Programa IT Talent
Quality Assurance

TURMA 02	SEMANA 01

ALUNO: José Lisiomar de Souza | **PROF.:** Mike Hill | **DATA**: 12/08/2023

Pesquisa 1 - Sopa de letrinhas da Área de QA

Ágile (Scrum, Kanban, Gráfico de Burndown)	**Ágile** é um conjunto de metodologias para o desenvolvimento de software e gestão de projetos que priorizam a flexibilidade, colaboração e desenvolvimento iterativo. As equipes seguem um ciclo de planejamento, execução e avaliação contínua do produto, onde podem ocorrer reajustes e melhorias, proporcionando uma entrega de qualidade. A metodologia ágil valoriza a interação constante com as equipes envolvidas (incluindo o cliente) e a reavaliação regular das prioridades. O **Ágile** envolve a elaboração de pequenos encontros (ciclos) iterativos, onde pequenas partes do trabalho é executada em intervalos previamente definidos. Isto proporciona uma rápida adaptação aos novos requisitos que podem surgir no decorrer do projeto. Nesse sentido, as equipes possuem autonomia, são colaborativas e capacitadas a tomar decisões independentes de uma hierarquia rígida, como nos métodos tradicionais. Vale ressaltar que a metodologia ágil possui uma abordagem centrada no cliente, e por isso sua implementação requer uma mudança cultural e uma mentalidade aberta à colaboração, promovendo um processo mais dinâmico para o sucesso do projeto. **Scrum** é uma estrutura (framework) ágil para gerenciar e aumentar a eficiência e a flexibilidade no desenvolvimento de produtos e projetos. Baseia-se em princípios colaborativos e iterativos para entregas frequentes e contínua adaptação. O trabalho é dividido em ciclos chamados de "sprints", geralmente com duração de 2 a 4 semanas. Cada sprint envolve planejamento, execução, revisão e retrospectiva, com o objetivo de produzir um incremento funcional do produto ao final de cada ciclo. O Scrum define papéis específicos que são: • **Product Owner:** (dono do projeto): é responsável por definir e priorizar os requisitos do produto. • **Scrum Master:** facilita o processo, remove obstáculos e promove a aderência aos princípios do Scrum. • **Development Team:** responsável por entregar o trabalho planejado durante os sprints. O framework valoriza a comunicação frequente e a colaboração entre esses papéis. Uma característica distintiva do **Scrum** é a reunião diária de acompanhamento chamada de *"Daily Scrum"* ou *"Daily Standup"*, onde os membros da equipe compartilham o que foi feito, o que será feito e quais obstáculos estão enfrentando.

Ágile (Scrum, Kanban, Gráfico de Burndown) Continuação	**Kanban** é uma abordagem de gestão visual e ágil para gerenciar e conduzir o trabalho em diversas áreas de negócio. O método ganhou popularidade em times ágeis de desenvolvimento graças a sua flexibilidade, velocidade e eficiência. Ele baseia-se na utilização de cartões visuais ou sinais para controlar o fluxo de trabalho e melhorar a produtividade operacional. Alguns princípios de destaque do **Kanban** são: • **Visualização de Fluxo de Trabalho:** Quadro visual dividido em colunas que representam as etapas do projeto (a fazer, em andamento e concluído). A cada mudança de status o cartão avança no processo; • **Padronização dos Trabalhos:** A padronização elimina as variações e inconsistências de procedimentos, garantindo a qualidade do trabalho realizado. • **Aperfeiçoamento Contínuo:** Ao identificar oportunidades de otimização através de análise de dados, o método estimula a busca constante por melhorias. • **Feedback e Colaboração:** Uma vez que todos podem visualizar o status atual do trabalho, o Kanban promove a colaboração entre os membros da equipe, além de permitir feedback imediato entre as equipes e seus líderes. • **Foco no Cliente:** Ao reduzir o tempo de entrega e melhorar a qualidade, o método ajuda as equipes a oferecer um produto ou serviço de maior valor para os clientes. **O Gráfico de Burndown** é uma ferramenta visual que mostra o progresso de um projeto em relação ao tempo disponível. Ele pode ser usado por equipes que trabalham com metodologias ágeis, como **Scrum** ou **Kanban**, para acompanhar o trabalho restante e verificar se os prazos serão cumpridos. Ele consiste em dois eixos: o eixo horizontal representa o tempo, geralmente dividido em dias, semanas ou sprints; o eixo vertical representa o trabalho, medido em horas, pontos ou tarefas. O gráfico também possui duas linhas: a linha ideal, que mostra a redução esperada do trabalho ao longo do tempo; e a linha real, que mostra a redução efetiva do trabalho conforme ele é concluído pela equipe. A diferença entre as duas linhas indica se o projeto está no caminho certo ou não. Se a linha real estiver abaixo da linha ideal, significa que a equipe está adiantada e pode terminar o projeto antes do previsto. Se a linha real estiver acima da linha ideal, significa que a equipe está atrasada e precisa acelerar o ritmo para entregar o projeto no prazo. Se as duas linhas se cruzarem, significa que a equipe está alinhada com a expectativa e pode manter o mesmo nível de produtividade.

BDD (Gherkin, Given-When-Then)	**Behavior-Driven Development (BDD)** ou Desenvolvimento Orientado a Comportamento, é uma metodologia de desenvolvimento de software que enfatiza a colaboração entre desenvolvedores, analistas de negócios e testadores, para criar software de alta qualidade que atenda às necessidades do cliente. **Gherkin** é uma linguagem de especificação que permite a descrição de comportamentos de software em uma linguagem compreensível tanto por técnicos quanto por não técnicos. Ele é projetado para ser legível e conciso, promovendo a comunicação eficaz entre os membros da equipe. A sintaxe do Gherkin é bastante simples e usa palavras-chave para estruturar as especificações de comportamento. Usando a estrutura Given-When-Then podemos descrever cenários de teste em BDD da seguinte forma: **"Given"(Dado):** descreve o estado inicial do sistema antes do cenário ser executado. **"When"(Quando):** descreve a ação que é tomada pelo usuário ou pelo sistema. **"Then"(Entao):** descreve o resultado esperado após a ação ser executada. A estrutura **Given-When-Then** é útil para garantir que os cenários de teste sejam claros e fáceis de entender. Ela também ajuda a equipe de desenvolvimento a se concentrar no comportamento do sistema, em vez de nos detalhes técnicos da implementação. Além disso, os cenários escritos em Gherkin podem ser facilmente convertidos em testes automatizados usando outras ferramentas.
BugTracking (Rastreabilidade, Mantis, Jira, Azure DevOPS)	**BugTracking** (rastreamento de bugs) é um processo que ajuda a rastrear e gerenciar bugs ou problemas em um projeto de software. Em geral, o rastreamento de bugs é um elemento essencial para manter a qualidade do software e garantir que problemas sejam identificados, priorizados e corrigidos de maneira eficiente. A **rastreabilidade** é outro aspecto importante do **BugTracking**, pois permite que os desenvolvedores acompanhem a origem de um bug e como ele foi corrigido. Isso pode ser feito vinculando vários objetos, como itens de trabalho, branches, commits, solicitações de pull, builds e versões. Existem várias ferramentas disponíveis para ajudar com o **BugTracking**, incluindo o **Mantis**, **Jira** e **Azure DevOps**. Este, por exemplo, oferece suporte para rastrear o trabalho de requisitos para implantação e fornece *insights* em cada etapa das decisões tomadas e do software implantado. Abaixo, descreveremos resumidamente cada uma dessas ferramentas: **Mantis** é uma ferramenta de código aberto (escrito em PHP) para dar suporte ao processo de gestão de defeitos do projeto. Ele permite que equipes registrem, classifiquem e acompanhem problemas, atribuam responsáveis e definam prioridades. A rastreabilidade é suportada através da vinculação de problemas a tarefas, casos de teste e outras informações relevantes. O download do programa pode ser obtido em **https://www.mantisbt.org/download.php**

BugTracking (Rastreabilidade, Mantis, Jira, Azure DevOPS) (continuação)	**Jira**, desenvolvido pela Atlassian, é uma plataforma ampla que abrange rastreamento de bugs, gerenciamento de projetos e colaboração. Ele oferece recursos avançados de rastreabilidade, permitindo vinculações entre problemas, histórias de usuário, tarefas e outros artefatos. Isso ajuda as equipes a entenderem como os problemas se relacionam com o desenvolvimento geral. Link para a ferramenta: **https://www.atlassian.com/br/software/jira** **Azure DevOps**, da Microsoft, é uma plataforma completa de desenvolvimento que inclui recursos de rastreamento de bugs, gerenciamento de projetos, controle de versão e integração contínua que permite as equipes rastrearem problemas, associando-os a requisitos ou tarefas, e acompanhando seu status até a resolução. Guia da ferramenta pode ser obtido em **https://portal.azure.com/#home**
Testes Automatizados (Cucumber, Specflow, Selenium WebDriver, Appium, Java, C#)	Os testes automatizados são uma prática fundamental no desenvolvimento de software, permitindo a execução rápida e repetível de testes para verificar se um aplicativo ou sistema está funcionando conforme o esperado. Diversas ferramentas estão disponíveis para auxiliar nesse processo, algumas das quais destacamos abaixo: • **Cucumber** e o **SpecFlow** são frameworks que suportam a prática de Behavior-Driven Development (BDD). Eles permitem que os testes sejam escritos em uma linguagem de fácil compreensão, como o Gherkin. Essa abordagem facilita a colaboração entre equipes técnicas e não técnicas e promove uma compreensão compartilhada dos requisitos. Ambos os frameworks são usados para automatizar testes funcionais e podem ser integrados com linguagens de programação como **Java** (**Cucumber - https://cucumber.io/**) e **C#** (**SpecFlow - https://specflow.org/**) • **Selenium WebDriver** é uma das ferramentas mais populares para testes automatizados de interfaces de usuário. Ele permite que os desenvolvedores criem scripts em linguagens como **Java** e **C#** para interagir com aplicativos da web, simulando ações do usuário, como clicar em botões, preencher formulários e verificar resultados. O **Selenium** é amplamente utilizado para testes de integração e aceitação. Ferramenta disponível em **https://www.selenium.dev/documentation/webdriver/** • **Appium** é uma ferramenta de automação de testes específica para aplicativos móveis, permitindo testes automatizados em dispositivos iOS, Android e Windows. Assim como o Selenium, o Appium permite que os desenvolvedores escrevam scripts em Java, C# e outras linguagens para simular interações com aplicativos móveis, como toques na tela e entrada de texto. Ferramenta disponível em **https://appium.io/docs/en/2.0/**

Testes Automatizados (Cucumber, Specflow, Selenium WebDriver, Appium, Java, C#) - continuação	• **Java** e **C#** são duas linguagens de programação populares usadas para escrever scripts de testes automatizados. Ambas as linguagens oferecem uma ampla gama de bibliotecas e frameworks para suportar a automação de testes em diferentes contextos, como testes de unidade, integração e aceitação. Essas linguagens também são amplamente suportadas pelas ferramentas mencionadas, como o **Selenium**, **Appium**, **Cucumber** e **SpecFlow**. Documentação das linguagens disponível em **https://www.java.com/pt-BR/** e **https://learn.microsoft.com/pt-br/dotnet/csharp/**
Controle de Versao (Git, Gitflow)	O controle de versão é uma prática que permite rastrear e gerenciar as mudanças em um código de software ou em qualquer tipo de arquivo digital. Ele facilita a colaboração entre desenvolvedores, a recuperação de versões anteriores, a comparação de alterações e a resolução de conflitos. Existem diferentes sistemas e ferramentas de controle de versão, mas uma das mais populares é o **Git.** **Git** é um sistema de controle de versão distribuído, ou seja, cada desenvolvedor tem uma cópia local do repositório inteiro, com todo o histórico de alterações. Isso torna o **Git** rápido, seguro e flexível. O **Git** também permite criar e gerenciar diferentes ramificações (branches) do código, que são linhas paralelas de desenvolvimento que podem ser mescladas (merged) posteriormente. **Gitflow** é um modelo de ramificação baseado no **Git**, que define um fluxo de trabalho padrão para equipes de desenvolvimento. Ele propõe a existência de cinco tipos de branches: **Master**, **Develop**, **Feature**, **Release** e **Hotfix**. Cada uma dessas branches tem um propósito específico e regras para interagir com as outras. O **Gitflow** ajuda a organizar o código, facilitar as integrações e garantir a qualidade das entregas. A seguir, faremos uma breve descrição da função e cada brach do **Gitflow**: **Master:** A branch principal que contém o código estável e pronto para produção. **Develop:** A branch de desenvolvimento contínua, onde as novas funcionalidades são integradas e testadas. **Feature:** Ramificações criadas a partir de Develop para desenvolver novas funcionalidades. **Release:** Ramificações para preparar uma versão para lançamento, permitindo correções de última hora e testes adicionais. **Hotfix:** Ramificações criadas a partir de Master para corrigir problemas críticos em produção. A documentação do **Git** e um tutorial do **Gitflow** podem ser encontradas respectivamente em **https://git-scm.com/** e **https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow**

Pipeline (CI, CD, Jenkins, GitLab)	Um **pipeline** é um componente de alto nível de integração contínua, entrega e implantação. No contexto de desenvolvimento de software se refere a um conjunto automatizado de etapas que abrangem desde a integração do código até a entrega e implantação em um ambiente de produção. Ele é composto por trabalhos, que definem o que fazer, e estágios, que definem quando executar os trabalhos. Os trabalhos são executados por corredores e vários trabalhos no mesmo estágio são executados em paralelo, se houver corredores concorrentes suficientes. **CI** (Integração Contínua) e **CD** (Entrega Contínua ou Implantação Contínua) são abordagens que visam automatizar e acelerar a entrega de software de alta qualidade. A Integração Contínua envolve a integração frequente do código-fonte em um repositório compartilhado, seguida pela execução automatizada de testes. A Entrega Contínua ou Implantação Contínua vai além, automatizando a implantação do código em ambientes de teste ou produção após passar pelos testes. **Jenkins** é uma ferramenta de automação de código aberto que permite a criação e execução de pipelines de CI/CD. Ele oferece uma interface amigável para configurar etapas de construção, teste e implantação, bem como integração com uma variedade de ferramentas e plugins para personalização. Jenkins é altamente flexível e amplamente usado para criar pipelines personalizados para equipes de desenvolvimento. **GitLab** é uma plataforma completa de DevOps que inclui funcionalidades de controle de versão (Git), gerenciamento de projetos, rastreamento de problemas, CI/CD e muito mais. O **GitLab** CI/CD permite criar pipelines automatizados usando um arquivo de configuração que define as etapas a serem executadas. Isso facilita a criação de pipelines complexos e personalizados diretamente no mesmo ambiente onde o código é mantido. Links relacionados: **https://docs.gitlab.com/ee/ci/pipelines/** **https://docs.gitlab.com/ee/integration/jenkins.html**

Referências

https://www.atlassian.com/br/agile/scrum
https://www.alura.com.br/artigos/metodo-kanban
https://asana.com/pt/resources/burndown-chart
https://www.microsoft.com/pt-br/microsoft-365/business-insights-ideas/resources/how-to-use-burndown-charts-for-better-decision-making
https://www.itdo.com/blog/ejemplos-bdd-behavior-driven-development-con-gherkin/
https://learn.microsoft.com/pt-br/azure/devops/boards/backlogs/manage-bugs?view=azure-devops
https://www.devmedia.com.br/gestao-de-defeitos-ferramentas-open-source-e-melhores-praticas-na-gestao-de-defeitos/8036
https://www.alura.com.br/artigos/git-flow-o-que-e-como-quando-utilizar
https://www.redhat.com/pt-br/topics/devops/what-cicd-pipeline
https://docs.gitlab.com/ee/ci/pipelines/