METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE: UMA REVISÃO BIBLIOGRÁFICA DAS METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE, DEMONSTRANDO SUA APLICABILIDADE E RECOMENDAÇÕES

Lucas de Oliveira Araújo[1]
Simone Sawasaki Tanaka [2]

RESUMO

O artigo responde à pergunta "Qual metodologia escolher no desenvolvimento de um projeto de software?", para isso analisa as metodologias de desenvolvimento utilizadas pelo mercado através do levantamento bibliográfico e sua aplicabilidade no contexto do desenvolvimento de software. O propósito é demonstrar, por meio de critérios, identificar qual caminho é recomendado a ser seguido no processo de escolha de metodologia. Dentro das diferentes abordagens, foi utilizado um estudo de caso de um projeto de software e com base nisso definido o processo de seleção da metodologia. Como resultado foi identificado a metodologia que poderia ser utilizada no projeto.

Palavras-chave: metodologias de desenvolvimento de software; desenvolvimento de aplicações; metodologias ágeis; engenharia de desenvolvimento web; engenharia de software.

ABSTRACT

The article addresses the question "Which methodology to choose in software project development?" by analyzing the development methodologies used in the market through literature review and their applicability in the software development context. The purpose is to demonstrate, using criteria, to identify which path is recommended to be followed in the methodology selection process. Among the different approaches, a case study of a software project was used to define the methodology selection process. As a result, the methodology that could be used in the project was identified.

Keywords: software development methodologies; application development; agile methodologies; web development engineering; software engineering.

1 INTRODUÇÃO

A engenharia de software divide a construção de softwares em etapas. Dentro dessas etapas, encontra-se a gestão de projetos, uma disciplina que tem como

[1] Discente no curso de Computação do Centro Universitário Filadélfia de Londrina - UniFil
[2] Docente - Centro Universitário Filadélfia de Londrina - UniFil

objetivo "Fornecer uma estrutura para gerenciar projetos software intensivo, fornecer orientação prática para planejar, formar a equipe, executar, monitorar projetos e fornecer uma estrutura para gerenciar riscos." (Corporation, 2023). A compreensão desses conceitos implica na forma como o software segue seu desenvolvimento e na maneira como o projeto toma forma.

Com base nessa necessidade, surgiram metodologias que permitem a gestão de projetos de software. Sommerville (2018,p. 32) descreve os modelos como "Esses modelos também são compreendidos como ciclos de vida. "Esses modelos genéricos são descrições mais gerais e abstratas dos processos de software, e podem ser utilizados para explicar as diferentes abordagens ao desenvolvimento de software.". Exemplos de metodologias incluem a em cascata, incremental e ágil. Para cada projeto, recomenda-se a utilização de metodologias diferentes, uma vez que a escolha correta impacta na entrega dos requisitos de software, estrutura e gestão do projeto.

O modelo em cascata segue uma forma de desenvolvimento de software que se baseia em requisitos bem estabelecidos pelos envolvidos. Pfleeger (2004) aponta que o modelo em cascata tem a necessidade de que todos o requisitos sejam levantados e compreendidos em sua completude, só após esse estudo é iniciado as atividades dos desenvolvedores. No entanto essa metodologia pode ser aplicadas a projetos que possuem os requisitos bem definidos, exemplo é a construção de software de uma aeronave onde o a apresentação do pontos de desenvolvimento não possuem tantas mudanças.

Por outro lado as metodologias ágeis apresentam valores que direcionam o foco para além da documentação, buscando entregas rápidas e valor para o cliente. Sutherland (2019, p .20) apresenta na sua obra que importante compreender a necessidade de entregar o valor para as pessoas em vez de processos ou até mesmo dar mais atenção ao funcionamento do produto e reduzir a quantidade de documentos que são constituídos no memento da construção do sistema. Vale ainda dentro desse contexto compreender que os processos são mutáveis e o software acompanha esse ritmo. Essas metodologias enfatizam a colaboração, o feedback contínuo e a adaptação aos requisitos em constante evolução, permitindo uma entrega mais rápida e flexível. Cada metodologia possui sua própria abordagem.

As metodologias tradicionais, como a cascata, foram amplamente utilizadas

no passado, mas frequentemente eram inflexíveis e não atendiam às necessidades de projetos complexos e em constante mudança.

No entanto, não existe uma abordagem única que seja adequada para todos os projetos de desenvolvimento de software. Cada projeto é único e possui requisitos específicos, sendo importante selecionar a metodologia adequada com base nas características e necessidades do projeto. Algumas organizações podem adotar uma abordagem híbrida, combinando elementos de diferentes metodologias para obter os melhores resultados.

A existência de várias metodologias de desenvolvimento mostra a diversidade de formatos que podem ser seguidos na gestão de projetos de software. Na existência de tantas metodologias, o questionário que levou a construção do artigo foi: Qual metodologia devo escolher ?. Desta forma este artigo apresenta metodologias de desenvolvimento de sistemas utilizadas pelo mercado atualmente e entender a sua aplicabilidade. Baseado em caso de uso, o leitor compreende o motivo por trás da existência de tais metodologias de desenvolvimento e como cada uma atende a cada demanda.

2 METODOLOGIA

Na busca de responder a pergunta principal "Qual metodologia devo escolhe?", foi realizado a pequisa de termos relacionados ao assunto, são eles: "Software development methodologies used ", "Most popular software development methodologies", "Software development methodologies", "Software process model", "Metodologias de desenvolvimento de software"e "Software development methodologies".

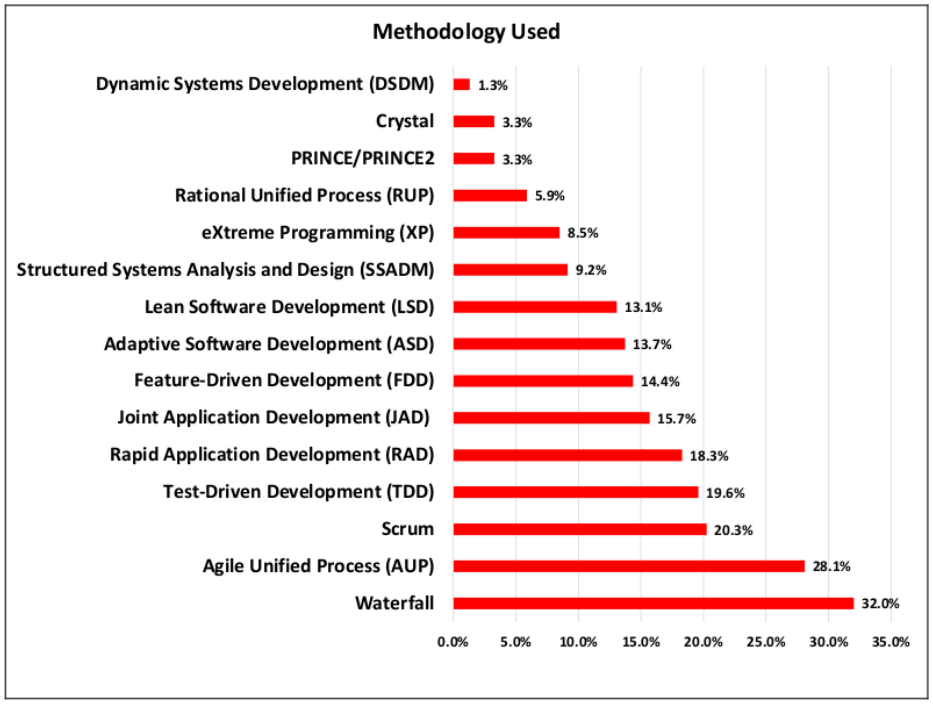
Para realizar o filtro, dentro dos artigos obtidos foi definido critérios que excluem ou incluem como base de pesquisa. Baseado no critérios de exclusão foi selecionado os artigos que não estudam sobre as metodologias de desenvolvimento de software e artigos que não comparam metodologias de desenvolvimento. Ao mesmo modo os critérios de inclusão foram usado na pesquisa metodologias de desenvolvimento de software, que estudam metodologias de desenvolvimento de software e que comparam metodologias de desenvolvimento de software.

3 FUNDAMENTAÇÃO TEÓRICA

A metodologia de desenvolvimento de software desempenha um papel fundamental na gestão de projetos, fornecendo um conjunto de práticas estruturadas e abordagens sistemáticas para orientar e otimizar todo o ciclo de vida do desenvolvimento de software. Pressman (2011) apresenta a ideia de que um processo de software pode ser compreendido como um conjunto organizado de atividades, ações e tarefas que são necessárias para criar software de alta qualidade. Esses processos são estruturados para orientar e facilitar o desenvolvimento de software, proporcionando um caminho definido para a criação de produtos ou sistemas de software. ela fornece uma estrutura organizada, define papéis e responsabilidades, padroniza processos, assegura a qualidade, promove a transparência e auxilia na gestão de riscos. Ao adotar uma metodologia eficaz, as equipes de desenvolvimento de software têm uma base sólida para conduzir projetos de forma eficiente, atingindo resultados de alta qualidade e satisfazendo as necessidades dos clientes. Desta forma foi utilizado como base o artigo "Choice of Software Development Methodologies – Do Project, Team and Organizational Characteristics Matter ?" (Butler, 2015), onde é realizado um estudo verificando qual metodologia de desenvolvimento de software é utilizada pela indústria, considerando o motivo da escolha e o ambiente de desenvolvimento de software. Foi realizado um questionário com 153 respostas.

Dentro das questões empregadas, foi feito um levantamento das metodologias utilizadas, conforme demonstrado na Figura 1.

Figura 1 – Metodologias usadas



Fonte: Butler (2015)

3.1 ENGENHARIA DE SOFTWARE

A engenharia de software é a disciplinas que é responsável por realizar a boa gestão dos processo de desenvolvimento de software. Sommerville(2018, p. 6) define "A engenharia de software é uma disciplina de engenharia que se preocupa com os aspectos da produção de software, desde sua concepção inicial até sua operação e manutenção". Desta forma a engenharia de software apresenta dentro dos seus aspectos a capacidade de mostrar as recomendações para a gestão de um projeto de software, onde ao final deve atender as necessidades do cliente. IEEE (1990) representa "Aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software."

A determinação de padrões no processo de desenvolvimento são importantes para a gestão de software. Com padrões bem definidos destaca a importância dos padrões de processo de software como uma ferramenta essencial para identificar problemas que possam surgir em qualquer processo de desenvolvimento de software. Esses padrões oferecem um método para descrever o processo de forma hierárquica,

começando por um nível de abstração mais elevado. (Pressman, 2011) .

Seguindo essa ideia, o software apresenta um ciclo de vida onde ele passa por fases que vão desde entender o que os interessados desejam até qual algoritmo será utilizado para determinado problema. Sommerville (2018) aponta a padrões dentro das ciclo de vida, essa padrões são definidos por ele como fases, são elas:

1. Especificação: etapa onde os requisitos e restrições devem ser definidos.
2. Desenvolvimento: momento onde é realizado a construção do software.
3. Validação: verificação se o software esta atendendo as necessidades do cliente.
4. Evolução: o software deve acompanhas as necessidades do cliente adaptando-se aos novos requisitos.

Essas atividades permeiam diversos projetos de software, independentemente de tamanho ou complexidade. Cada uma desempenha um papel no ciclo de vida do desenvolvimento de software.

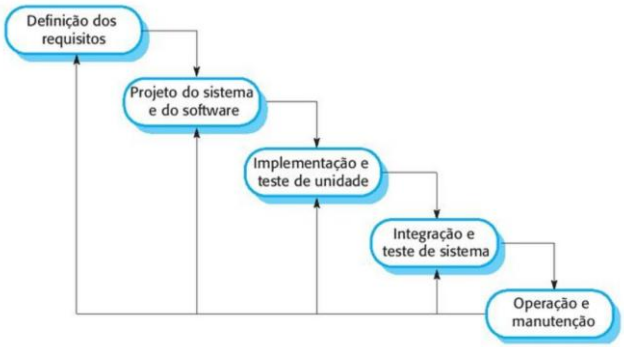
Além disso, a prática da engenharia de software é apresentada como uma atividade de resolução de problemas, guiada por um conjunto de princípios básicos. (Pressman, 2011). Isso implica que não se trata apenas da aplicação de ferramentas e métodos, mas também de uma abordagem sistemática e orientada a princípios na solução de desafios e na criação de soluções eficientes para o desenvolvimento de software. No geral, enfatiza-se a natureza estruturada das atividades envolvidas na engenharia de software, além da abordagem disciplinada e fundamentada em princípios para resolver problemas nesta área.

Neste artigo será abordada as 5 (cinco) metodologias mais utilizadas como demonstrado abaixo, explicando as sua aplicabilidade, vantagens e desvantagens sobre elas levando em consideração um caso de uso.

3.1.1 Waterfall

O modelo em cascata faz parte da categoria prescritivo que funciona de forma linear e sistemática. O corpo de conhecimento foi baseado em Summeville(2018). Conforme demonstrado na Figura 2, as fases do modelo cascata é dividido em cinco etapas.

Figura 2 – Modelo em cascata(Waterfall)



Fonte: Sommerville (2018)

Definição dos requisitos: No inicio é realizado o levantamento ou definição dos requisitos do software, nesta fase é realizada a consulta com o cliente para definir as regras de negócio.

Projeto do sistema e do software: Com os requisitos em mãos o projeto passa de fase e inicia o planejamento do funcionamento das próximas fases.Na fase de planejamento ou projeto do sistema é definido arquiteturas globais, ou seja, a modelagem do software.

Implementação e teste de unidade: Em seguida é iniciado o desenvolvimento do software e testes de unidades. O desenvolvedor realiza a materialização da arquitetura definida na fase anterior, a cada parte desenvolvida é realizada os testes de funcionalidade.

Integração e teste de sistema: Ao término do desenvolvimento, o projeto avança novamente e chaga o momento de implanta-ló. Na implantação é realizado a instalação e teste por completo do sistema a fim de determinar se os requisitos foram atendidos.

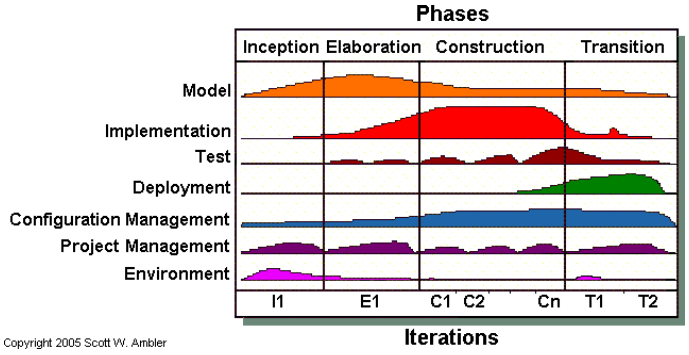
Operação e manutenção: Por fim o processo chega na fase manutenção, todo software possui mudanças e essa fase tem a responsabilidade de adaptar o software de acordo com as necessidades do cliente.

3.1.2 Agile Unified Process (AUP)

O AUP, ou Processo Ágil Unificado, consiste na adaptação do modelo RUP (Rational Unified Process) para metodologias ágeis. O modelo AUP utiliza uma

estrutura similar ao famoso RUP da IBM (Furlan, 2012), ou seja, define fases e disciplinas que percorrem em todo o projeto, mas empregando no desenvolvimento dessas atividades os valores ágeis. O AUP é simples e fácil de entender abordagem para desenvolver uso de software relacionado a negócios técnicas e conceitos ágeis, permanecendo fiel ao RUP (Palaiologou, 2009). Desta forma o AUP é um bom candidato para realizar a transição entre uma metologia não ágil para uma metodologia ágil (Palaiologou, 2009). O ciclo de vida AUP é realizado como demonstrado na Figura 3.

Figura 3 – Ciclo de vida AUP



Fonte: Edeki (2023)

O ciclo de vida do AUP apresenta-se utilizando fases e disciplinas. No eixo horizontal são representadas as disciplinas e no vertical as fases.

As fases são iterativas e incrementais, desta forma o resultado da fase anterior é adicionada ao montante implicando diretamente no resultado da fase posterior. O AUP possui 4 fases, são elas:Inspeção, Elaboração, Construção e Transição.

Inspeção (*Inception*): A fase de inspeção é utilizada para gerir o projeto como um todo, desta forma ela define o escopo do projeto, requisitos, cronograma , riscos, custos e viabilidade do projeto, incluindo dentro dela pontos de checagem utilizados para definir se o curso do projeto esta seguindo o que foi planejado.

Elaboração (*Elaboration*): Ao término da fase de inspeção inicia a elaboração. Nesta fase é realizada diversas rodadas iterativas onde técnicas de prototipação e mock-ups são aplicadas com o objetivo de validar os requisitos levantados na fase anterior. A equipe realizou workshops com os usuários um com cada representante de unidade e um *workshop* global com todas as unidades,

coletando seus comentários, observações e sugestões sobre a interface do usuário, o fluxo de negócios e a funcionalidade do sistema. (Palaiologou, 2009). Por meio de milestones é realizado pontos de checagem que definem se o projeto está atingindo o seu objetivo. Como abordado por Christou (2009) ao realizar diversas rodadas iterativas com usuário é compreendido os riscos e priorizados. Através dessa fase são definidas estruturas importantes como: arquitetura, ferramentas e hardwares, preparando o software para a próxima fase.

Construção (*Construction*): A fase de construção é a que possui maior tempo de execução, após obter e refinar os requisitos é iniciado o processo de desenvolvimento do software. Christou (2009) mostra que nesta fase desenvolve o sistema para que esteja pronto para testes de pré-produção. As fases anteriores identificaram a maioria dos requisitos e criaram uma base para a arquitetura do sistema. Nesta fase é necessário que ao término dela tenha todos os casos de usos 100% finalizados, é importante levar em consideração que o estado que uma caso de uso precisa atingir é de "estabilidade", ou seja, é normal que ao decorrer do projeto que o caso de uso sofra pequenas alterações, mas não altere a sua base. Tais mudanças devem ter sido mapeadas nas fases anteriores. Ao finalizar cada caso de uso é realizado rodadas de teste de usabilidade determinando se o software atingiu as necessidades.

Transição (*Transition*): Esta fase foca na entrega do software e testes de aceitação. Através da integração gradual o sistema é implantando, é importante que os usuários que inciam essa fase são de menor escala permitindo analisar e realizar otimizações finas dentro da usabilidade e performance do sistema. Dentro da fase também é o momento onde os treinamentos para uso são realizados, bem como a documentação técnica com os detalhes sobre a arquitetura física e manuais de utilização. Para determinar a conclusão dessa etapa é necessário passar nos pontos de checagem(milestones). Esse pontos são concentrados em medir o nível de aceitação dos interessados ao produto entregue colando a frente os seguintes testes:

- Aceitação do time de negócio;
- Aceitação do time operacional;
- Aceitação do suporte;
- Estimativa vs Custo final.

Cada fase possuem dentro delas a atividades ou disciplinas empregadas no projeto de software. As disciplinas do AUP são executadas de forma iterativa, definindo as atividades que os membros da equipe de desenvolvimento devem realizar para construir, validar e entregar o software de forma que atenda as necessidades do negócio (Amber, 2023). As fases do AUP possui 7 disciplinas: Modelagem, Implementação, Teste, Implantação, Gerenciamento das configurações e Ambiente.

Modelagem(*Model*): A disciplinas de modelagem tem como objetivo entender o comportamento da lógica do negócio, identificar a origem do(s) problema(s) que inicia o projeto e identificar a solução possível e viável para o(s) problema(s).

Implementação(*Implementation*): O objetivo da disciplina de implementação é transformar os modelos levantados em software. A fase de implementação é importante, pois é na prática que os casos de uso saem do mundo abstrato e ganham uma cara que o cliente consegue interagir.

Teste(*Test*): Esta disciplina tem como objetivo garantir a qualidade necessária para que o software seja entregue. Através dos testes é realizado as validações que contribuem na para uma entrega consistente sem erros.

Implantação(*Deployment*): A implantação é a disciplina responsável por realizar a entrega do sistema e validando se o sistema aderiu a infraestrutura planejada.

Gerenciamento das configurações(*Configuration Management*): O objetivo da disciplina é realizar a gestão de acesso ao projeto e produto de trabalho. Incluindo são somente uma forma de rastrear as versões do produto, mas também controlar e gerenciar as mudanças entre elas.

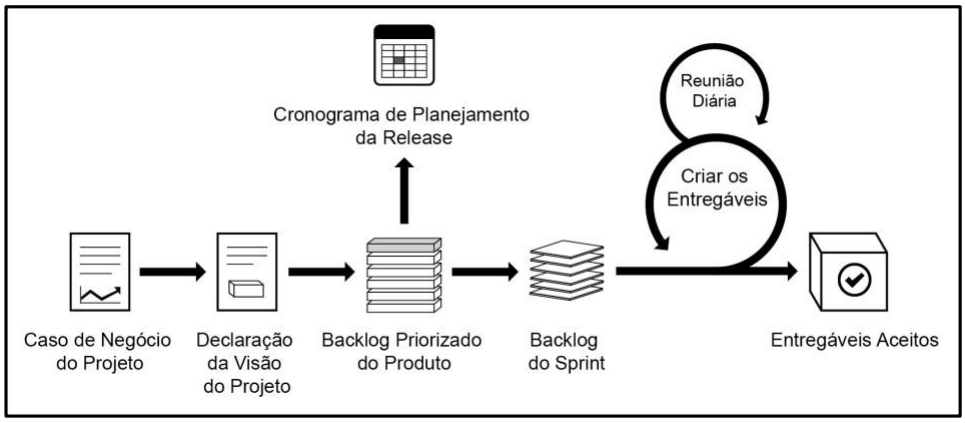
Gerenciamento de projeto(*Project Management*): A disciplina tem como ponto central é controlar as atividades do projeto. Nesta etapa ocorre a gestão de riscos, definir e gerenciar as tarefas para com as pessoas e garantir que o sistema vai seguir o cronograma trassado.

Ambiente(*Environment*): Verifica como as ferramentas inter sistema irão funcionar, e caso não são aplicadas correções.

3.1.3 Scrum

O Scrum é uma metodologia ágil criada por Mike Beedle, Ken Schwaber e Jeff Sutherland com o objetivo de aumentar a produtividade da equipe, realizando pequenas entregas que passam por fases de inspeção e revisão. Dessa forma, as necessidades apontadas pelos clientes são entregues de maneira transparente. Seu ciclo de vida segue em forma de Sprints, onde ocorre o monitoramento das entregas,como demonstrado na Figura 4.

Figura 4 – Ciclo de vida de uma Sprint - Scrum



Fonte: (Satpathy, 2023)

Como corpo de conhecimento do Scrum é utilizado o guia desenvolvido por Ken Schwaber e Jeff Sutherland (2020) onde é mantido e atualizado. O Scrum participa do rol de metodologias ágeis, desta forma como mostra (Sommerville, 2018) as metodologia ágeis apresenta sua abordagem iterativa e incremental, ou seja, ao término de cada Sprint é realizado o incremento das atividades desenvolvidas. O Scrum possui bases com pilares empíricos, dentre eles: transparência, inspeção e adaptação.

Transparência: Ter transparência para um projeto significa que tanto quem irá produzir o software quanto quem requerer precisão apresentar o trabalho realizado. A transparência é um fator importante para tomadas de decisão mais precisas e seguras conforme Schwaber(2020) "Artefatos com baixa transparência podem levar a decisões que diminuem o valor e aumentam o risco.".

Inspeção: A inspeção é responsável por averiguar se os artefatos estão de

acordo com as metas, é realizada com frequência e diligência com o objetivo de detectar problemas. O Scrum conta com um ritmo esperado dentro dos seus eventos justamente para ajudar na inspeção. A inspeção possui a necessidade de estar junta ao pilar de adaptação, sendo considerada ineficaz se não estiverem juntas. (Sutherland, 2023). Os eventos do Scrum são concebidos para instigar e promover mudanças.

Adaptação: A adaptação é a capacidade de ajustar-se as mudanças, dentro do Scrum é possível que o projeto tenha a capacidade de adaptar-se as mudanças ao longo do caminho, mas essa mudança devem está dentro dos limites aceitáveis.

Segundo Schwaber(2020) os ajuste devem ocorrer o mais rápido possível, tendo em vista minimizar que estes desvios não levem a novos. Para que a adaptação ocorra de maneira mais fácil é necessário que os envolvidos sejam auto-gerenciadas e empoderadas. Schwaber(2020) uma característica esperada de um *Scrum Team*, é ter a capacidade de adaptação as mudanças e aprendendo algo novo por meio da inspeção.

3.1.3.1 Funcionamento

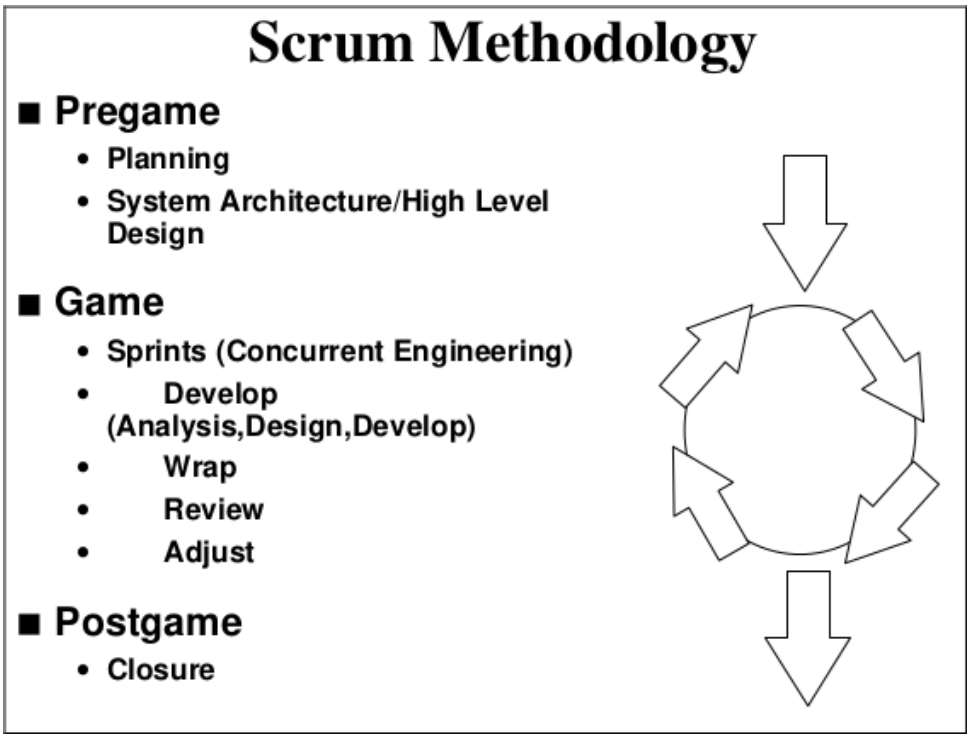
O Scrum possui no seu workflow a Sprint, onde é o evento inicial com duração de no máximo 30 dias. Cada Sprint inicia com o *Sprint Planning*, onde o time define os objetivos que devem ser entregues naquela Sprint e preencher o primeiro artefato chamado *Product backlog*. Dentro do *Product backlog* é listado todas as funcionalidades levantadas na Sprint planning em seguida é selecionado quais funcionalidades devem ser desenvolvidas, também conhecida como histórias de usuário. A quebra das histórias são desenvolvidas dando origem ao evento *Sprint backlog*. Com as funcionalidades levantadas é iniciado o desenvolvimento, onde diariamente é realizada momentos para descobrimento das atividades, onde os membros coordenam e compreendem o as atividades desenvolvidas para atender os objetivos da Sprint. (Schwaber, 1996). Esses incrementos devem ser desenvolvidos, significa que eles estão conforme a qualidade definida pelo a equipe levantados na definição de feito.

Antes do término de cada sprint, o time compartilha os incrementos com os

envolvidos obtendo o *feedback*, esse evento é conhecido por *Sprint Review*. O *Product Backlog* é atualizado de acordo com a fase de descoberta. Finalmente, uma Sprint Retrospective é mantido no time para aprender e adicionar ao processo. Nesse contexto, a figura do *Scrum Master* é fundamental para o desenvolvimento dessa metodologia (Schwaber, 1996), da mesma forma o *Product Owner* é importante para mostrar que o trabalho do time é valioso. Todos os outros membros do time são considerados desenvolvedores do produto e incluindo todas as habilidades e especializações para realizar as entregas dos incrementos.

O *Scrum Master* concentra-se em seis atividades centrais: facilidade, orientação, negociação, adaptação de processos, coordenação e proteção. Entre as outras atividades exercidas pelo *Scrum Master* estão o gerenciamento do projeto, compreendendo os pontos negativos que impactam a performance. O *Product Owner* é relacionado com as atividades desenvolvidas pelo time, desta forma o ele está centralizado para analisar os requisitos, priorizando a gestão das das entregas e dos envolvidos. Isso se trata da fase de descobrimento e comunicação das necessidades do negócio. Os *Product Owners* possuem mais sucesso quando envolvem o time na estratégia de formulação, refinamento e a gestão dos envolvidos. Para o sucesso do time Scrum é necessário que sejam auto gerenciáveis, é importante ressaltar que as capacidades técnicas e interpessoal são necessárias para o desenvolvimento das atividades. (Schwaber, 1996). De maneira geral é possível acompanhar as seguintes fases Figura 5.

Figura 5 – Visão geral das fases do Scrum



Fonte:Schwaber (1996)

Pregame - Definição de novas entregas baseadas no que foi levantado no *Backlog*, ao longo de uma estimativa os cronogramas e custos são definidos. Se um novo sistema está sendo desenvolvido, esta fase consiste nas parte conceito e análise. Se for em um sistema existente, esta fase consiste na analise (Schwaber, 1996).

Arquitetura: Define como os itens *Backlog* devem ser implementados. Nesta fase incluem a arquitetura do sistema, mudanças e o design de alto nível.

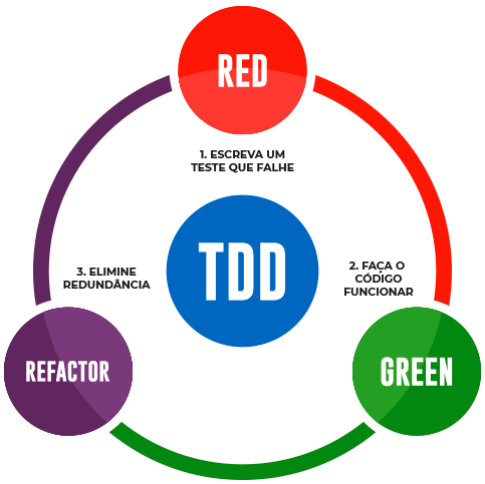
Game - Desenvolvimentos das Sprints: O desenvolvimento de novas entregas de funcionalidades, com constância respeitando as variáveis de tempo, requisitos, qualidade, custo e concorrência. Interação com as variáveis definem o fim dessa fase (Schwaber, 1996). As múltiplas iterações de desenvolvimento, ou ciclos, e a usabilidade que envolve o sistema.

Postgame - Closure: Preparação para entrega, incluindo a documentação final, pre-entrega, estágio de teste e entrega das funcionalidade.

3.1.4 Test-Driven Development (TDD)

Originada da metodologia ágil XP (*Extreme Programming*), o TDD ou Desenvolvimento Orientado a Testes determina que para cada especificação é necessário realizar testes unitários antes da implementação da aplicação. O ciclo a ser seguido é o *Red*, *Green* e *Refactor*, como apontado na Figura. 6.

Figura 6 – Ciclo de vida TDD



Fonte: Guedes (2023)

Apesar de ser citada em (Butler, 2015), o TDD não é considerada uma metodologia de de desenvolvimento, mas uma estratégia de para a construção do sistema. Num caminho oposto às metodologias tradicionais de desenvolvimento de software, esta abordagem começa inicialmente criando o teste de funcionalidade (Hammond, 2012), visando atender ao resultado final. Dessa forma, o código se ajusta às necessidades. Beck (2010) destaca que os desenvolvedores devem primeiro produzir os resultados esperados para, em seguida, construir ao redor, adaptando-os às necessidades.

Vamos considerar a criação de uma funcionalidade de login para um aplicativo:

1. **Definição do Teste de Funcionalidade:** Antes de iniciar a codificação, é definido o teste de funcionalidade para o login. Este teste deve cobrir diferentes aspectos, como a verificação de credenciais corretas e incorretas, a navegação para a tela de login, a validação dos campos de entrada, entre outros.

2. **Produção do Resultado Esperado:** Com base no teste de funcionalidade definido, o desenvolvedor cria o código de login que atenda aos critérios estabelecidos no teste. Isso pode incluir a criação da interface de usuário para inserir credenciais, a lógica para autenticar os usuários e o redirecionamento para a página inicial após o login bem-sucedido.

3. **Adaptação e Refinamento:** Durante a implementação, o código pode ser ajustado e refinado à medida que novos requisitos ou insights surgem. Isso pode envolver aprimorar a segurança, otimizar a interface do usuário ou incluir validações adicionais com base no feedback do teste ou das necessidades do usuário.

4. **Construção ao Redor da Funcionalidade:** Uma vez que a funcionalidade básica de login esteja operacional e atenda aos requisitos principais, outras características podem ser construídas ao redor dela. Por exemplo, a recuperação de senha, a autenticação de dois fatores ou a integração com serviços de terceiros podem ser implementadas, expandindo assim a funcionalidade inicial do login.

Portanto, a abordagem se concentra em definir os testes de funcionalidade desejados, criar o código para atender a esses testes e, em seguida, adaptar e expandir conforme necessário, visando sempre ao resultado esperado e às necessidades do usuário.

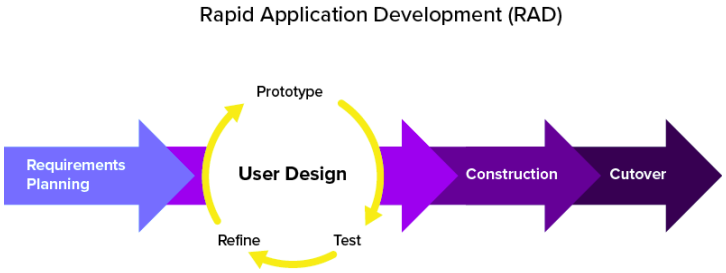
Para melhor compreensão do TDD, o sinal vermelho representa o código de teste que falha, ou possivelmente que não compila. O sinal verde é o resultado final de escrever a quantidade mínima de código necessária para fazer com que o código de teste passe. A refatoração é usada para eliminar qualquer duplicação de código ou técnicas de programação ruins que foram introduzidas ao chegar ao estado verde o mais rapidamente possível (Hammond Susan; Umphress, 2012).

3.1.5 Rapid Application Development (RAD)

Esta metodologia parte da ideia de realizar incrementos em cada fase de desenvolvimento, ou seja, cada fase aproveita o resultado da fase anterior adicionado sempre dentro do produto de software às necessidades empregadas pela fase atual. Criado por James Martin, este modelo apresenta sua utilização para projetos

menores, entregando de forma rápida as necessidades dos clientes. Seu ciclo de vida apresenta fases como planejamento de requisitos, design de usuário, prototipação, teste, refinamento, desenvolvimento e implementação como demonstrado na Figura 7.

Figura 7 – Ciclo de vida RAD



Fonte: Guedes (2023)

O RAD apresenta componentes importantes para compreender as o seu funcionamento: Desenvolvimento de aplicação conjunta, Desenvolvimento rápido, Salas limpas, Time boxing, Prototipação incremental, Ferramentas de desenvolvimento rápido, Projetos altamente interativos e de baixa complexidade, Ferramentas de desenvolvimento rápido e Projetos altamente interativos e de baixa complexidade Davies *et al*. (1999).

Desenvolvimento de aplicação conjunta (JAD): O RAD é caraterizado por ter o time pequeno de desenvolvimento que possui tipicamente entre 4 à 8 pessoa. No desenvolvimento do projeto de software é importante ressaltar que tanto os usuários quanto o time de desenvolvimento podem tomar decisões dentro do projeto Davies *et al.* (1999). Desta forma é necessário que as duas mãos de tomada de decisões portanto a comunicação e feedback contínuo são realizados constantemente entre o usuário e o time de desenvolvimento. Os Usuários possuem o conhecimento detalhado da área em que o software será aplicado. Com a mesma importância o time de desenvolvimento deve ter conhecimento avançado nas tecnologias e ferramentas empregas no processo de desenvolvimento. A maioria das abordagens para o RAD utilizam de Join Application Development (JAD) ou Design de aplicação conjunta em vários pontos do projeto, destacando o processo de levantamento de requisitos. Dentro de eventos como workshops, os usuário chave, clientes, alguns desenvolvedores produzem o escopo e os requisitos do negócio, sobe eles um

facilitador que torna-se a interface entre todos os envolvidos e o time de de desenvolvimento. É esperado que o time de desenvolvimento é realize o levantamento dos requisitos e documentações em um timeboxing de três a cinco dias. Com as funcionalidade em mãos, é possível definir as fases que determinam as entregas, do mesmo jeito que foi realizado na fase de levantamento de requisitos são entregues as funcionalidades em workshops Davies *et al*. (1999).

Desenvolvimento rápido: Projetos que utilizam o RAD são relativamente de pequena escala e de curta duração. Dentro da metodologia é discutido que um projeto de dois a seis meses de tamanho normal de um projeto. A principal justificativa não utilizar o RAD para projetos que ultrapassem mais de seis meses é devido ao fato que esse projeto transborde os objetivos do negócio.

Salas limpas: A utilização de "salas limpas"não estão relacionadas a limpeza do ambiente, mas empenhar o foco do time no desenvolvimento do projeto.

Time boxing: O gerenciamento de um projeto com a metodologia RAD prioriza o desenvolvimento e a entrega final ao enfocar o escopo do projeto, fazendo uso de pontos de entrega ou timeboxings. Se houver dificuldades no progresso do projeto, a ênfase recai na necessidade de ajustar o cronograma das entregas, em vez de ampliar o prazo Davies *et al.* (1999).

Prototipação incremental: RAD é frequentemente aponta em termos prototipação incremental e fases de entrega. A prototipação apresenta aos envolvidos uma forma concreta do produto em desenvolvimento. Os desenvolvedores e os usuários então discutem o protótipo, concordando com aprimoramentos e alterações. Esse ciclo de inspeção-discussão-alteração geralmente é repetido pelo menos três vezes em projetos RAD, até que o usuário esteja satisfeito com o sistema. Dentro da metodologia RAD a prototipação segue o seguinte fluxo: coleta de requisitos; design de aplicativo; construção de aplicativo; teste; entrega Davies *et al*. (1999).

Ferramentas de desenvolvimento rápido: Não é surpreendente encontrar que as abordagens modernas para o Desenvolvimento Rápido de Aplicações (RAD) exigem um bom suporte de ferramentas para mudanças de desenvolvimento rápidas. Isso normalmente significa uma combinação de linguagens de quarta geração (4GLs), construtores de interface gráfica do usuário (GUI), sistemas de gerenciamento de banco de dados (DBMS) e ferramentas de engenharia de software auxiliada por

computador (CASE). Usando essas ferramentas, algumas alterações nos protótipos podem ser feitas no local durante reuniões entre o usuário e o desenvolvedor Davies *et al.* (1999).

Projetos altamente interativos e de baixa complexidade: A maioria dos projetos RAD parece ser realizada em aplicativos altamente interativos, que têm um grupo de usuários claramente definido e não são computacionalmente complexos. Por exemplo, a empresa financeira do Reino Unido, Norwich Union (Anônimo, 1996c), desenvolveu um sistema de negociação eletrônica originalmente para o setor de seguros automotivos da empresa usando uma abordagem RAD interna Davies *et al.* (1999). Aparentemente, a equipe de desenvolvimento levou apenas três meses para converter este sistema para o setor de seguros residenciais.

Tipos de projeto RAD: Geralmente, existem dois tipos de projeto RAD: o intensivo e o projeto RAD faseado. No tipo altamente intensivo de projeto, uma equipe de desenvolvedores e usuários fica isolada em uma sala limpa por algumas semanas e espera-se que produza um entregável funcional ao final desse período. Um projeto faseado é aquele que se estende por vários meses. Tais projetos normalmente são iniciados por um *workshop* de Requisitos Conjuntos (JAD) ou Planejamento de Requisitos Conjuntos (JRP) Davies *et al.* (1999).

As fases subsequentes do projeto são normalmente organizadas em termos da entrega e demonstração de três protótipos incrementais. O objetivo é refinar continuamente o protótipo até que ele se torne algo entregável no final do período determinado (*timebox*). A tendência é descartar a aplicabilidade do RAD para projetos de grande escala, especialmente na construção de sistemas de informação distribuídos em larga escala, como bancos de dados corporativos. A evidência sugere que tal infraestrutura é melhor estabelecida antes de empreender projetos RAD. Essa infraestrutura pode então servir como alimentadora para sistemas desenvolvidos usando o RAD. Isso talvez não seja surpreendente quando se considera que a maioria das ferramentas RAD trabalha de alguma forma com um banco de dados. Portanto, o banco de dados precisa ser criado antes do início do desenvolvimento do aplicativo.

4 DESENVOLVIMENTO

Para aplicar as metodologias estudas, foi definido um estudo de caso de um projeto de software para ser utilizado como base. Este projeto é a ferramenta de gerenciamento de concursos, utilizado para a contratação de colaboradores onde auxiliano processo de aplicação das provas que envolvem concursos. Este projeto abrangea contratação, definição de função e pagamento dos colaboradores.

Os critério utilizados para avaliar qual metodologia utilizar foram:

1. Requisitos do Projeto: Verificar se a metodologia é adequada para lidar com os requisitos específicos do projeto. Isso inclui o tamanho e complexidade do projeto, os objetivos de negócio envolvidos, as restrições de tempo e recursos, entre outros fatores relevantes.
2. Adaptabilidade: Avaliar a capacidade da metodologia de se adaptar a mudançase incertezas ao longo do projeto. Uma metodologia flexível e iterativa pode ser mais eficaz em lidar com requisitos em evolução e respostas rápidas a mudanças.
3. Comunicação e Colaboração: Verificar se a metodologia promove uma comunicação eficiente e uma colaboração eficaz entre os membros da equipe de desenvolvimento, envolvidos e usuários finais. A interação e o *feedback* contínuossão aspectos importantes para o sucesso do projeto.
4. Controle de Qualidade: Avaliar se a metodologia possui mecanismos adequados para garantir a qualidade do software desenvolvido. Isso pode incluir práticasde teste, revisões de código, automação de processos, entre outros aspectos relacionados à qualidade do produto final.
5. Produtividade: Analisar o potencial da metodologia para aumentar a produtividade da equipe de desenvolvimento. Isso envolve considerar a eficiência das práticas e ferramentas utilizadas, bem como a capacidade de promover a reutilização de código e a automação de tarefas repetitivas.
6. Experiência da Equipe: Levar em consideração a experiência e as habilidades da equipe de desenvolvimento em relação à metodologia em questão. Uma equipe mais familiarizada e experiente com uma determinada metodologia provavelmente terá um

desempenho mais eficaz.

7. Suporte e Recursos: Verificar se a metodologia possui suporte adequado, documentação, recursos e ferramentas disponíveis para auxiliar a equipe de desenvolvimento durante o processo.

8. Tamanho do projeto: Avalia qual o tamanho do projeto, metodologias utilizam como base o tamanho do projeto.

Ao considerar esses critérios, é possível realizar uma avaliação mais abrangente e objetiva da eficácia de uma metodologia de desenvolvimento de software para um determinado tipo de projeto. Pela as características do projeto ele é considerado de baixa complexidade e com pequena duração. O time de desenvolvimento apresenta histórico de projetos utilizando prototipação incremental de forma contínua. A cada funcionalidade desenvolvida é comunicado aos envolvidos, ou seja, apresenta caraterísticas de *feedback* para avaliar a evolução do projeto.

Foi definido a partir dos critérios uma pontuação a fim de avaliar para cada metodologia Tabela 1.

Tabela 1 – Tabela com pontuação dos critérios

Critério	Pontuação
Requisitos do Projeto	1
Adaptabilidade	1
Comunicação e Colaboração	1
Controle de Qualidade	1
Produtividade	1
Experiência da Equipe	1
Suporte e Recursos	1
Tamanho do projeto	1

Fonte: Autor(2023)

Foi realizado levantamento e a soma das pontuações obtidas ao analisar as métricas, equipe e projeto Tabela 2.

Tabela 2 – Tabela com a pontuação entre cada metodologia

Metodologia	Pontuação	Critérios
Cascata	0	Não atendeu nenhum dos critérios.
AUP	3	1,3 e 4
Scrum	4	1,2,3 e 4
TDD	0	Não atendeu nenhum dos critérios.
RUP	8	Atígiu todos os critérios.

Fonte: Autor (2023)

Como apontado pelos os pesquisadores Butler (2015), Sommerville (2018) e (Thorbergsson, 2006) não existe metodologia que atinja todos os aspectos do projeto, mas aquela que se adapta melhor para cada processo de desenvolvimento.

5 CONCLUSÃO

Com base no estudo de caso analisado e nas reflexões apresentadas, torna-se evidente que a abordagem de desenvolvimento RAD (*Rapid Application Development*) se destaca como a metodologia mais adequada para o projeto em questão. Não apenas não há soluções milagrosas à vista no momento, a própria natureza do software torna improvável que existam (Brooks, 1987). No entanto, ao considerar os critérios estabelecidos no estudo, a RAD demonstrou ser a escolha mais apropriada, devido à sua ênfase na prototipação, no desenvolvimento conjunto e na comunicação contínua para obtenção de *feedback*. Além disso, a natureza de baixa complexidade e o tamanho reduzido do projeto, com um prazo de desenvolvimento de apenas 6 meses, tornam a RAD ainda mais compatível com as necessidades da empresa. Nesse contexto, a seleção da metodologia adequada é um passo essencial para o sucesso do projeto, garantindo uma abordagem eficaz e eficiente para atender às demandas específicas da organização e do software a ser desenvolvido. Portanto, a escolha da RAD representa um passo importante em direção à realização bem-sucedida deste projeto, embasada em sólidos fundamentos e considerações práticas.

REFERÊNCIAS

AMBER, S. w. **The Agile Unified Process.** 2023. Disponível em: http://www.ambysoft.com/downloads/agileUP.zip. Acesso em: 18 set. 2023.

BROOKS, F. P. No silver bullet essence and accidents of software engineering**. Computer**, v. 20, n. 4, p. 10–19, 1987.

BUTLER, L. R. V. C. W. Choice of software development methodologies – do project, team and organizational characteristics matter? **IEEE Software**, v. 33, n. 5, p. 86–94, 2015.

CORPORATION, R. software. **RUP**. 2023. Disponível em: http://walderson. com/ibm/rup7/largeprojects. Acesso em: 18 set. 2023.

EDEKI charles. Agile unified process. **International Journal Of Computer Science AndMobile Applications - IJCSMA,** v. 1, n. 3, p. 13–17, 2023.

FURLAN, R. C. **Agile unified process e a aderência ao modelo de processo de software mps.br nível g.** 2012. Especialização (Monografia em Engenharia de Software)- Universidade Tecnológica Federal do Paraná,Medicaneira, 2012.

GUEDES marylene. **Afinal, o que é tdd?** 2023. Disponível em: https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd. Acesso em: 18 set. 2023.

HAMMOND SUSAN;UMPHRESS, D. Test driven development: The state of the practice. *In*: **Proceedings of the 50th Annual Southeast Regional Conference**. New York, NY, USA: Association for Computing Machinery, 2012. (ACM-SE '12), p. 158–163. Disponível em: https://doi.org/10.1145/2184512.2184550. Acesso em: 18 set. 2023.

PALAIOLOGOU ioannis Christou; stavros Ponis; eleni. **Using the agile unified processin banking. IEEE Software**, v. 27, n. 3, p. 72–79, 2009.

PRESSMAN roger s. **Engenharia de software**: uma abordagem profissional. 7. ed. Porta alegre: Mcgraw Hill Brasil, 2011.

SATPATHY tridibesh. **Um guia para o conhecimento em scrum (guia sbok™).** 2023. Disponível em: https://portaldaobmep.impa.br/uploads/material_teorico/ d9gycv8klfcwc.pdf. Acesso em: 18 set. 2023.

SCHWABER ken. **Scrum development process**. [*S.l.*]: Scrum Org, 1996.

SOMMERVILLE ian. **Engenharia de software**: teoria e prática. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

SUTHERLAND ken S. **O Guia Definitivo para o Scrum: As Regras do Jogo.** 2023.

Disponível em: https://scrumguides.org/docs/scrumguide/v2020/ 2020-Scrum-Guide-PortugueseBR-3.0.pdf. Acesso em: 18 set. 2023.

THORBERGSSON, O. B. . D. D. . H. Comparison of software development life cycles: a multiproject experiment. **IEEE Software**, v. 153, n. 3, p. 87–101, 2006.