: timebox do projeto, prazos legais, complexidade do negócio e existência de sistemas legados, entre outras características.
Saídas:
• Backlog de releases • Backlog do produto

1.15.2 Atividades de Construção

Atividades:
Planejar a release atual
Descrição:

O planejamento da sprint é feito em uma reunião de aproximadamente uma hora no início de cada release e conta com a participação de toda a equipe do projeto (desenvolvedores, scrum master, product owner, líder técnico, quando for o caso, representantes das demais áreas envolvidas).

Entradas:

- Backlog da release
- Lista de ocorrências, defeitos e débitos técnicos, se houver

Tarefas:

1. O product owner deve revisar o objetivo da release atual;
2. O product owner deve se certificar de que o objetivo da release permanece alinhado aos problemas do negócio;
    a. O product owner deve certificar-se de que o objetivo da release continua agregando valor ao negócio;
    b. O product owner pode solicitar o apoio da equipe de desenvolvimento na revisão do objetivo.
3. A equipe do projeto deve revisar o backlog da release (refinamento do backlog).
4. O product owner deve verificar se todas as histórias contribuem para que o objetivo da release seja alcançado;
    a. A release deve conter um conjunto mínimo de funcionalidades que, juntas, sejam úteis para o negócio;
    b. A equipe do projeto deve verificar se há histórias muito grandes que devem ser quebradas;
    c. A equipe de desenvolvimento deve revisar as estimativas das histórias;
    d. A equipe de desenvolvimento deve revisar a data da release de acordo com as estimativas das histórias, a velocidade da equipe e o timebox do projeto;
    e. As histórias associadas à release podem ser revistas, incluídas ou excluídas sempre que necessário.
5. O product owner deve priorizar as histórias da release (técnica user story mapping).
6. O product owner deve atribuir um valor de negócio para cada história.
    a. Na priorização convém que sejam considerados: valor de negócio, esforço de implementação, risco, restrição;
    b. A priorização pode ser revista sempre que necessário.
7. A equipe do projeto deve estabelecer, em consenso, os critérios de aceitação para a release;
8. Convém que a equipe do projeto considere o custo de validação dos critérios de aceitação ao defini-los;
9. O product owner deve propor critérios de aceitação de negócio que validem o atendimento ao objetivo da release;
10. O líder técnico deve propor critérios de aceitação técnicos derivados dos padrões arquiteturais e normativos da PGDF;

11. A equipe de desenvolvimento pode propor outros critérios de aceitação técnicos destinados a mitigar riscos identificados no projeto.
12. O scrum master deve atuar como facilitador na reunião de planejamento da release;
13. O líder técnico deve monitorar o registro do backlog e dos critérios de aceitação da release atualizados;
14. O líder técnico deve atualizar o plano de releases na ferramenta de gestão de projetos, se necessário.
15. A equipe do projeto deve definir a estratégia de publicação (entrega contínua) para a release;
16. A equipe do projeto pode rever a estratégia de publicação a qualquer momento durante a release;
    a. Convém que as publicações sejam frequentes, pois uma quantidade menor de mudanças em cada publicação a torna mais segura.

Saídas:

- Backlog da release
- Plano de releases atualizado
- Estratégia de publicação

Atividade:
Planejar a Sprint Atual
Descrição:
É a iniciação de cada ciclo iterativo de desenvolvimento do projeto, onde os itens de backlog priorizados devem ser estruturados em uma divisão racional de tarefas menores.
Entradas:
• Backlog da release • Lista de ocorrências, defeitos e débitos técnicos
Tarefas:
1. O product owner deve informar à equipe de desenvolvimento o objetivo e as prioridades da sprint; a. Convém que sejam implementadas as histórias que agreguem maior valor ao negócio; b. As histórias devem estar alinhadas ao objetivo da release. 2. A equipe de desenvolvimento deve informar ao product owner sua capacidade de entrega na sprint; a. A equipe deve considerar a sua velocidade e a quantidade de pessoas na sprint atual 3. A equipe do projeto deve selecionar as histórias a serem implementadas na sprint; a. A equipe do projeto deve definir o objetivo da sprint;

b. A equipe de desenvolvimento deve comprometer-se com o objetivo da sprint;
c. O objetivo da sprint deve estar disponível no ambiente informativo do projeto.

4. A equipe do projeto deve definir, em consenso, a lista de critérios de aceitação a ser aplicada a cada história incluída no backlog da sprint;
   a. Convém que a equipe do projeto considere o custo de validação dos critérios de aceitação ao defini-los;
   b. O product owner deve propor critérios de aceitação de negócio que validem o atendimento ao objetivo da release;
   c. O líder técnico deve propor critérios de aceitação técnicos derivados dos padrões arquiteturais e normativos da PGDF;
   d. A equipe de desenvolvimento pode propor outros critérios de aceitação técnicos destinados a mitigar riscos identificados no projeto;
   e. Os critérios de aceitação devem ser refinados ao longo da Sprint por meio de reuniões de refinamento.
5. A equipe de projeto deve definir a regra para que itens de backlog possam "furar a fila".
6. A equipe de desenvolvimento deve decompor as histórias do backlog da sprint em tarefas.
   a. O product owner deve esclarecer as dúvidas de negócio da equipe de desenvolvimento a respeito das histórias;
   b. A equipe de desenvolvimento deve considerar os aspectos significativos de arquitetura e projeto durante a identificação das tarefas;
   c. Convém que a duração de uma tarefa não ultrapasse um dia de trabalho.
7. A equipe de desenvolvimento deve identificar e incluir no backlog da sprint as tarefas técnicas e de débito técnico.
8. A equipe de desenvolvimento deve disponibilizar o objetivo, as histórias e tarefas no ambiente informativo do projeto.
9. O scrum master deve atuar como facilitador na reunião de planejamento da sprint.
10. O líder técnico deve monitorar a atualização do backlog da sprint e critérios de aceitação das histórias.

Saídas:

- Backlog da Sprint
- Ambiente informativo atualizado

Atividade:
Executar a Sprint
Descrição:

Dia a dia do projeto, é o conjunto de atividades que corresponde à transformação das necessidades de negócio em um incremento de software.

Entradas:

- Backlog da Sprint
- Produto de software

Tarefas:

1. A equipe de desenvolvimento deve gerir o backlog da sprint;
   Notas: A equipe de desenvolvimento deve atualizar o status das tarefas no ambiente informativo;
   a. A equipe de projeto pode incluir, alterar ou excluir tarefas no backlog da sprint observando sempre seu objetivo.
2. A equipe de desenvolvimento deve selecionar as próximas tarefas do backlog da sprint a serem executadas;
   a. Convém que a equipe de desenvolvimento trabalhe em uma história por vez, seguindo a priorização do backlog da sprint;
   b. O timebox da sprint deve ser respeitado.
3. A equipe de desenvolvimento deve atribuir responsável para as tarefas selecionadas.
4. O product owner deve definir, em conjunto com a equipe de desenvolvimento, os cenários de testes de aceitação especificação por exemplo das histórias a serem implementadas:
   a. O product owner deve considerar, além dos requisitos funcionais, os requisitos de interface, de usabilidade e outros requisitos não funcionais
5. A equipe de desenvolvimento deve implementar a solução mais simples que atenda aos requisitos do negócio;
   a. A implementação da solução pode compreender a modelagem do domínio, o projeto da solução, a modelagem do banco de dados, a instrumentação dos cenários de teste de aceitação, a implementação de testes de unidade, a codificação, a refatoração, a integração contínua do código, dentre outras ações.
   b. O product owner deve estar disponível para a equipe de desenvolvimento, quando solicitado;
   c. Convém que o product owner convide outros stakeholders para esclarecer dúvidas junto à equipe de desenvolvimento e dar feedback do produto, quando necessário;
   d. Todos os artefatos produzidos devem estar aderentes aos padrões corporativos e de qualidade vigentes;
   e. Todos os artefatos devem ser colocados sob a gerência de configuração segundo seus padrões técnicos;
   f. A equipe de desenvolvimento deve garantir a propriedade coletiva do código e sua manutenibilidade;
   g. A equipe de desenvolvimento deve criar tarefas para os débitos técnicos identificados.
6. O scrum master deve atuar como facilitador na execução da sprint;

a. O scrum master deve assegurar que o processo seja executado corretamente viabilizando a atuação eficaz de toda equipe de projeto;
b. O scrum master deve observar o rodízio dos desenvolvedores na execução das tarefas.

7. O líder técnico deve garantir à equipe de desenvolvimento um ambiente livre de interferências externas.
   a. O líder técnico e o scrum master devem atuar para remover os impedimentos relatados na reunião diária pelos desenvolvedores;
   b. O líder técnico deve comunicar o andamento do projeto às partes interessadas.

Saídas:

- Incremento de software
- Ambiente informativo atualizado

Atividade:
Realizar a reunião diária
Descrição:
A reunião deve ser realizada diariamente, em horário fixo definido pela equipe. Deve ser breve, com duração aproximada de 15 minutos. A equipe de desenvolvimento deve ter acesso ao ambiente informativo do projeto.
Entradas:
• Ambiente informativo
Tarefas:

1. A equipe de desenvolvimento deve reunir-se diariamente para comunicação, nivelamento dos participantes, acompanhamento da sprint e planejamento dos trabalhos;
   a. Necessário que o product owner participe da reunião;
   b. Stakeholders podem ser convidados a participar da reunião;
   c. Cada membro da equipe de desenvolvimento deve responder às seguintes perguntas: “o que fiz desde a última reunião?”; “o que pretendo fazer até a próxima reunião?” e “quais impedimentos tenho para realizar as tarefas?”;
   d. Convém que os assuntos de natureza administrativa não sejam tratados na reunião, tais como controle de horário e prestação de contas;
   e. A equipe de desenvolvimento deve comunicar as soluções ou problemas encontrados de forma sucinta;
   f. O timebox da reunião diária deve ser respeitado.
2. A equipe de desenvolvimento deve avaliar se está convergindo para o objetivo da sprint;

a. A equipe de desenvolvimento deve discutir as propostas de ajustes necessários após a reunião
3. A equipe de desenvolvimento deve avaliar se as ações de melhoria identificadas na última reunião de retrospectiva estão sendo executadas.
4. A equipe de desenvolvimento deve entrar em consenso sobre quais tarefas serão executadas até a próxima reunião.
5. O líder técnico deve registrar tarefas para as ações de mitigação dos impedimentos relatados pelos desenvolvedores.

Saídas:

- Ambiente informativo atualizado

Atividade:
Apresentar o resultado da Sprint
Descrição:
Reunião, geralmente de até uma hora, realizada ao final de cada sprint, na qual a equipe de desenvolvimento apresenta o resultado do trabalho da sprint ao product owner, e eventuais stakeholders e usuários por ele convidados.
Entradas:
• Backlog da sprint • Produto de software
Tarefas:

1. O product owner pode convidar os stakeholders ou grupo de usuários das funcionalidades implementadas para participar da reunião;
2. O líder técnico deve apresentar aos presentes o objetivo da sprint no início da reunião;
3. A equipe de desenvolvimento deve apresentar o incremento de produto de software gerado;
    a. Convém que a equipe de desenvolvimento apresente a situação de cada história do backlog da sprint;
    b. Convém que o foco da apresentação esteja em questões de negócio, e não técnicas;
    c. O product owner deve usar o software para testar as funcionalidades entregues.
    d. É recomendável que o product owner convide usuários-chave do sistema para que testem a solução apresentada. Durante essa etapa, a equipe de desenvolvimento pode colher insumos para melhorar usabilidade do sistema;
    e. No curso dessa reunião os usuários podem sugerir novas funcionalidades e melhorias do sistema. Cabe ao product owner e à equipe de desenvolvimento registrar esses pedidos para avaliar sua implementação em momento posterior.
4. O product owner deve dar feedback sobre o incremento de produto de software apresentado.
5. O product owner deve verificar se os critérios de aceitação de negócio das histórias entregues foram atendidos.
6. O líder técnico deve verificar se os critérios de aceitação técnicos das histórias entregues foram atendidos.
7. A equipe de desenvolvimento deve incluir, no backlog do produto, histórias ou tarefas para os defeitos e pontos de melhoria identificados pelo product owner.
    a. O product owner deve priorizar as histórias e tarefas criadas.
8. O scrum master deve atuar como facilitador da reunião de apresentação do resultado da sprint
9. O líder técnico deve registrar o resultado da sprint: a velocidade da equipe, as histórias prontas, a data de finalização e se o objetivo da sprint foi alcançado, entre outros dados.
    a. O líder técnico deve monitorar a atualização do backlog do produto;
    b. O líder técnico deve atualizar os status de acompanhamento gerencial do projeto, quando necessário.

Saídas:

- Backlog do produto atualizado.
- Relatório de execução da Sprint.

Atividade:
Realizar a reunião retrospectiva da sprint
Descrição:

Reunião, de até meia hora geralmente, realizada após a reunião de apresentação de cada sprint, na qual a equipe avalia a execução da sprint.
Responsáveis:
Equipe de desenvolvimento, scrum master e líder técnico.
Entradas:
• Prime Directive: não importa o que descobriremos nessa reunião, consideraremos que as pessoas agiram dessa forma devido aos conhecimentos que possuíam na época, tempo e recursos disponíveis. • Pontos positivos e Negativos da Sprint.
Tarefas:
1. A equipe do projeto deve identificar os pontos positivos e negativos observados durante a execução da sprint; 2. A equipe do projeto deve identificar a causa raiz dos pontos negativos levantados; 3. A equipe do projeto deve propor soluções para as causas raiz identificadas; a. A equipe do projeto deve decompor as ações em tarefas ou lembretes e disponibilizá-los no ambiente informativo. 4. O scrum master deve atuar como facilitador na reunião de retrospectiva da sprint
Saídas:
• Ações de melhoria no processo de trabalho da equipe • Ambiente informativo atualizado

Atividade:
Apresentar do resultado da release
Descrição:
Reunião, de até duas horas geralmente, realizada ao final de cada release, na qual o product owner e o líder técnico, em conjunto com toda a Equipe, apresentam o resultado da release aos stakeholders e usuários por eles convidados.
Entradas:
• Incremento de Software; • Documentação Técnica, quando existente.
Tarefas:

1. O product owner deve apresentar o objetivo da release;
    a. O product owner deve convidar os stakeholders interessados nas funcionalidades entregues na release para participar da reunião;
    b. O product owner deve contextualizar os stakeholders para o entendimento da apresentação;
    c. O product owner deve informar os impactos nos processos de negócio envolvidos.
2. O product owner deve apresentar o produto de software aos stakeholders com o apoio da equipe de desenvolvimento;
3. O líder técnico deve registrar os defeitos e pontos de melhoria identificados pelos stakeholders;
    a. Os defeitos e melhorias identificados devem ser registrados no backlog do produto.
4. O scrum master deve atuar como facilitador na reunião de apresentação do resultado da release;
5. O product owner deve avaliar se os critérios de aceitação de negócio da release foram atendidos;
6. A área de qualidade deve avaliar se os critérios de aceitação técnicos da release foram atendidos;
7. O product owner deve formalizar o aceite das funcionalidades entregues na release;
8. O líder técnico deve registrar o resultado da release: data de finalização, situação, aceite do product owner, dentre outros.
    a. O líder técnico deve atualizar o Plano de Releases;
    b. O líder técnico deve evidenciar as histórias prontas, entregues na release;
    c. O líder técnico deve atualizar o ambiente informativo.

Saídas:

- Backlog do produto atualizado;
- Plano de integração atualizado;
- Plano de releases atualizado;
- Relatório de execução da release.

1.15.3 Atividades de Implantação

Atividade:
Publicar o produto de software
Descrição:
Tornar disponível uma versão do produto de software no ambiente de produção.
Entradas:

- Produto de software
- Estratégia de publicação
- Aprovação do PO para entrada do sistema em produção.

Tarefas:

1. A equipe de desenvolvimento deve qualificar o software para publicação em produção;
    a. i. A equipe de desenvolvimento deve garantir que o produto de software atende os requisitos de eficiência (carga, desempenho e outros) e os padrões visuais; ii. A equipe de desenvolvimento deve obter a qualificação do software quanto à vulnerabilidade e acessibilidade caso o mesmo seja destinado ao uso na internet.
2. A equipe de desenvolvimento deve agendar a publicação;
    a. A equipe de desenvolvimento deve avaliar o impacto e o risco da publicação de acordo com o momento da publicação.
3. A equipe de desenvolvimento deve obter autorização para a publicação.
4. A equipe de desenvolvimento deve disponibilizar a publicação;
5. A equipe de desenvolvimento deve verificar se o produto de software está funcionando conforme esperado;
    a. Convém automatizar as verificações do produto de software.
6. Convém que a equipe de desenvolvimento atualize a lista de mudanças publicadas e notifique o *product owner*.

Saídas:

- Produto de software disponível no ambiente de produção
- Lista de mudanças publicadas contendo, no mínimo:
    1. Data de publicação;
    2. Versão publicada;
    3. Funcionalidades publicadas.

Atividade:

Liberar as mudanças

Descrição:

Tornar as mudanças ativas para os usuários

Entradas:

- Produto de software disponível no ambiente de produção
- Lista de mudanças publicadas

Tarefas:

1. O product owner deve definir quais mudanças serão liberadas;
Nota: Convém que as mudanças publicadas sejam liberadas o quanto antes para obtenção do feedback dos usuários.
    a. O product owner deve definir quais usuários terão acesso às mudanças liberadas;
    Notas: Convém liberar mudanças de maior risco inicialmente para um grupo reduzido de usuários, com o intuito de minimizar possíveis impactos negativos;
    b. Convém liberar a mudança para os demais usuários assim que possível.
    c. A equipe do projeto deve liberar as mudanças conforme definido pelo product owner;
    d. Convém que o product owner notifique os usuários da release de mudanças significativas;
    e. Convém que a equipe do projeto atualize a lista de mudanças liberadas;
    f. Convém tornar disponível aos interessados a lista de mudanças liberadas.

Saídas:

- Produto de software com mudanças liberadas
- Lista de mudanças liberadas

Atividade:
Monitorar o produto do software
Descrição:
Acompanhar o comportamento do produto de software e definir ações corretivas ou preventivas
Entradas:

- Produto de software disponível no ambiente de produção
- Lista de mudanças liberadas
- Lista de mudanças publicadas

Tarefas:

1. A equipe do projeto deve acordar a estratégia de monitoramento do produto de software;
2. A equipe de desenvolvimento deve monitorar o ambiente do produto de software conforme estratégia acordada;
3. A equipe de desenvolvimento deve monitorar as ações dos usuários e respostas do produto conforme estratégia acordada;
4. A equipe de desenvolvimento deve decidir as ações preventivas ou corretivas necessárias.

Saídas:

- Lista de ações preventivas ou corretivas

Atividade:

Preparar a transferência de conhecimento

Descrição:

Compreende o planejamento das ações de transferência de conhecimento sobre o produto de software aos operadores, mantenedores, gestores e usuários, bem como a preparação do material necessário à realização dessas ações. Ocorre ao longo do desenvolvimento do produto de software, tendo duração variável e envolve toda a equipe do projeto.

Entradas:

- Produto de software
- Plano de releases
- Documento de Arquitetura

Tarefas:

1. O product owner, com o apoio do líder técnico, deve planejar a estratégia de transferência de conhecimento para gestores de negócio e usuários do produto de software;
   Notas: Convém que o planejamento da estratégia identifique:
   a. Os aspectos técnicos relevantes ao uso do software;
   b. Os conhecimentos e as habilidades necessárias aos gestores de negócio e usuários para o exercício de suas responsabilidades.
   c. Convém que o product owner considere o plano de liberações para orientar a estratégia de transferência de conhecimento.
2. O product owner deve, com o apoio da equipe de desenvolvimento, produzir o manual de usuário e o material de treinamento, quando necessário;
   Nota: O product owner deve produzir o material específico para treinamento dos gestores de negócio ou usuários;
3. A equipe de desenvolvimento deve planejar a estratégia de transferência de conhecimento para operadores e mantenedores do produto de software;
   Notas: A equipe de desenvolvimento deve documentar os aspectos técnicos do software e os procedimentos necessários para a manutenção do mesmo;
   a. A equipe de desenvolvimento deve atualizar a seção “Arquitetura da aplicação” do Documento de Arquitetura;
   b. A documentação do sistema deve permanecer sempre atualizada.
4. A equipe do projeto deve registrar o planejamento e as estratégias definidas no plano de transferência de conhecimento.

Saídas:

- Plano de transferência de conhecimento
- Material de treinamento
- Manual de usuário
- Documento de Arquitetura atualizado

Atividade:
Realizar a transferência de conhecimento
Descrição:
Efetiva transferência de conhecimento sobre o produto de software aos operadores, mantenedores, gestores e usuários do produto de software. Ocorre de acordo com a estratégia definida e tem a participação de toda a equipe do projeto.
Entradas:
• Manual de usuário • Material de treinamento • Plano de transferência de conhecimento • Produto de software
Tarefas:
1. O product owner, com o apoio do do líder técnico, deve realizar a transferência de conhecimento aos gestores de negócio e usuários do produto de software conforme a estratégia definida; 2. A equipe de desenvolvimento deve realizar a transferência de conhecimento aos operadores e mantenedores do produto de software conforme a estratégia definida. **Nota:** Convém que a equipe de desenvolvimento faça um repasse presencial para a equipe de manutenção.
Saídas:
• Transferência de conhecimento realizada

Referências

ADZIC, Gojko. **Specification by Example: How successful teams deliver the right software**. Shelter Island, NY: Manning Publications, 2011.

Agile Manifest web site. 2001. Disponível em: <http://agilemanifesto.org/iso/ptbr/>. Acesso em: janeiro de 2022. (Agile Manifest, 2001).

ARANHA, Dandara e CARDOSO, Maxuel. **"Métodos e Métricas para Estimativas e Planejamento de Projetos Ágeis de Software: Estudo Comparativo entre Pontos de Função e Story Points". UNB**, Brasília, 2017.

AXELROD, Arnon. **Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects**. NYC: Apress, 2018.

AMMANN, P. & OFFUTT, J. **Introduction to Software Testing.** Cambridge: Cambridge University Press, 2017.

BARTIÉ, A. **Garantia da Qualidade de Software**. Elsevier Editora. Rio de Janeiro. 2002.

CONH, Mike. "**Agile Estimating and Planning".** New York: Prentice Hall, 2005.

_____. **User Stories Applied for Agile Software Development.** Boston: Addison Wesley, 2004.

GAFFAR, Nisma MOUSSA, Hanan, KAMEL, Amr e H. GALAL-EDEEN, Galal. **"A Proposed Framework for Enhancing Story Points in Agile Software Projects"**. Indian Journal of Science and Technology, Vol 11(31): India, 2018.

KURNIA, Reni**. "Software Metrics Classification for Agile Scrum Process: A Literature Review".** International Seminar on

Research of Information Technology and Intelligent Systems: Yogyakarta, Indonesia, 2018.

SOMMERVILLE, Ian. **Engenharia de software**. 8ª ed. São Paulo: Addison-Wesley, 2007.

NBR ISO/IEC 12207 - Tecnologia de informação - Processos de ciclo de vida de software; ABNTAssociação Brasileira de Normas Técnicas; Outubro, 1998.

SCRUM GUIDE. Disponível em https://scrumguides.org/. Acesso em janeiro de 2022 (Scrum Guide 2021).

SMART, J. F. **BDD in Action Behavior-Driven Development for the whole** software lifecycle. Shelter Island, NY: Manning Publications, 2015.

PATTON, J. & ECONOMY, P. **User Story Mapping: Discover the Whole Story, Build the Right Product**. O'Reilly Media: 2014.

ROSSEBERG, Joachim. "**Agile Project Management with Azure DevOps. Concepts, Templates na Metrics".** California: APRESS, 2019.

SLTI/MPOG. **Instrução Normativa Nº 4, de 12 de novembro de 2010**. Secretaria de Logística e Tecnologia da Informação, Ministério do Planejamento, Orçamento e Gestão. Dispõe sobre o processo de contratação de serviços de Tecnologia da Informação pela Administração Pública Federal direta, autárquica e fundacional. Brasília, 2008.

VAZQUEZ, Carlos Eduardo; Simões, Guilherme e Albert, Renato. **Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software**. 9ª ed. São Paulo: Érica, 2010.

WAKE, William C. **"INVEST in Good Stories, and SMART Tasks"**. Disponível em xp123.com/articles/invest-in-good-stories-and-smart-tasks: 2003, acesso em 17/06/2023.

ZAHRAOUI, Hind e IDRISSI, Mohammed Abdou Janati. "**Adjusting Story Points Calculationin Scrum Effort &Time Estimation".** Índia, 2015.

1.16 Anexo I – Definições, Conceitos E Fundamentos

Conceitos Gerais

Cobertura de Testes (*code coverage*): os testes podem ser qualificados quanto a sua extensão e escopo. A cobertura é o limite de incidência do teste, podendo atingir os seguintes níveis de abstração: negócio (teste de aceitação), requisitos (teste de sistema), projeto (teste de integração) e código (testes unitários);
DevOps: é relação de interdependência entre os desenvolvedores de *software* e os profissionais de tecnologia da informação que viabilizam a operação do *software* e necessidades afins por meio de um sistema de produção e integrado de pessoas, informação, ativos de hardware, serviços, ferramentas e aplicativos, que apoiam as atividades de gestão de projetos, produção e gerência de requisitos de *software*, construção de *software*, prototipação de interfaces gráficas de usuário, construção e disponibilização do banco de dados, testes automatizados, inspeção automatizada da qualidade dos produtos, integração contínua dos componentes de *software*, disponibilização automática do *release* em ambiente apropriado, gestão informatizada dos ambientes de disponibilização, gestão de configuração de artefatos de engenharia e gestão de mudanças. Todas estas atividades de engenharia de *software* são desenvolvidas por meio de processos, métodos e ferramentas, alguns destes são informatizados, promovendo o princípio da impessoalidade e a diminuição do esforço da execução pela automação de processos operacionais. Os profissionais especializados em *DevOps* são responsáveis pela sua instalação, configuração, disponibilização, operação, conectividade, continuidade, segurança e manutenção. O *DevOps* exige comunicação estreita entre os profissionais, colaboração contínua e cooperação máxima (Scaled Agile

Framework, 2014), (SWEBOK, 2004);
Veja mais informações em:
http://guide.agilealliance.org/subway.html
http://www.scaledagileframework.com/devops/

Débito técnico: pendências técnicas introduzidas no código em virtude de o mesmo ter sido implementado sem o design ou cuidados necessários. Estas pendências podem estar relacionadas com: qualidade, bom design que permita fácil adaptação e manutenção, testes, etc. A existência de débito técnico indica que a tarefa não foi totalmente concluída e que a equipe deverá ter mais trabalho futuramente em virtude destas pendências deixadas no projeto.
Veja mais informações em:
http://martinfowler.com/bliki/TechnicalDebt.html
http://martinfowler.com/bliki/TechnicalDebtQuadrant.html http://www.ontechnicaldebt.com/

Defeito: problema no *software* que faz com que ele se comporte de forma distinta da esperada/desejada;

Desvio ou Impedimento: é todo problema que afeta o ritmo de trabalho da equipe do projeto, podendo comprometer o alcance da meta estabelecida para a Sprint ou *Release*;

Definição de Pronto: é um entendimento compartilhado entre os envolvidos no projeto do que significa trabalho completo, assegurando a transparência. O "pronto" será utilizado para definir um nível de maturidade e ou qualidade de entregáveis em cada grupo de atividades do processo. Por exemplo: artefatos de planejamento, requisitos, incremento de software (sprint) e versão funcional do software (release);

Épico: é uma necessidade de negócio não refinada, correspondendo a algum caso de negócio. O épico do produto ou do produto-parte de uma solução informatizada (de um determinado problema) será decomposto em características macro funcionais que o refinam. A solução deve agregar valor

perceptível ao negócio do cliente. (*Scaled Agile Framework*, 2014);

Funcionalidade: é a menor parte e mais elementar de um sistema informatizado, que agrega algum valor ao usuário. É uma história de usuário conforme definido pelo *framework* XP e que, para o Processo Unificado, seria equivalente à decomposição de um caso de uso ou mesmo à descrição do cenário de um caso de uso;

História de usuário (user story): descrição curta de uma característica do produto contada na perspectiva do usuário, utilizando uma linguagem comum ao negócio.

Impact Mapping: técnica que auxilia a equipe a entregar *software* focado em objetivos de negócio, nas partes interessadas e suas necessidades.

Incremento de *Software*: acréscimo de funcionalidades do produto de *software* final que é gerada a cada sprint. O incremento de *software* deve ser algo observável e, preferencialmente funcional, para que o product owner e demais usuários possam experimentá-lo da forma mais realista possível;

Inspeção: Todos os aspectos do processo de entrega que possam impactar o resultado final do projeto devem ser inspecionados frequentemente, para que qualquer variação prejudicial possa ser identificada e corrigida o mais rápido possível.

Veja mais informações em:

http://guide.agilealliance.org/guide/iteration.html

Meta: breve descrição em função dos objetivos de negócio e prazo que a equipe pretende alcançar durante um *release* ou sprint;

Produto: resultado final de um projeto, cuja finalidade é responder a demandas associadas ao *software*, ajustes na organização, necessidade de um ou mais clientes, avanços tecnológicos ou obrigatoriedades legais. O termo produto está relacionado a um produto físico, serviço ou associação de ambos. Independente da

origem da demanda, um produto sempre é destinado a um cliente ou grupo de clientes, internos ou externos (Scrumex, 2014).

Refatoração: é o processo de alterar um software de uma maneira que não mude o seu comportamento externo e ainda melhore a sua estrutura interna. Ela é utilizada para manter um software bem projetado mesmo com o decorrer do tempo e as mudanças que ele virá a sofrer.

Refinamento: é uma técnica para aprimorar o *backlog* do produto por meio dos seguintes procedimentos: a descoberta de novos itens, assim como alteração e remoção de itens antigos; quebra de histórias muito grandes; priorização dos itens do *backlog* (trazendo os mais importantes para o topo); preparação dos itens mais importantes para a próxima reunião de planejamento; estimativa dos itens do *backlog*; inclusão de critérios de aceitação.

Refinamento dos Requisitos: considerando os aspectos do desenvolvimento ágil, define-se refinamento dos requisitos (retrabalho) como sendo os refinamentos referentes a funcionalidades que estão em desenvolvimento dentro de uma mesma release, ou seja, mudanças entre as iterações do release. Anexo V apresenta um quadro sobre itens de tipos de alterações em funcionalidades em métodos ágeis;

Release: conjunto de funcionalidades extraídas do *backlog* do produto. Representa a entrega de parte ou tudo de um *software* funcional incluindo funcionalidades de uma ou mais Iterações;

Requisitos Funcionais: é a descrição do comportamento e a informação que a solução gerenciará. Descrevem capacidades que o sistema será capaz de executar em termos de comportamentos e operações – ações ou respostas específicas de aplicativos de tecnologia da informação (BABOK, 2011), (IEEE610, 1990);

Requisitos não funcionais: são condições que não se relacionam diretamente ao comportamento ou funcionalidade da solução, mas descrevem condições ambientais sob as quais a solução deve

permanecer efetiva, ou qualidades que os *softwares* precisam possuir. Também são conhecidos como requisitos de qualidade ou suplementares. Podem incluir requisitos relacionados à capacidade, velocidade, segurança, disponibilidade, arquitetura da informação e apresentação da interface com o usuário (BABOK, 2011);

Scrum: é um framework de desenvolvimento ágil de *software* iterativo e incremental para o gerenciamento de desenvolvimento de produtos (Guia do *Scrum*, 2003);

Veja mais informações em:

http://guide.agilealliance.org/subway.html

***Sprint*:** um ciclo de desenvolvimento, de duração fixa e curta, que corresponde a uma subdivisão do release e que produz uma versão estável e executável do produto de *software*;

***Stakeholder*:** indivíduo ou organização que tem um direito, ação, declaração ou interesse no *software* em desenvolvi- mento; envolvido; interessado (PMBOK, 2004);

Teste de Aceitação: teste escrito sob a perspectiva do cliente [BECK04]. Esse tipo de teste, geralmente, especifica o resultado esperado após a execução de determinada funcionalidade (IEEE610, 1990);

Teste de Desempenho: teste realizado para avaliar a adequação de um sistema ou componente com determinados requisitos de desempenho (IEEE610, 1990);

Teste de Integração: encontrar falhas provenientes da integração interna dos componentes de um *software*, ele conduz ao descobrimento de possíveis falhas, erros e defeitos associados à interface do *software*. A critério da gerência de projetos, os testes de comunicação entre interfaces com outros *softwares* (integração entre eles) poderão ser realizados (IEEE610, 1990);

Teste de Sistema: o objetivo é executar as funcionalidades do sistema sob a ótica de quem utilizará, explorando as

funcionalidades em busca de inconformidades em relação aos objetivos e especificações funcionais. Os testes são executa- dos em condições operacionais similares, por exemplo, ambiente, interfaces sistêmicas e massas de dados, àquelas que um usuário utilizará na sua rotina de operação do *software* (IEEE610, 1990);

Teste de Software: consiste numa verificação dinâmica do comportamento de um programa em um conjunto finito de casos de teste contra o comportamento esperado. (SWEBOK, 2004), (IEEE610, 1990);

Teste Unitário: teste escrito sob a perspectiva do programador [BECK04], aplicado sobre a menor unidade de execução do código. Cada caso de teste deve avaliar um possível comportamento do código (IEEE610, 1990);

***Time-box*:** a tradução literal é "caixa de tempo" e deve ser entendido como um intervalo de tempo para realização de uma atividade. É uma técnica utilizada para assegurar que atividades importantes do projeto sejam cumpridas e com o menor desperdício possível;

Veja mais informações em:

http://guide.agilealliance.org/guide/timebox.html

***User story mapping*:** técnica para priorização de histórias de usuários por meio de sua organização em interações significativas para o usuário final.

Principais Artefatos:

***Backlog*:** tem por objetivo transformar o trabalho em andamento visível em um quadro para toda equipe, criando um sinal visual que aponta as informações mais relevantes para a execução, como a quantidade de tarefas, suas prioridades e atribuições.

***Backlog* da Sprint:** é um subconjunto do *backlog* do produto, priorizado na reunião de planejamento pelos clientes e product owner e que representa os itens que serão desenvolvidos em um

determinado Sprint;

***Backlog* do Produto:** é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. O *backlog* do produto lista todas as características, funções, requisitos, melhorias e correções que formam as mudanças que devem ser feitas no produto;

Casos de Testes: Descreve um conjunto de condições usadas para testar um software. Contém as seguintes informações:

> **Resumo:** Contém uma descrição do caso de teste, descrevendo a finalidade ou o objetivo do teste e o escopo.
> **Pré-condições:** Para cada condição de execução, descreve o estado obrigatório do sistema antes do início do teste.
> Entradas:
> **Ação:** Para a execução do teste, são as ações que o usuário deve fazer para que o sistema possa cumprir com o que será testado.
> **Resultados Esperados:** É o estado resultante ou as condições observáveis esperadas como resultado da execução do teste. Observe que isso pode incluir respostas positivas e negativas (como condições de erro e falhas).
> **Pós condições:** Para cada condição de execução, descreve o estado ao qual o sistema deverá retornar para permitir a execução de testes subsequentes.

Critérios de Aceitação: Documento que contém os critérios técnicos e de negócio que estabelecem a regra para a qual cada história de usuário ou funcionalidade deve satisfazer para ser aceita pela área de qualidade e de Negócio;

Documento de Visão: documento que contém as informações identificadas durante o entendimento da demanda, tais como objetivos de negócio, características-chaves, aspectos tecnológicos e riscos, utilizados para orientar o desenvolvimento do produto de *software*;

Gráfico *Burndown*: gráfico que mostra o esforço restante para a conclusão da sprint, bem como mostrar o quão próximo ou distante a equipe de desenvolvimento está de atingir a meta;

Ordem de Serviço: Documento contratual elaborado pelos executores e fiscais formais do contrato de acordo com as regras

estabelecidas pela IN 04/2016 que contém informações relativas ao prazo, condições, entregas e equipe que compõe a equipe de desenvolvimento do projeto, em especial os perfis técnicos da Empresa Contratada, quando o desenvolvimento envolver equipe terceirizada;

Plano da *Release*: descreve informações importantes do *release*, tais como: metas do *release*, quantidade e duração de Iterações, equipe do projeto, data estimada da entrega, valor estimado do *release*, premissas, riscos e impedimentos;

Plano da Sprint: descreve informações importantes da Sprint, tais como: Meta da Sprint, data inicial e final da Sprint, definições de pronto, itens de *backlog* selecionados e outros;

Plano de Integração: descreve os itens de backlog a serem disponibilizados em ambiente de homologação e produção.

Relatório de Execução da Sprint: Relatório construído pelo Líder Técnico que apresenta um resumo dos resultados alcançados pela equipe ágil durante a sprint, além aprovações ou recusas de incrementos pelo PO.

Relatório de Testes: Relatório apresentado pela equipe de desenvolvimento da Contratada, quando for o caso, que consolida as informações dos testes realizados nos incrementos de software originados pelas histórias de usuário criadas durante a sprint;

***Roadmap*:** é uma visão global das necessidades do produto e uma ferramenta valiosa para o planejamento e organização da jornada de desenvolvimento de produtos. O product owner cria o *roadmap* de produtos, com a ajuda da equipe de desenvolvimento. O roteiro é usado para categorizar os requisitos, para priorizá-los, e para determinar um calendário para a sua release.

Anexo II – Roteiro para Estimativa Ágil em Projetos de Software

Introdução

Em sintonia com as melhores práticas da comunidade ágil, o Processo de Desenvolvimento de Software da PGDF (PDS-PGDF) preconiza a contação de histórias como modo de descobrir, elucidar e explicitar as necessidades de negócio para as quais há que se prover solução de software. Com efeito, narrativas são um modo milenar de construir e partilhar sentido tanto sobre os problemas do cotidiano quanto sobre grandes acontecimentos e se mostraram, na prática, capazes de estimular uma cultura onde flui a comunicação entre atores de origem e formação distintas, como quase sempre é o caso em projetos de software. A ferramenta que difundiu esta forma de solucionar problemas é a 'história de usuário' (*user story*). Com uma estrutura simples, que pode rapidamente ser dominada por todos os envolvidos, ele exerce a função de capturar o requisito de software desde a perspectiva do usuário, colocado - princípio do ágil - como protagonista.

Por privilegiar a perspectiva do usuário final, os atores passam a compartilhar não apenas uma compreensão do domínio de negócio, com suas regras específicas, mas, sobretudo, do valor de negócio. Enquanto especificações verbosas e detalhistas buscam exaurir o domínio, as histórias de usuário assumem, ao contrário, que mesmo os especialistas de negócio possuem algum nível de ignorância e, portanto, uma exploração conjunta do problema, sempre orientada pela busca do valor, permitirá a construção de soluções eficientes e inovadoras. É assim que o cerne da ferramenta está em servir

como um lembrete, um ponto de apoio, sobre um tema que precisa ser discutido pela equipe. Durante as cerimônias de planejamento da sprint e ao longo do desenvolvimento, busca-se estimular os clientes e potenciais usuários finais a narrar e detalhar o que desejam alcançar com a funcionalidade em construção, esclarecendo contexto, metas e critérios de aceitação. Deste modo, mais que uma estrita relação contratual do tipo cliente-fornecedor que seria sustentada na especificação de requisitos, o que emerge é uma compreensão coletiva sobre o que são e como se relacionam os requisitos que condicionam o valor de negócio perseguido.

Ao negócio, no entanto, não interessa apenas lograr a liberação de um produto, mas um produto dentro de certas restrições, como tempo, orçamento e prazos legais: um projeto será visto como promissor, via de regra, quando for capaz de entregar valor a um certo custo de oportunidade. Sendo assim, há uma inversão na perspectiva, na medida em que os stakeholders do projeto precisam também compreender, sob o ponto de vista dos desenvolvedores, qual o esforço e complexidade envolvidos na viabilização das funcionalidades que compõem o produto. Dimensionar e estimar projetos de software são atividades que perpassam desde a fase de planejamento até o gerenciamento, mas não prescinde, naturalmente, da escolha de métricas capazes de subsidiar os processos decisórios com indicadores. Erros de estimativa podem, e eventualmente levam, a decisões erradas que, no limite, farão fracassar todo o empreendimento. Os gestores possuem uma intuição disto e, fiéis a ela, podem se obsessionar com a construção de estimativas precisas. Ora, estimar possui um custo, demanda tempo, e, ensinou Mike Cohn (2006: p. 50), também se sujeita à restrição dos rendimentos decrescentes. Ou seja, é uma tarefa que, de início, permite um ganho de acurácia alto relativamente ao esforço dispendido, mas apresenta taxas decrescentes após

certo ponto e, ao fim, torna-se detrimental ao projeto, pois os ganhos de acurácia são baixos e logo decaem em resposta a um aumento o do esforço:

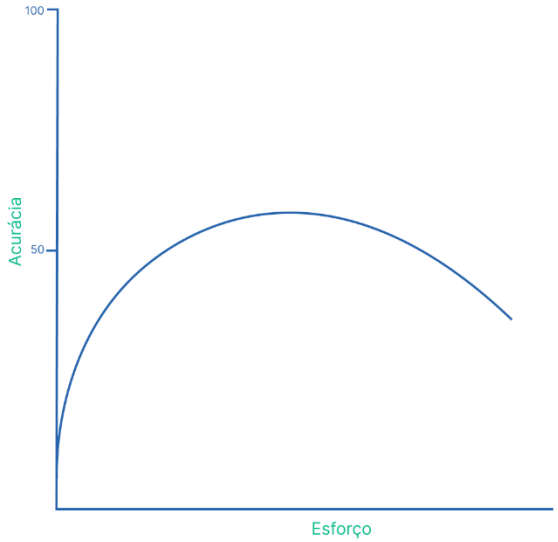


Figura 12. O esforço adicional de estimativa proporciona muito pouco valor além de um certo ponto.

O que a figura anterior evidencia é que a estimativa não é meramente uma questão da melhor métrica, mas de qual o melhor processo em termos de tempo e acurácia, no que se deve levar em consideração desde as características do domínio de negócio até a cultura organizacional, a composição das equipes, o parque tecnológico, as práticas de desenvolvimento, dentre outros fatores. A escolha do modelo ágil no PDS-PGDF sinaliza o reconhecimento da incerteza como um dado insuperável, mas, ao mesmo tempo, não despreza a lição de que se pode alcançar um grau de acurácia a um esforço relativamente baixo. Times ágeis "reconhecem que não podemos eliminar a incerteza das

estimativas, mas abraçam a ideia de que pequenos esforços são recompensados com grandes ganhos[2]" (COHN, 2006: p. 51).

Além desta breve introdução, este roteiro é composto pela apresentação dos Objetivos da Estimativa (2), onde são analisadas as razões pelas quais uma estimativa ágil pode e deve estar alinhada com os objetivos organizacionais. Há que se constituir uma verdadeira estratégia de metrificação, capaz de agregar valor ao processo de desenvolvimento e de contribuir para o alcance dos objetivos gerais e específicos prementes ao domínio de negócio e aos projetos em execução. No tópico Métricas Ágeis (3), sem a intenção de exaurir ambiente tão diverso, serão discutidos os dois tipos mais generalizados de métricas, aquelas baseadas em esforço (3.1) e em velocidade (3.2), buscando-se mostrar que há complementaridade entre elas e justificar, deste modo, a decisão deste roteiro (3.3 - Mensuração de Projetos de Software na PGDF) de utilizar *story points* como métrica básica para definição da velocidade das equipes. Avança-se, então, para o tópico específico das Técnicas de Estimativa (4) que, respeitadas as características de maturidade dos mensuradores e as especificidades de negócio, define as técnicas de metrificação a serem utilizadas na PGDF. Por fim, o tópico final apresenta os Templates de Histórias de Usuário (5) que, em consonância com o PDS-PGDF, servem como cardápio básico à mão das equipes.

1. Objetivos da Estimativa

A estimativa não é um fim em si mesma. Antes, trata-se de uma atividade que, por meio de certas técnicas e orientada por

2 "They [agile teams] acknowledge that we cannot eliminate uncertainty from estimates, but they embrace the idea that small efforts are rewarded with big gains".

algum método de raciocínio, responde a perguntas bem definidas. O que se busca é fornecer informações valiosas para o planejamento, controle, tomada de decisões, comunicação e melhoria contínua dos processos técnicos e negociais. As métricas são ferramentas essenciais para uma gestão eficaz e uma melhor compreensão do esforço e dos recursos envolvidos no desenvolvimento de software. Neste sentido, realizar estimativas de software ao longo de todo o processo de desenvolvimento atende a objetivos diversos:

1. Planejamento e gerenciamento de projetos, pois fornecem uma base factual mínima a partir da qual os gestores podem alocar recursos, definir prazos e estabelecer metas realistas. As métricas ajudam, assim, a identificar o esforço necessário para concluir as tarefas e avaliar a viabilidade do projeto.
2. Controle de custos e recursos, pois o monitoramento do progresso do projeto subsidia eventuais necessidades de realocação de recursos e priorização de atividades alternativas em vistas de dados concretos sobre alterações nos custos de oportunidade do projeto.
3. Estimativas precisas implicam decisões informadas. Por possuírem dados precisos, os gestores podem avaliar diferentes opções, priorizar atividades, ajustar recursos e tomar decisões estratégicas durante o desenvolvimento do software.
4. As métricas podem ser utilizadas para construir um ambiente de trabalho transparente, que incentiva e recompensa a comunicação e a colaboração. Ter uma base de estimativas clara e que leve em consideração o ponto de vista de todos os atores envolvidos promove uma melhor compreensão e compartilhamento das atividades envolvidas no projeto.

5. Identificação de problemas e riscos, tais como atrasos significativos em relação ao cronograma planejado, viabilizando, deste modo, a identificação tempestiva de impedimentos e a adoção antecipada de medidas corretivas.
6. Métricas bem definidas promovem a melhoria contínua. Com a habilidade de identificar o que se passa ao longo de todo o desenvolvimento do projeto, padrões, tendências e áreas de melhoria contínua se tornam objeto de discussão pela equipe e pelos gestores. Isso permite ajustes de processo, eliminação de ineficiências, ganhos de produtividade e aprimoramento do ambiente corporativo.

Nota-se que as métricas podem servir a objetivos muito diferentes, conforme a etapa do projeto, a pergunta que se pretende responder e, não menos importante, os padrões e metodologias que regem a gestão. Uma métrica focada em atores externos à equipe e que pretende fornecer insumos, por exemplo, de controle de produtividade pode não ser o melhor dado se o problema de gerenciamento for relacionado à qualidade do produto. Consequente com seus princípios, e por não ser uma metodologia fechada, o Scrum não exige a adoção de métricas específicas, preferindo, ao contrário, focar na diversidade de objetivos e em como estes se relacionam. Em interessante estudo que passou em revista a literatura mais recente sobre métricas no Scrum, Kurnia et al. (2018: p. 175) agruparam estes objetivos da seguinte forma:

Figura 13. Objetivo de medição de software

Neste sentido, o PDS-PGDF requer um processo de mensuração capaz de responder a perguntas de atores internos e externos ao projeto e que, a par de uma visão de qualidade focada em pessoas e no valor de negócio, seja instrumental ao alcance de metas e objetivos estratégicos. A métrica de software, portanto, não é meramente uma questão de determinação da quantidade de software por tempo, mas de produção de valor, segundo critérios de qualidade e *compliance*, de modo a contribuir na missão institucional da PGDF e, finalmente, alcançando o bem público.

2. Métricas Ágeis

Por não adotar uma visão antagonista quanto à incerteza e a mudança, a cultura ágil tende a se distanciar de métricas que, a despeito da promessa de objetividade, demandam um esforço de

produção alto e, sobretudo, são divorciadas da perspectiva dos atores envolvidos. Em extensa revisão sistemática de literatura com objeto em publicações sobre as métricas efetivamente utilizadas na indústria de software, Kupianen *et al.* (2015: p. 150) encontraram uma alta frequência, 39%, de métricas não mencionadas nos trabalhos originais sobre métodos ágeis e que são utilizadas na prática. Na visão dos autores, isto indica, por um lado, que os praticantes do ágil adicionam e inventam novas métricas de acordo com suas necessidades e, por outro, que a diversidade de métricas compatíveis com os princípios do ágil é maior do que aquela encontrada nas obras basilares.

Não obstante, tanto em termos quantitativos quanto qualitativos, o mesmo estudo, em linha com o mais recente de Kurnia *et al.* (2018: p. 177), indica uma predominância e influência altas das métricas focadas em velocidade e esforço. Isto se explica pela necessidade de rastrear a capacidade de entrega de valor das equipes ágeis, mas também deriva de certas características da metodologia. Por exemplo, conquanto o monitoramento de erros e a satisfação do cliente sejam centrais e, de fato, muitas indústrias construam métricas para acompanhá-las, supõe-se que o ágil não tolere entregas que não agregam valor para o negócio nem entregas com baixa qualidade. Neste sentido, já estariam monitorados na própria definição de “pronto” e, por conseguinte, a velocidade de entrega tornaria visível.

Como já ressaltado, a metrificação é também uma atividade que exige esforço e, não menos relevante, experiência. A imprevisibilidade em que toda estimativa está imersa diminui com o tempo - não só do projeto, mas sobretudo dos profissionais. Assentado este fato, convém, no aprimoramento do PDS-PGDF, começar pelo estruturante, reagindo às mudanças e

necessidades de adaptação em momento oportuno. Portanto, a partir de métricas de esforço (3.1) e velocidade (3.2), este roteiro preconiza a adoção de uma métrica que conjugue estes dois pilares (3.3).

2.1 - Métricas baseadas no esforço

Expostas na clássica obra de Cohn (2005), as duas métricas mais relevantes de esforço são "dias ideias" e "*story points*". Embora possuam objetivo semelhante, qual seja, estimar um cronograma a partir do esforço necessário para executar as tarefas associadas a uma história de usuário, elas diferem quanto à estratégia.

Os dias ideias procuram identificar o tempo efetivamente dedicado ao trabalho, desconsiderando interrupções, reuniões, pausas e outros fatores que possam afetar a produtividade. Esta abordagem supõe que a produtividade da equipe não é constante ao longo do dia de trabalho e que existem interrupções e atividades não relacionadas ao desenvolvimento que podem diminuir o tempo real disponível para trabalho efetivo. Portanto, ao fazer estimativas por dias ideais, considera-se apenas o tempo em que a equipe está totalmente focada e produtiva no desenvolvimento de software.

A abordagem dos dias ideais busca tornar as estimativas mais realistas e evitar superestimar a capacidade de entrega da equipe. Ademais, por se concentrar no tempo efetivamente dedicado ao trabalho, estas estimativas tendem a ser bastante intuitivas tanto para os atores internos quanto para pessoas de fora da equipe. No limite, cada item será estimado a partir de uma

visão global do total de horas (ideais) necessárias à equipe para concluí-lo, não do tempo real necessário, o que pode também se revelar um indicador para outros fatores que, eventualmente, estejam afetando a produtividade e dedicação da equipe.

Story points, por sua vez, são uma unidade de medida relativa que busca estimar o **esforço** necessário para concluir uma história de usuário, uma tarefa ou uma funcionalidade em um projeto de desenvolvimento de software. Os story points são uma maneira de apreciar a complexidade, o trabalho envolvido e outros fatores relevantes para a implementação de uma determinada atividade. A mensuração de tamanhos relativos por meio de uma escala ordinal é um método, ainda que relativamente recente no desenvolvimento de software (popularizado a partir dos anos 2.000), de consistência secularmente demonstrada em outras áreas do conhecimento, como as ciências econômicas. Assim como os marginalistas perceberam que não era necessário uma medida objetiva e cardinal para quantificar as preferências e utilidades individuais, os agilistas compreenderam que não era preciso abstrair a subjetividade inerente ao esforço de desenvolvimento para se chegar a uma medida deste mesmo esforço. Uma escala ordinal permite a ordenação e comparação de elementos com base em suas posições relativas, sem atribuir valores numéricos absolutos.

Assim, que uma história possua certa quantidade de pontos não possui qualquer significado em termos numéricos absolutos, antes quer dizer sua posição no interior de uma escala ordinal que expressa o esforço relativo à execução do que a história propõe. Com efeito, ao atribuir story points, as equipes ágeis consideram diversos aspectos, como a complexidade técnica, o nível de incerteza, o risco, a dependência de outros componentes do sistema, etc., todos fatores que ao final indicam um certo nível de esforço necessário para concluir a tarefa. Uma

escala numérica habitualmente é usada para representar os story points, como Fibonacci (1, 2, 3, 5, 8, 13, etc.) ou uma sequência linear (1, 2, 3, 4, 5, etc.). Não importa a unidade de tempo, mas a percepção coletiva da equipe sobre o esforço necessário para concluir a atividade em relação a outras atividades de referência. Por exemplo, se uma tarefa é considerada duas vezes mais complexa que uma atividade de referência com 3 story points, ela pode ser atribuída com 6 story points. Conforme Cohn (2005: p. 38), os story points tornam o cronograma uma função do tamanho dos itens, aferidos segundo sua complexidade relativa.

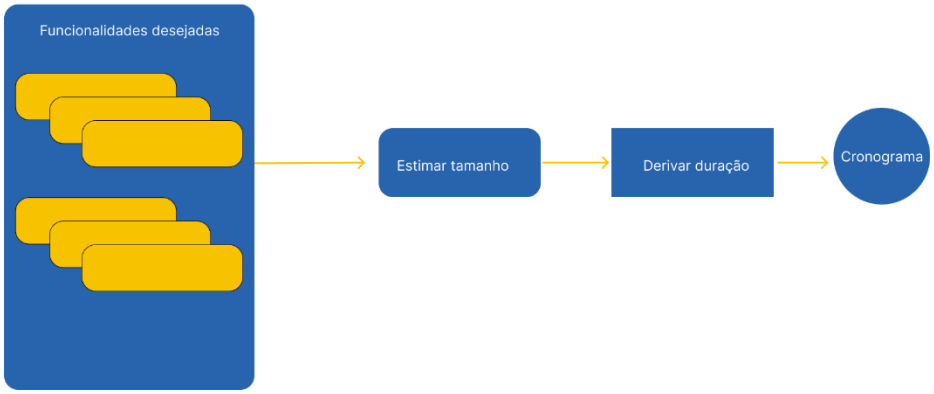


Figura 14. Estimar a duração de um projeto começa com a estimativa de seu tamanho.

Em comparação com os dias ideias, os story points possuem algumas vantagens como métrica de esforço que têm contribuído para sua generalização na indústria de software. Cohn (2005: p. 69) argumentou pela facilidade com que esta métrica impulsiona o comportamento multidisciplinar na equipe, seu maior tempo de validade, o fato de serem estimativas puramente de tamanho, a facilidade de estimar e a dificuldade de

se definir para uma equipe o que seja um dia ideal para todos os indivíduos:

> Os story points têm a vantagem de promover o comportamento multidisciplinar da equipe. Além disso, como os story points são uma medida mais pura de tamanho, eles não precisam ser reestimados se a equipe melhorar em uma tecnologia ou domínio. Estimar em story points é frequentemente mais rápido do que estimar em dias ideais. Por fim, ao contrário dos dias ideais, os story points podem ser comparados entre os membros da equipe. Se um membro da equipe acredita que algo levará quatro dias ideais para ser concluído, e outro membro da equipe acredita que levará um dia ideal, ambos podem estar corretos, mas não há base para argumentar e estabelecer uma estimativa única[3] (COHN, 2005: p. 74).

Por tais razões, recomendamos a utilização de *story points* como métrica de esforço nas estimativas de projetos de software da PGDF. Ademais, é uma métrica que, consoante a ampla difusão no mercado privado, apresenta a vantagem adicional em um cenário de terceirização de desenvolvimento da facilidade em encontrar profissionais com algum nível de experiência na métrica.

2.2. Métricas baseadas na velocidade

3 Story points have the advantage of helping promote cross-functional team behavior. Additionally, because story points are a more pure measure of size, they do not need to be re-estimated if the team improves in a technology or the domain. Estimating in story points is often faster than estimating ideal days. Finally, unlike ideal days, story points can be compared among team members. If one team member thinks something will take her four ideal days, and another team member thinks it will take him one ideal day, they may each be right, yet they have no basis on which to argue and establish a single estimate.

Métricas baseadas em velocidade são usadas no desenvolvimento ágil de software para medir o ritmo de trabalho da equipe ao longo do tempo. Elas fornecem uma medida quantitativa da capacidade da equipe de concluir as histórias de usuário ou tarefas em cada iteração ou sprint. Essas métricas ajudam a prever o progresso futuro, estimar prazos e facilitar o planejamento do projeto.

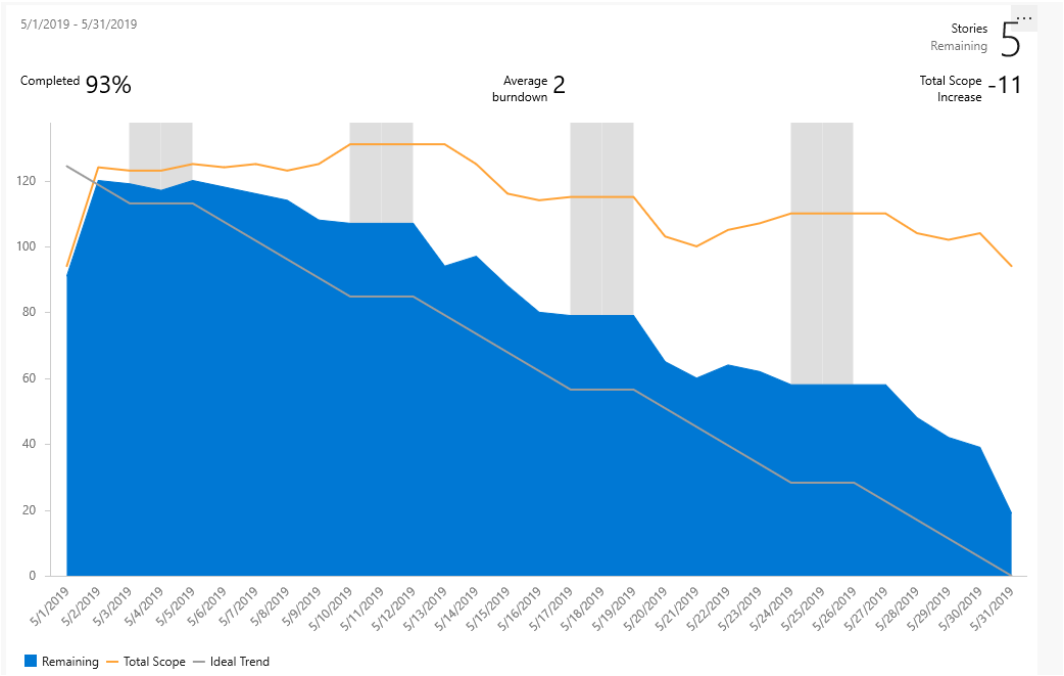
A principal métrica baseada em velocidade é a "velocidade da equipe" ou "story points concluídos". Ela representa o número de pontos de história (ou outras unidades de tamanho) que a equipe é capaz de entregar em uma iteração específica. A velocidade da equipe é calculada somando-se os story points concluídos em cada iteração e obtendo uma média ao longo de várias iterações. Essa média se torna uma referência para estimar a capacidade de trabalho da equipe em iterações futuras. Por exemplo, se a equipe concluiu em média 20 pontos de história nas últimas três iterações, pode-se estimar que ela terá uma velocidade média de 20 pontos de história para as próximas iterações.

Além da velocidade da equipe, outras métricas relacionadas podem ser utilizadas, como a taxa de conclusão de histórias, que mede a porcentagem de histórias de usuário concluídas em relação ao total planejado para uma iteração. Essa métrica ajuda a avaliar o progresso em relação às metas estabelecidas.

As métricas baseadas em velocidade no desenvolvimento ágil permitem que a equipe e os gestores acompanhem o desempenho ao longo do tempo, identifiquem tendências, ajustem as estimativas e realizem um planejamento mais preciso. No entanto, é importante lembrar que a velocidade da equipe pode variar de iteração para iteração, dependendo de vários

fatores, como mudanças na equipe, complexidade das histórias de usuário e interrupções. Portanto, as métricas de velocidade devem ser usadas como uma referência e adaptadas às circunstâncias específicas de cada projeto.

A estimativa de velocidade responde perguntas relevantes para a gestão, relativas à produtividade da equipe e ao cronograma de recursos e projetos, mas também responde a perguntas da própria equipe, que pode, por meio de artefatos como o gráfico de burndown, acompanhar em tempo real seu próprio ritmo e atuar para realizar correções. Neste sentido, metrificar a velocidade deve também ser compreendida como uma prática destinada a motivar pessoas e a perseguir um ritmo de trabalho sustentável, que coloque as pessoas em primeiro lugar.



A adoção de uma métrica de velocidade é, portanto, consequência direta e complementar às métricas de esforço. Mais

do que isto, ela permitirá às equipes da PGDF ajustarem suas atribuições de *story points* ao longo do tempo, aprimorando sua capacidade de atribuir ao esforço algum nível de intuição sobre a duração da execução.

2.3 - Mensuração de Projetos de Software na PGDF

A literatura sobre métricas de software demonstra que a difusão dos *story points* é função direta da efetividade da métrica em responder aos objetivos de estimativa. Com efeito, em que pese sua natureza essencialmente empírica, os times que a utilizam tendem a convergir, num curto espaço de tempo, para estimativas próximas do observado. Aranha & Cardoso (2017: p. 59), em estudo comparativo com o ponto de função, encontraram uma convergência a partir da terceira sprint. Como métrica angular das estimativas da área de software da PGDF, assim, recomenda-se a aferição por *story points*. Quanto à métrica de velocidade, esta deve ser definida, ao mesmo tempo, como o número de histórias de usuário e de story points concluídos por sprint. Esta métrica deve ser expressa em um gráfico de *burndown,* visível para todos os *stakeholders* do projeto de forma constante. Devem os membros da equipe, portanto, zelar pelo registro tempestivo no quadro do projeto de forma a propiciar a coleta de dados precisos e que reflitam o ritmo de trabalho observado na prática.

3. Técnicas de Estimativa

A escolha da técnica de estimativa deve respeitar o princípio central de que as estimativas são compartilhadas pela

equipe. Neste sentido, deve-se evitar a técnica da estimativa de julgamento pelo *expert*, pois, de um lado, o planejamento ágil não se baseia na mera atribuição de tarefas (os times são auto-organizados) e, por outro, a opinião de todos os membros deve, necessariamente, ser levada em conta - o esforço que se busca aferir é da equipe. O segundo passo é a determinação da escala de estimativa. Recomenda-se a escolha de escalas com apenas uma ordem de magnitude e que possuam uma lógica interna facilmente compreensível por todos os participantes.

Na sequência são apresentadas as técnicas e escalas que estão à disposição das equipes.

3.1 Planning Poker

O Planning Poker é uma técnica colaborativa usada para estimar o esforço necessário para concluir uma tarefa, história de usuário ou funcionalidade. Na prática do Planning Poker, cada membro da equipe recebe um conjunto de cartas com valores numéricos representando os story points, como as cartas da sequência Fibonacci (mais recomendável). Em uma sessão de estimativa, a equipe discute a tarefa e, em seguida, simultaneamente, cada membro seleciona a carta que melhor reflete sua estimativa individual. Em seguida, as cartas são reveladas e as estimativas discrepantes são discutidas para buscar um consenso.

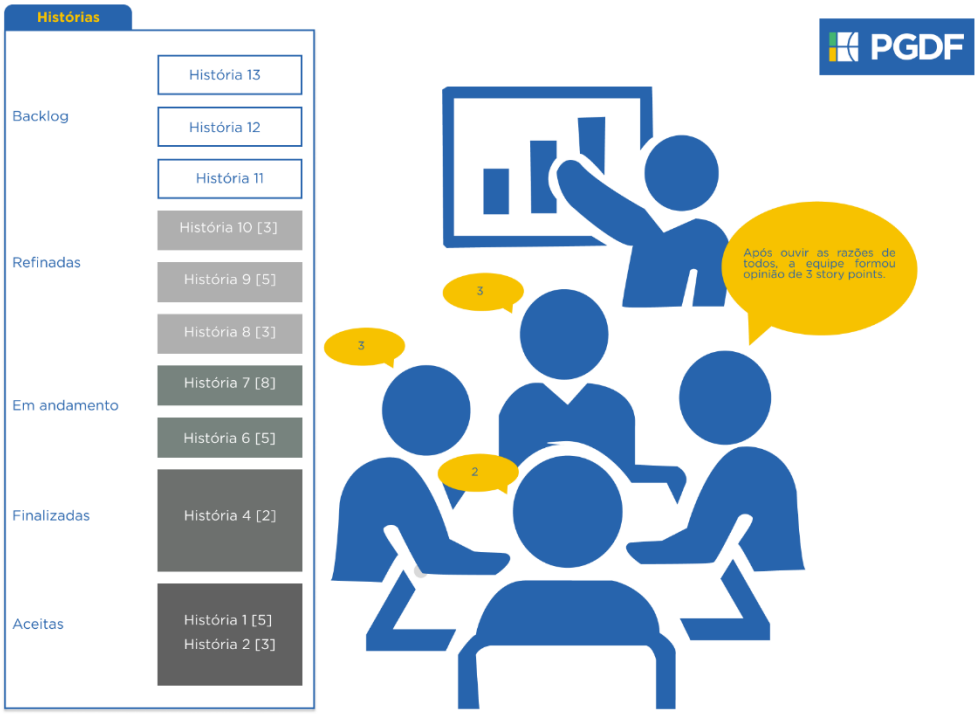


Figura 15. Planning Poker

Esse método de estimativa baseado em comparações relativas foi adotado como uma forma mais ágil e colaborativa de estimar o esforço envolvido em uma atividade de desenvolvimento de software, em contraste com as estimativas baseadas em horas ou dias. A técnica do Planning Poker ganhou popularidade e se tornou um componente essencial do Scrum e de outras metodologias ágeis, evoluindo para a prática de usar story points para estimativas relativas.

3.2 - Tamanho Relativo

Nessa técnica, as histórias de usuário são comparadas umas com as outras, considerando seu tamanho relativo. A equipe atribui um valor de story point às histórias de referência

e, em seguida, compara outras histórias em relação a essas referências, atribuindo-lhes um número de story points correspondente.

3.3 - Estimativa “tamanho de camiseta”

Essa técnica utiliza tamanhos de camisetas (pequeno, médio, grande, extra grande) para representar a complexidade ou esforço necessário para concluir uma história de usuário. A equipe atribui um "tamanho de camiseta" a cada história, com base em sua percepção geral de sua complexidade em relação a outras histórias.

3.4 - Estimativa em Sequência Linear

Nessa abordagem, as histórias de usuário são atribuídas a um número sequencial (1, 2, 3, 4, 5, etc.) com base na percepção da equipe sobre a complexidade e o esforço necessário. Essa técnica é mais simples e direta, mas pode ser menos precisa em comparação com técnicas baseadas em sequências não lineares, como a sequência Fibonacci, uma vez que a distância linear faz com que a soma dos itens facilmente perca o sentido ordinal da sequência.

3.5 - Escalas

Dadas a necessidade de maior precisão de algumas estimativas, a literatura especializada vem explorando formas de mitigar os fatores que podem levar as equipes ágeis a errarem na avaliação dos *story points.* Gaffar *et al.* (2018) propuseram um framework de aprimoramento que pondera os níveis de incerteza e complexidade associados aos requisitos das histórias de usuário. Zahraoui & Idrissi (2015) buscaram compor um fator de

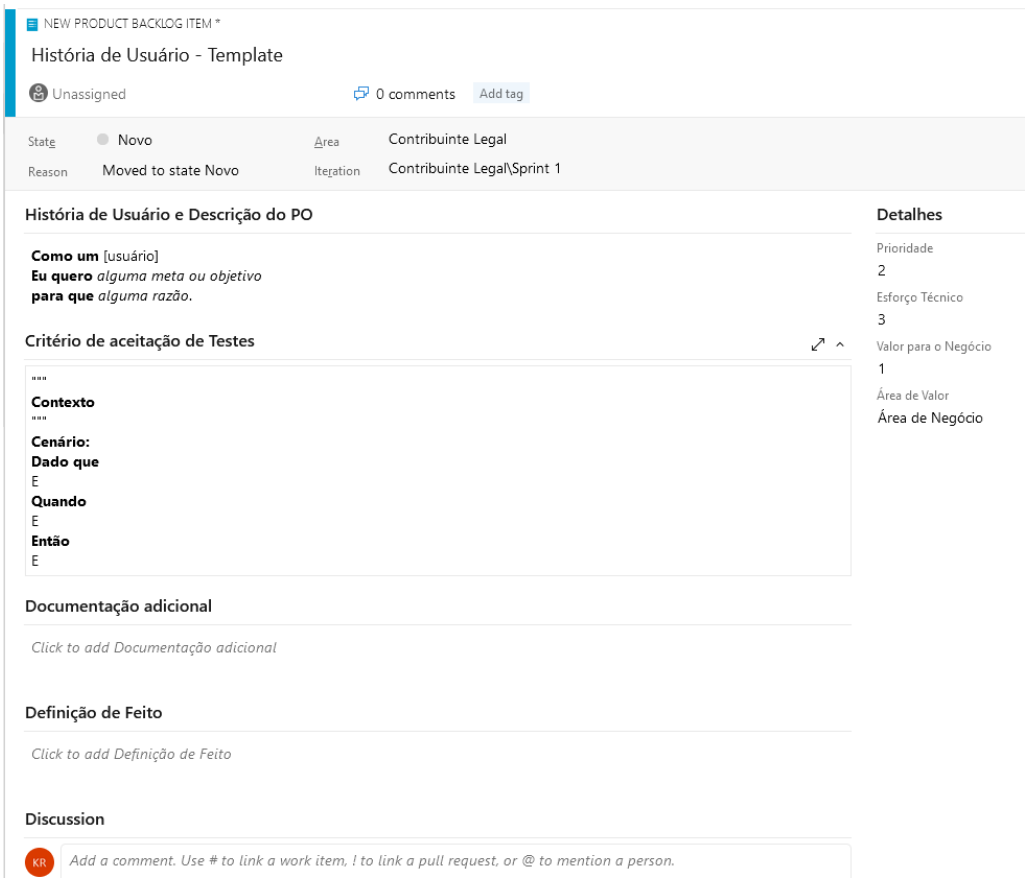
ajuste nas estimativas de tempo e esforço. Sem adotar de forma rígida estes modelos, o presente roteiro adapta suas contribuições em uma escala que pode ser escolhida pelas equipes ágeis nos contextos de dificuldade de aprendizado da métrica ou de erros detectados na mensuração.

Fator de Complexidade	
Complexidade máxima	Extremamente complexo e muito difícil de descrever com precisão
	Muitas dependências de outras histórias, outros sistemas ou subsistemas
	Requer habilidades de programação avançadas que estão ausentes na equipe
Complexidade média-máxima	Muito complexo e difícil de ser descrito com precisão pelo proprietário do produto
	Múltiplas dependências de outras histórias, outros sistemas ou subsistemas
	Requer habilidades de programação avançadas que estão presentes, mas não são fortes na equipe
Complexidade média	A história é moderadamente complexa e um tanto difícil de ser descrita com precisão pelo proprietário
	Número moderado de dependências de outras histórias, outros sistemas ou subsistemas
	Requer habilidades de programação avançadas que estão presentes na equipe
Complexidade média-mínima	A história é um tanto complicada com uma descrição menos clara
	Poucas dependências de outras histórias, outros sistemas ou subsistemas
	Requer habilidades de programação

	intermediárias que estão presentes na equipe
Complexidade mínima	A história é simples, com uma descrição muito clara
	Sem dependências de outras histórias, outros sistemas ou subsistemas
	Requer habilidades de programação básicas que estão presentes na equipe

Incerteza
A visão do produto e a estratégia estão claras?
Os stakeholders chave estão envolvidos no projeto?
As dependências do projeto estão identificadas?
As necessidades de negócio foram levantadas e especificadas?
Os critérios de aceitação da história de usuário estão claros?
Complexidade
Quantos verbos e palavras-chave estão descritos na história de usuário?
Quantas interfaces externas precisam ser consumidas?
Há uma quantidade grande de inputs e outputs nas transações demandadas?
A história de usuário requer desenvolvimento em mais de uma tela?

4. Template de História de Usuário




PGDF