
**BRENO HENRIQUE TISO TANA**

# DESENVOLVIMENTO E MANUTENÇÃO DE SOFTWARES UTILIZANDO METODOLOGIA ÁGIL

**LAVRAS – MG**

**2021**

**BRENO HENRIQUE TISO TANA**

# DESENVOLVIMENTO E MANUTENÇÃO DE SOFTWARES UTILIZANDO METODOLOGIA ÁGIL

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Raphael Winckler de Bettio

Orientador

**LAVRAS – MG**

**2021**

**Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos**

**da Biblioteca Universitária da UFLA**

Tana, Breno Henrique Tiso

Desenvolvimento e manutenção de softwares utilizando metodologia ágil / . $1^a$ ed. rev., atual. e ampl. – Lavras : UFLA, 2021.

40 p. : il.

Relatório de Estágio (graduação)–Universidade Federal de Lavras, 2021.

Orientador: Prof. Dr. Raphael Winckler de Bettio.

Bibliografia.

1. TCC. 2. Monografia. 3. Dissertação. 4. Tese. 5. Trabalho Científico – Normas. I. Universidade Federal de Lavras. II. Título.

CDD-808.066

**BRENO HENRIQUE TISO TANA**

# DESENVOLVIMENTO E MANUTENÇÃO DE SOFTWARES UTILIZANDO METODOLOGIA ÁGIL

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 13 de Maio de 2021.

| | |
|---|---|
| Prof. Dr. Neumar Costa Malheiros | UFLA |
| Mr. Eduardo Vilela Pierangeli | dti digital |

Prof. Dr. Raphael Winckler de Bettio
Orientador

**LAVRAS – MG**
**2021**

*Dedico aos meus pais e irmãs, que sempre me apoiaram e incentivaram.*

## AGRADECIMENTOS

Agradeço primeiramente à minha família, que sempre foi meu porto seguro, me impulsionando e amparando. Agradeço a todas as amizades proporcionadas pela UFLA e também aos amigos de longa data, em especial aos "barbas". Todos os momentos vividos juntos foram excepcionais e serão guardados com muito carinho. Agradeço aos membros e ex-membros da Emakers Júnior, por todas as oportunidades e aprendizados proporcionados e compartilhados. Agradeço ao orientador e colega de projeto Fábio, que sempre me apoiou e confiou no meu trabalho. Agradeço à todos os professores do DCC, em especial aos professores Paulo Afonso e Raphael Bettio, pelos ensinamentos e apoio nos projetos desenvolvidos. Para todas as pessoas que de alguma forma contribuíram com minha formação, obrigado!

*Aqueles que não conseguem lembrar o passado estão condenados a repeti-lo.*

*(George Santayana)*

# RESUMO

A cultura ágil se popularizou no ramo de desenvolvimento de softwares, com a gestão de projetos utilizando ciclos iterativos e incrementais, provendo flexibilidade e adaptabilidade às equipes, objetivando a entrega de produtos que atendam as reais necessidades e anseios dos clientes. Tal cultura vem se difundindo em outras áreas do mercado e ganhando cada vez mais visibilidade e credibilidade. Nos dias de hoje, com a alta competitividade e a necessidade de soluções cada vez mais robustas, o mercado enxerga a necessidade de uma mudança de paradigma. Dessa forma, o trabalho busca mostrar como a adoção da cultura ágil em um empresa de desenvolvimento de softwares promove uma maior aproximação com o cliente e entregas de qualidade, fidelizando o cliente e possibilitando um maior engajamento das partes.

**Palavras-chave:** Metodologia ágil. Sistema Web. Serviço Web. dotNET.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

O presente trabalho tem como objetivo apresentar os processos e atividades executadas pelo autor durante o período de estágio na DTI Sitemas Ltda (dti digital). Tal ciclo se mostrou de suma importância, onde os conhecimentos adquiridos durante o curso puderam ser aplicados e desenvolvidos, somados ao desenvolvimento pessoal e profissional possibilitado pelas oportunidades oferecidas pela dti digital.

A dti digital é uma empresa de Belo Horizonte - MG, fundada em 2009 e com filial em Lavras - MG. Seu número de colaboradores atualmente supera os 800, com diversos clientes de diferentes áreas de atuação. Sua organização interna é baseada na horizontalidade, com equipes interdisciplinares divididas em *squads* que juntas formam tribos, que por sua vez, juntas formam as alianças. Geralmente as tribos, e em alguns casos as alianças como um todo, atendem o mesmo cliente, de forma a facilitar a troca de conhecimentos e colaboração entre os times.

Essa estrutura organizacional possibilita que as tribos tenham autonomia de gestão, sendo elas responsáveis pelos seus gastos, contratações etc. Com isso se faz necessária a divisão de responsabilidades entre diversas pessoas e grupos, onde uma grande confiança é depositada nos funcionários, que por outro lado tem seu crescimento, profissional e pessoal, alavancado constantemente.

A cultura ágil é o principal pilar da dti, tanto para sua gestão, quanto para a execução e oferta de projetos para seus clientes. O objetivo dos *squads* é promover a transformação digital para os clientes, construindo juntos produtos satisfatórios e que gerem valor, não se limitando a entrega das demandas solicitadas.

Os projetos e atividades desempenhadas durante o estágio foram, principalmente, relacionadas a sustentação e melhoria de sistemas existentes de uma instituição financeira (IF). As tecnologias utilizadas foram .NET Framework, .NET Core e Spring Boot, para a construção/melhoria de sistemas web, serviços web e rotinas. Além disso, o uso de bancos de dados relacionais se fez presente em todos os projetos, sendo necessário desenvolver *scripts* e procedimentos para a manipulação dos dados em diferentes Sistemas Gerenciadores de Bancos de Dados, como Microsoft SQL Server, Sybase ASE e Oracle Database.

Dentre as tarefas executadas estão a análise de demandas, riscos e esforço, construção de roteiros de testes e testes unitários, desenvolvimento de softwares, validações, comunicação com clientes e ritos contínuos de revisão e melhoria dos processos, utilizando o *Scrum* e outras ferramentas.

O restante deste relatório está organizado da seguinte forma: No Capítulo 2, serão apresentados os conceitos necessários para o entendimento das atividades. No Capítulo 3, serão abordadas as atividades, processos e métodos executados durante o estágio. Por fim, no Capítulo 4, serão descritas as considerações finais e as contribuições proporcionadas pelo estágio.

# 2 REFERENCIAL TEÓRICO

Nesse capítulo, serão apresentadas as principais tecnologias, ferramentas e os conceitos relacionados e utilizados durante o período de estágio.

## 2.1 Metologia ágil

Em meados dos anos 90 surgiu um movimento em alternativa aos pesados métodos que predominavam na época, também chamados de métodos tradicionais, de gerenciamento de desenvolvimento *software*. Esse movimento alternativo é conhecido como ágil, o qual foi escolhido para representar essa nova abordagem, mais dinâmica e equivalente as necessidades de projetos complexos modernos (SABBAGH, 2014).

Detre os métodos tradicionais, o mais conhecido e utilizado era o cascata, onde o desenvolvimento de um produto se dava de forma sequencial, com uma etapa se iniciando após o resultado e término da anterior (SEMEDO, 2012).

A maioria das metodologias ágeis não possuem nada de novo. O que as diferencia das metodologias tradicionais são o enfoque e os valores. As pessoas e as interações se sobressaem a processos ou algoritmos, com a preocupação de se gastar menos tempo com documentação e mais com a implementação e entrega de resultados. Uma característica importante das metodologias ágeis se da pelo fato de serem adaptativas ao invés de preditivas, se adaptando aos novos fatores, ao invés de analisarem previamente tudo o que pode acontecer no decorrer do processo (SOARES, 2004).

A popularidade das metodologias ágeis cresceu após um encontro, realizado em 2001, de dezessete especialistas em processos de desenvolvimento de *software*, com o objetivo de discutir e estabelecer valores das metodologias ágeis (SOARES, 2004). O encontro resultou na criação da Aliança Ágil e o estabelecimento do "Manifesto Ágil" (*Agile Manifesto*) (BECK et al., 2001), considerado uma referência para essa filosofia de desenvolvimento. Seus principais pontos são: (i) indivíduos e interações devem estar acima de processos e ferramentas; (ii) *software* em funcionamento mais que documentação abrangente; (iii) colaboração com o cliente deve estar acima da negociação contratual; (iv) responder a mudanças acima de se seguir um plano.

Uma das metodologias ágeis mais conhecidas e utilizadas, não só no desenvolvimento de *softwares*, é o *Scrum*. Dentre as organizações que utilizam o ágil, 58% delas fazem uso do *Scrum* e 18% fazem uso de alguma variante do *Scrum* (VERSIONONE, 2020). Tal metologia será descrita na seção a seguir.

### 2.1.1 *Scrum*

O *Scrum* é um *framework* para o desenvolvimento ágil, ou seja, é uma estrutura básica que guia a construção de produtos complexos (como os *softwares*) não prescrevendo como realizar as atividades e sim papéis a serem desempenhados, ritos a serem executados e artefatos a serem utilizados. O *Scrum* utilizada uma abordagem iterativa e incremental para entregar pequenas porções do *software*, com o intuito de reduzir os riscos do projeto e gerar valor de forma crescente (SABBAGH, 2014). A Figura 2.1 exemplifica as etapas e uso de alguns ritos e artefatos do Scrum.

Segundo Cruz (2013), os papéis desempenhados no *Scrum* são:

1. *Product Owner* (PO): Responsável pelo gerenciamento do *Backlog* do produto, atuando como garantidor do valor do trabalho realizado pelo Time, priorizando os itens do *Backlog* e defendendo-os das influências externas ao produto. De modo geral, é responsável por entender o negócio do produto, entregar valor ao cliente e garantir que o Time compreenda o produto;

2. *Scrummaster*: Responsável por garantir que o Time esteja aderido aos valores e práticas do *Scrum*, bem como atuar como um removedor dos impedimentos que possam interferir no objetivo do Time;

3. Time: Conjunto de pessoas responsáveis por transformar o *Backlog* do produto em incrementos de funcionalidades que possam ser entregues ao cliente. Geralmente formado por áreas interdisciplinares, de forma a possuir todo o conhecimento necessário para criar um incremento de trabalho.

Os ritos são descritos por Cruz (CRUZ, 2013) da seguinte forma:

1. *Sprint*: Uma iteração de duração definida, geralmente entre duas a quatro semanas, com uma meta estabelecida e um objetivo claro;

2. Planejamento da *Sprint*: Reunião na qual a *Sprint* é planejada, nela é definido o que será feito e como será feito. A estimativa do esforço das atividades pode ser realizada nesse momento, preferencialmente pelo membro do Time que irá realiza-lá;

3. Reunião diária: Momento para o encontro diário dos membros do time em uma reunião de curta duração, no máximo 15 minutos. O objetivo é que cada membro do Time

exponha o que foi realizado por ele desde a última reunião diária, o que será realizado até a próxima reunião diária e quais são seus obstáculos/impedimentos;

4. Revisão da *Sprint*: Reunião que objetiva a revisão, pelo PO ou pelo cliente, dos itens concluídos pelo Time. Possui duração máxima de quatro horas e é realizada antes das Retrospectivas da *Sprint*;

5. Retrospectivas da *Sprint*: Momento em que o Time inspeciona o que ocorreu na última *Sprint*, leavantando os pontos positivos e que devem ser mantidos, os pontos que podem ser ainda melhores e os itens que devem ser descartados da próxima Sprint. Ao fim, o Time deve possuir uma identificação de medidas de melhoria factiveis que serão implementadas na próxima; *Sprint*. Esse encontro marca o término de uma *Sprint*.

Os artefatos são descritos Schwaber e Sutherland (2020) da seguinte forma:

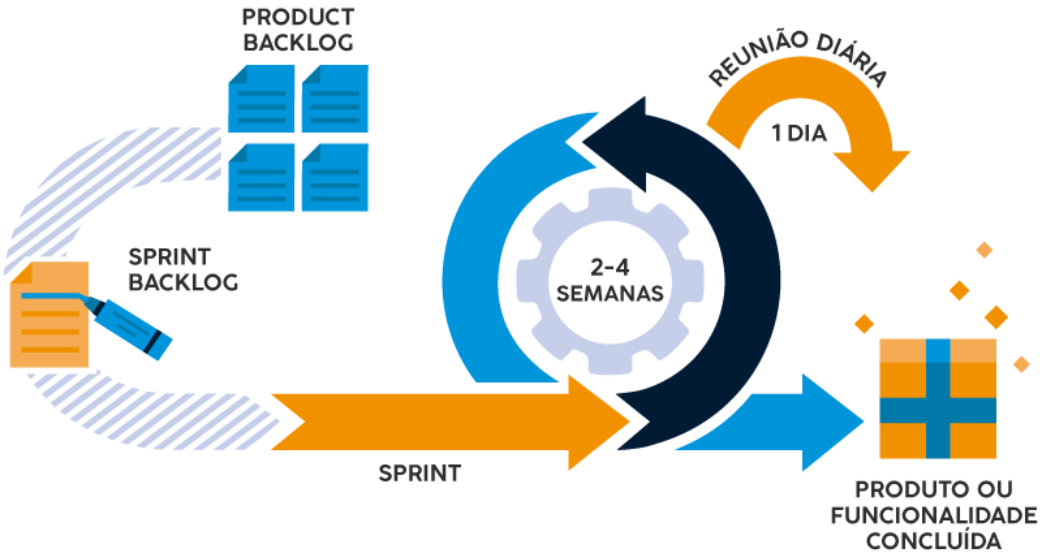
1. *Backlog*: Uma lista ordenada do que é necessário para desenvolver/melhorar o produto. Ela é a única fonte do trabalho a ser realizado pelo Time. Os itens do Backlog que podem ser concluídos durante a *Sprint* são considerados prontos para seleção em um rito de Planejamento da *Sprint*;

2. *Backlog* da *Sprint*: Uma lista que representa o Objetivo da *Sprint*, composta pelo conjunto de itens do *Backlog* selecionados para a *Sprint*, bem como um plano factível para entregar o Incremento. Os Objetivos da *Sprint* são criados e acordados entre os membros do Time durante o evento de Planejamento da *Sprint*;

3. Incremento: Um Incremento é um passo em direção ao Produto idealizado. Cada Incremento se dá pela junção de um novo Incremento à todos os Incrementos anteriores, que por sua vez foram completamente verificados. Para entregar valor o Incremento precisa ser utilizável, dessa forma nenhum trabalho pode ser considerado parte do Incremento a não ser que atinja uma Definição de Pronto combinada pelo Time.

Um dos artefatos adotados para a elaboração dos itens do *Backlog* é chamado de Estórias de Usuário (*User Stories*), que consiste em uma explicação geral sobre um recurso, ou funcionalidade, do produto escrita a partir da perspectiva do usuário, representando as necessidades e desafios diários dos usuários de modo que a solução supra tais necessidades. Schön, Thomaschewski e Escalona (2017) apontam que tal recurso é o mais utilizado no desenvolvimento

Figura 2.1 – Exemplo de funcionamento do *Scrum*




Fonte: Tecnicon (2019).

de *softwares* utilizando metodologias ágeis. Vale ressaltar que uma Estória de Usuário (EU) não está diretamente ligada a uma única atividade, ela pode ser dividida em várias atividades factíveis e isoladas que combinadas atendem ao requisito da EU.

Um dos pontos importantes no evento de Planejamento da *Sprint* é a estimativa dos itens do *Backlog* da *Sprint*. Uma das técnicas é chamada de *Planning Poker*, que parte da premissa que várias pessoas produzem um resultado melhor do que uma (MAHNIČ; HOVELJA, 2012). Dessa forma, o objetivo do jogo é realizar uma estimativa das atividades de maneira assertiva por meio de pontos de estória, que representam a complexidade das atividades, que por fim se traduz em horas de desenvolvimento. Por exemplo, em determinado projeto um ponto de estória pode representar oito horas de desenvolvimento, consequentemente uma atividade de complexidade três levaria 24 horas para ser finalizada.

O *Planning Poker* pode ser conduzido da seguinte forma segundo Grenning (2002): (i) O PO lê a Estória de Usuário ou atividade relacionada a ela e sana as dúvidas do Time caso existam; (ii) cada membro do Time faz sua estimativa; (iii) a estimativa de todos os membros é apresentada simultaneamente; (iv) caso não haja diferença fim da estimativa da atividade; (v) em caso de diferença nas estimativas é preciso que o Time discuta até chegar em um consenso.

Os pontos de estória normalmente seguem um conjunto numérico, podendo ser entendido como o conjunto de cartas disponíveis para o jogo. Exemplo desses conjuntos são a sequência de Fibonacci (1, 2, 3, 5, 8...), sequência de notas de dinheiro (1, 2, 5, 10, 20...), número em potência de dois (0, 1, 2, 4, 8...) etc. Fica a cargo da equipe escolher aquele

conjunto que melhor atende suas necessidades, uma vez que podem existir atividades subestimadas e aquelas sobrestimadas, tais conjuntos buscam diminuir esse deficit através dos intervalos. Ferramentas *online* podem ser utilizadas de forma a facilitar o processo do jogo, como o *Scrumpoker-Online*[1], *ScrumpokerOnline*[2] e *Planning Poker Online*[3]

## 2.2 .NET

A Microsoft define o .NET (comumente pronunciado como *dotNet* ou pontoNet) como uma plataforma de desenvolvimento de *softwares* para a criação de diversos tipos de aplicações como aplicativos da área de trabalho, jogos, aplicações *web*, aplicativos móveis, etc (MICROSOFT, 2020a). Sua primeira versão, o *.NET Framework*, foi lançado em 2002 provendo acesso a diversas APIs e serviços do Windows (sistema operacional de propriedade da Microsoft) possibilitando a criação de aplicativos do Windows (THAI; LAM, 2003; MICROSOFT, 2020a).

Uma aplicação .NET pode ser escrita utilizando três linguagens de programação distintas, que são elas: (i) O C#, uma linguagem de programação orientada a objeto e de tipificação segura; (ii) A F#, que prove o suporte a programação funcional, orientada a objeto e imperativa; e (ii) O *Visual Basic*, também chamado de VB.NET, é uma linguagem de programação orientada a objeto, com sintaxe próxima a língua inglesa, e que é considerada estável, isso é, que não recebe mais atualizações da Microsoft, apenas há suporte para APIs Web (MICROSOFT, 2020a).

Anunciado em 2014 e lançado em 2016, o *.NET Core* é a solução de *software* livre e multiplataformas da Microsoft que sucede o *.NET Framework*, mesmo que esse ainda tenha suporte. Essa nova implementação da plataforma passou a ser chamada apenas de .NET após sua versão 3.1, a próxima versão é o .NET 5, que busca unificar as plataformas até então existentes. A Microsoft evitou o número de versão 4 para evitar confusão entre a nova implementação do .NET e o *.NET Framework*, que está em sua versão 4.8 (MICROSOFT, 2020a).

Com diferentes plataformas .NET coexistindo a Microsoft criou o *.NET Standard*, uma definição formal das APIs .NET a fim de estabelecer a uniformidade e compatibilidade entre as plataformas. Cada implementação do .NET está associada com uma versão do *.NET Standard*. Um exemplo é o .NET Framework 4.6, que implementa o *.NET Standard* 1.3, com isso ele expõe todas as APIs definidas nas versões do *.NET Standard* 1.0 a 1.3 (MICROSOFT, 2020b).

[1] https://www.scrumpoker-online.org/
[2] https://scrumpoker.online
[3] https://planningpokeronline.com

Com a criação do .NET 5, o uso do *.NET Standard* se faz cada vez mais desnecessário em muitos cenários, uma vez que a nova versão da plataforma busca uma unificação das demais implementações. Esta unificação é possível através de um conjunto de recursos e APIs que podem ser usados para a criação de aplicativos de área de trabalho do Windows, aplicativos de console multiplataformas, serviços de nuvem e sites (MICROSOFT, 2020b).

## 2.3 Java

Java é uma linguagem de programação desenvolvida pela Sun Microsystems e lançada em 1995, orientada a objeto de tipificação segura e para propósitos gerais. Em seu lançamento possuía o *slogan* "Escreva uma vez, execute em qualquer lugar" a fim de ilustrar os benefícios da tecnologia multiplataforma do Java, que a tornou muito popular (FARRELL, 2011).

Um programa em Java é compilado para o chamado *bytecode*, uma linguagem intermediária que possui um pequeno e simples conjunto de instruções, que então é executado na *Java Virtual Machine* (JVM). É papel da JVM interagir com o sistema operacional (SO) e computador hospedeiro, assim garantindo que o código escrito em Java será executado em diversos sistemas operacionais e *hardwares* diferentes, além de prover uma camada a mais de segurança contra invasores, sendo esse outro fator que tona o Java uma linguagem de programação popular (DAHM, 1999; FARRELL, 2011).

Assim como outras linguagens, o Java possui diversos *frameworks* para o desenvolvimento de aplicações de nível empresarial, como o Play[4], Struts[5], Spring Framework[6], etc. Atualmente os projetos mantidos pela Spring[7] são os mais populares (MAPLE; BINSTOCK, 2018), como o Spring Boot, Spring MVC, Spring Batch, entre outros. Vale ressaltar que todos eles são de código aberto e com uma grande comunidade de apoiadores.

O Spring Boot é um *framework* para o desenvolvimento de aplicações *backend*, que tem como diferencial uma configuração simples, através de uma convenção de configuração, possibilitando aos desenvolvedores agilidade e tempo para focar nas funcionalidades, não precisando preparar um novo projeto (WALLS, 2016). No geral é um *framework* focado na agilidade do desenvolvimento e no lançamento de novas versões das aplicações, com isso é muito utilizado em arquiteturas de microsserviços e soluções de atualizações constantes.

[4] https://www.playframework.com
[5] https://struts.apache.org
[6] https://spring.io/projects/spring-framework
[7] https://spring.io

## 2.4 *Web Service*

Os *web services* (WS), ou serviços web, promovem o desacoplamento e interoperabilidade entre diferentes aplicações e plataformas, possibilitando a integração de diferentes sistemas, a centralidade de recursos e reutilização de soluções (FERRIS; FARRELL, 2003). Essas características se devem a padronização da forma de comunicação entre as aplicações e o WS, que normalmente utilizam os formatos *Extensible Markup Language* (XML) ou *JavaScript Object Notation* (JSON).

AlShahwan e Moessner (2010) classificam os *web services* em duas categorias, dependendo da sua arquitetura e tecnologia utilizada. São elas:

1. RESTful: *Web services* que seguem as regras estabelecidas pelo Representational State Transfer (REST), sendo essa uma tecnologia, orientada a recursos, definida por Fielding (2000). O REST se trata de um conjunto de restrições e regras que definem a maneira adequada de usar padrões da web, como HTTP e URIs. Embora REST seja originalmente definido no contexto da web, ele está se tornando uma tecnologia de implementação comum para o desenvolvimento de WS, devido a sua simplicidade e leveza.

2. Baseado em *Simple Object Access Protocol* (SOAP): *Web services* que utilizam o protocolo SOAP, que por sua vez é uma tecnologia baseada em orientação a objetos que especifica uma definição de "envelope" XML. Além disso o SOAP defini um conjunto de regras para converter tipos de dados específicos da plataforma/linguagem em representações XML, possibilitando que os tipos dos dados e métodos do WS sejam representados por classes nas aplicações que utilizam o serviço.

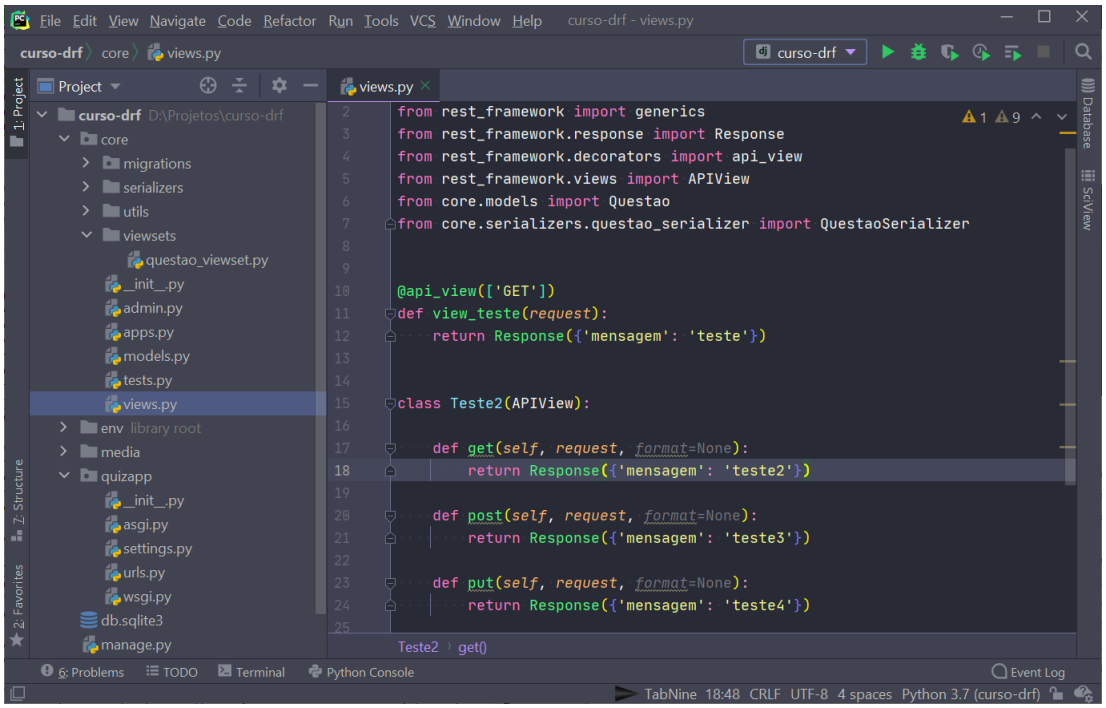
## 2.5 IDE

Os ambientes de desenvolvimento integrados, também conhecidos como IDE (do inglês *Integrated Development Environment*), são *softwares* utilizados por desenvolvedores para automatizar e facilitar o processo de criação de uma nova aplicação (ANDRADE, 2020), através de um conjunto de ferramenta reunidas em uma única interface gráfica do usuário, conhecidas como GUI (do inglês *Graphical User Interface*).

Recursos comuns aos IDEs são as análises e validações de sintaxe em tempo real, destaque da sintaxe com indicadores visuais, fácil navegação entre classes e objetos, visualização de diagramas, gerador de códigos, compilação e execução de aplicações de forma simplificada,

*debbuger* do código etc (RED HAT, 2019; ANDRADE, 2020). Desse modo, o desenvolvedor ganha agilidade e produtividade em suas tarefas, além de economizar o tempo necessário para aprender diversas ferramentas. Na Figura 2.2 está apresentado o IDE PyCharm[8], muito utilizado por desenvolvedores Python.

Figura 2.2 – IDE PyCharm executando no SO *Windows*




Fonte: Autor.

Como apontado pela *Red Hat* (2019), é completamente possível desenvolver aplicações sem o uso de um IDE, apesar de ser uma prática pouco comum se tratando de um contexto empresarial. No mesmo artigo a *Red Hat* apresenta algumas características importantes dos IDEs, como: (i) O suporte a diferentes sistemas operacionais, apesar de algumas IDEs serem desenvolvidas apenas para determinados SOs; (ii) Automação de tarefas; (iii) Possibilidade de extensões, com o uso de *plugins*; e (iv) A quantidade de linguagens compatíveis, como é o caso do IntelliJ[9] que apesar de suportar diferentes linguagens possui o foco no Java.

[8] https://www.jetbrains.com/pycharm/
[9] https://www.jetbrains.com/idea/

## 2.6 Git

Um sistema de controle de versões (SCV) é um *software* capaz de registrar e comparar alterações em arquivos ao longo do tempo, com possibilidade de recuperar, analisar a origem e período de modificações, entre outros recursos (CHACON; STRAUB, 2014).

O Git é um SCV de código aberto, distribuído sob a licença GPLv2[10], desenvolvido por Linus Torvalds em 2005 como uma alternativa ao BitKeeper[11] e com a finalidade de ser utilizado para o controle de versões do kernel Linux, *software* de código aberto criado por Torvalds e mantido por toda uma comunidade. As premissas iniciais do Git eram: (i) rapidez; (ii) design simples; (iii) suporte forte para desenvolvimento não linear (milhares de ramificações paralelas); (iv) totalmente distribuído; e (v) capacidade de lidar com grandes projetos com eficiência (velocidade e tamanho dos dados) (CHACON; STRAUB, 2014).

Uma diferença importante do Git em relação aos seus antecessores se dá pela qualidade das revisões que podem ser realizadas no *software*. Ao gerenciar as revisões, o Git permite que o desenvolvedor selecione precisamente quais mudanças serão integradas por completo, até mudanças parciais em um único arquivo. Outro recurso importante é fato do Git manter um histórico completo, em formato de grafo, mostrando quais mudanças foram mescladas em quais ramos, permitindo assim que os desenvolvedores pensem em termos de revisões integradas, no lugar de alterações em arquivos de ramos distintos (SPINELLIS, 2012).

O GitLab[12] é uma aplicação *web* utilizada para gerenciar repositórios Git. Com ela, é possível que as equipes projetem, desenvolvam e gerenciem com segurança o código e os dados de projetos a partir de um único sistema de controle de versão distribuído, permitindo uma iteração rápida e entrega de valor (GITLAB, 2020).

### 2.6.1 *Git-Flow*

O *Git-Flow* é um modelo de ramificações proposto por Vincent Driessen (2020) e amplamente utilizado pela comunidade. Seus principais ramos possuem tempo de vida infinito. Ou seja, evoluem junto ao *software*. São eles:

1. O ramo *master*: O principal ramo do repositório. O código presente na sua "cabeça" (o último estado do ramo) está pronto para o ambiente de produção.

[10] https://www.gnu.org/licenses/old-licenses/gpl-2.0.html
[11] Um SCV proprietário. Mais informações em http://www.bitkeeper.org
[12] https://about.gitlab.com

2. O ramo *develop*: O ramo onde o código-fonte da cabeça sempre reflete um estado com as últimas mudanças de desenvolvimento entregues para a próxima versão.

3. Ramos temporários: Utilizados para desenvolver novas funcionalidades, correções de problemas ou refinamentos para um lançamento.

Driessen (2010) propõe o fluxo de trabalho onde: (i) cada nova funcionalidade deve ser criado a partir do ramo *develop*; (ii) quando determinada funcionalidade estiver concluída seu ramo deve ser mesclado ao ramo *develop*; (iii) quando uma versão estável estiver preparada, o ramo *develop* deve ser mesclado ao ramo *master*; e (iv) para problemas que necessitam de um reparo imediado, um novo ramo deve ser criado a partir do ramo *master* e quando seu desenvolvimento for concluído tal ramo deve ser mesclado aos ramos *develop* e *master*. Na Figura 2.3 está exemplificado o uso do fluxo.

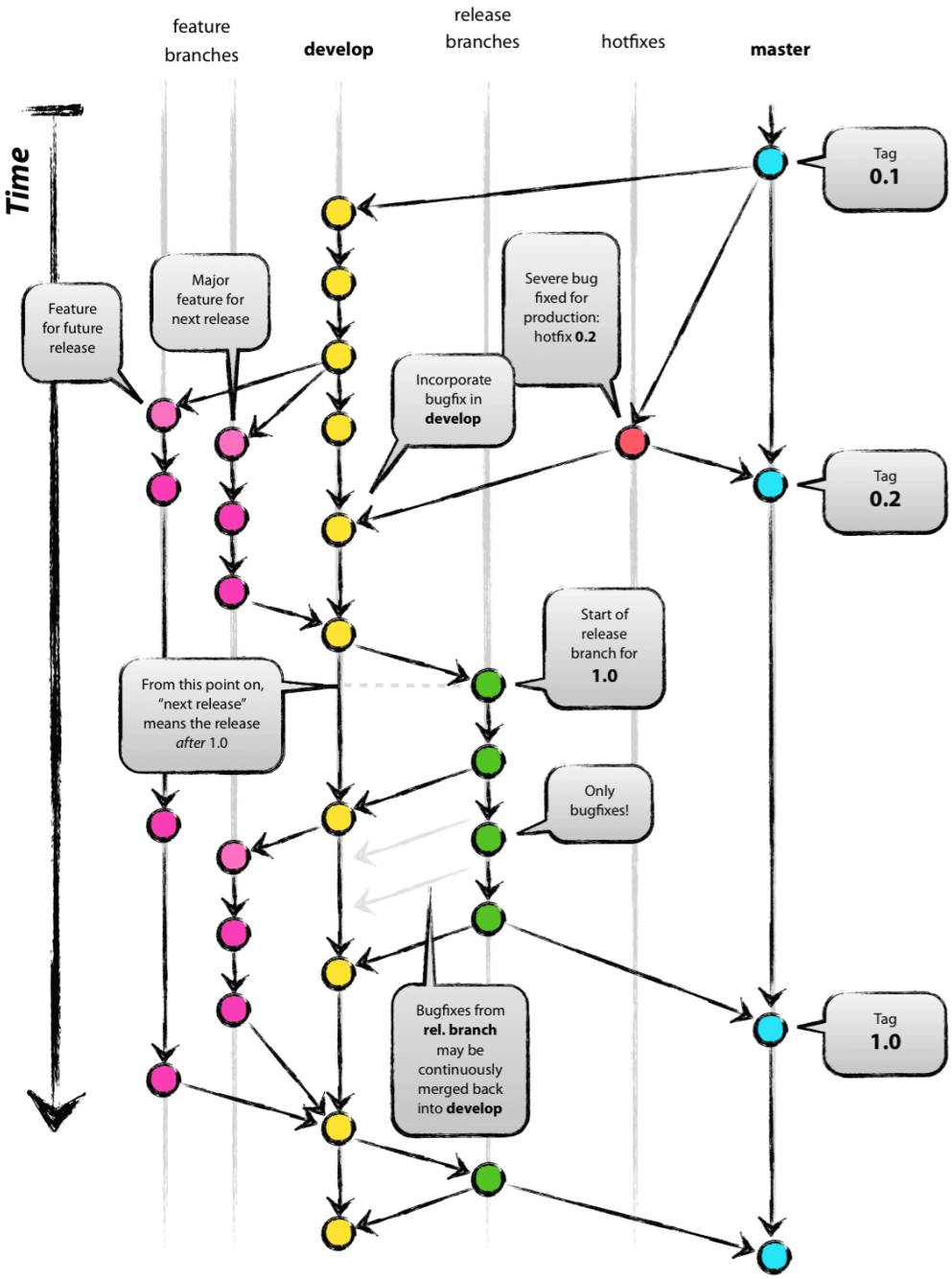
Uma atividade corriqueira em projetos colaborativos de desenvolvimento de *softwares*, que utilizam ferramentas como o GitLab, é chamada *Merge Requests*[13] (MR), em português é conhecido como solicitação de mesclagem. A interface da aplicação provê uma lista das alterações realizadas no ramo e uma visualização de comparação, que mostra as diferenças em cada arquivo entre o ramo de desenvolvimento e o ramo em que será feita a mesclagem. Dessa forma os membros do time ou a comunidade, no caso de um projeto livre, podem discutir a respeito do entendimento, se necessário, bem como propor melhorias no código através dos comentários. Quando estiver satisfeito, um membro da equipe pode mesclar o ramo no ramo alvo a partir da interface do MR. A ferramenta usará o Git para lidar com a mesclagem, e quaisquer conflitos detectados pelo Git serão anotados no MR, esses que podem ser resolvidos antes da mesclagem (ENGWALL; ROE, 2020).

## 2.7 Banco de Dados Relacional

Bancos de dados são coleções estruturadas de dados, ou informações, organizadas e controladas geralmente por um Sistema Gerenciador de Bancos de Dados (SGBD). Esses dados representam fatos não processados, que isolados podem não significar nada, enquanto informações representam dados processados, contextualizados e com alguma significância, seja por organização dos dados, processamento estatístico ou inferências (ROB; CORONEL, 2011).

[13] Também conhecido como *Pull Request* (PR).

Figura 2.3 – Exemplo de ramos segundo o *Git-Flow*.



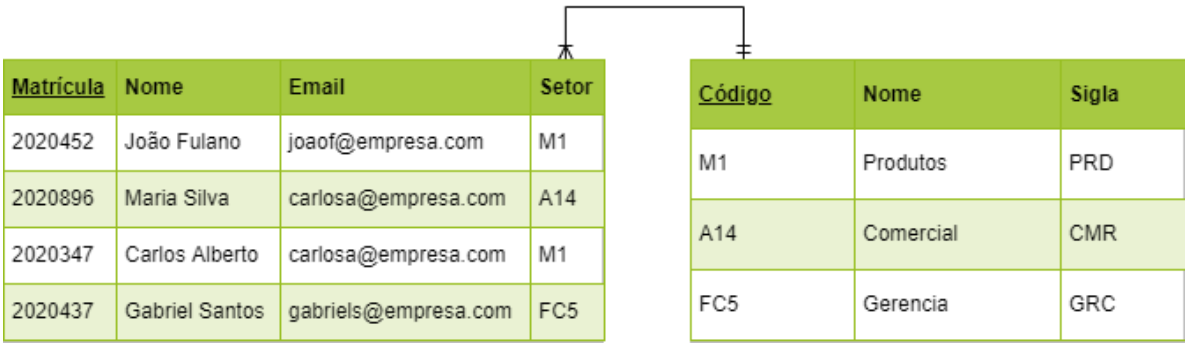

Fonte: Driessen (2010).

Em meados da década de 60 surgiram os primeiros SGBDs comerciais, que proviam o armazenamento e gerenciamento dos dados de forma independente da aplicação, como alternativa ao cenário da época, onde os dados eram mantidos em arquivos vinculados as aplicações (BOSCARIOLI et al., 2006). Um SGBD serve como intermediário único entre diversos usuários e os repositórios de dados, permitindo que os dados sejam compartilhados entre diversos

usuários e aplicações, bem como diferentes visões possam ser aplicadas sobre eles (ROB; CORONEL, 2011).

O modelo de banco de dados relacional foi proposto por Codd (1970), objetivando a separação do armazenamento físico dos dados de sua representação conceitual, utilizando uma base matemática para representar os dados e prover o acesso a eles (ELMASRI; NAVATHE, 2011). Tal modelo introduziu linguagens de consulta de alto nível, sendo a SQL, originalmente nomeada de SEQUEL (*Structured English Query Language*), a mais difundida e utilizada pela maioria dos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) hoje em dia (ELMASRI; NAVATHE, 2011; CHAMBERLIN et al., 1981). Na Figura 2.4 etá exemplificado uma representação de um banco de dados relacional, comumente representado por tabelas.

Figura 2.4 – Exemplo de banco de dados que armazena informações de funcionários de uma empresa.



| Matrícula | Nome | Email | Setor |
|---|---|---|---|
| 2020452 | João Fulano | joaof@empresa.com | M1 |
| 2020896 | Maria Silva | carlosa@empresa.com | A14 |
| 2020347 | Carlos Alberto | carlosa@empresa.com | M1 |
| 2020437 | Gabriel Santos | gabriels@empresa.com | FC5 |

| Código | Nome | Sigla |
|---|---|---|
| M1 | Produtos | PRD |
| A14 | Comercial | CMR |
| FC5 | Gerencia | GRC |

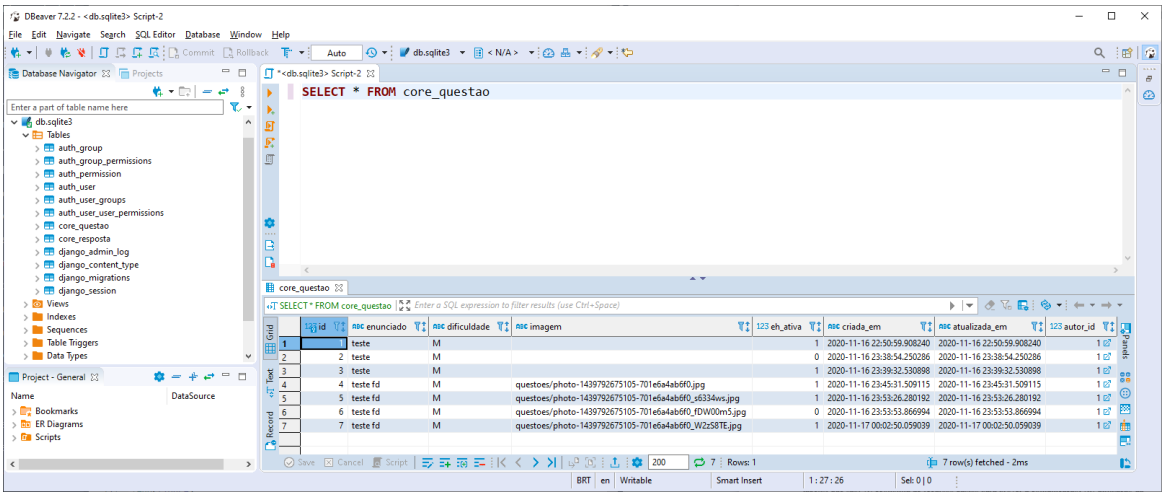
Fonte: Autor.

A interação com diversos bancos de dados e diferentes servidores SGBDs pode ser uma tarefa morosa e pouco produtiva via linha de comando. Atualmente diversos programas provêm uma interface de gráfica para facilitar a interação e visualização dos dados de diversos SGBDs. Exemplos desse tipo GUI são o SQL Server Management Studio[14], DBeavear[15], MySQL Workbench[16], etc. Na Figura 2.5 está apresentada a interface gráfica do DBeaver após uma consulta SQL.

Outro mecanismo que auxilia os desenvolvedores, dessa vez no desenvolvimento das aplicações, é o Mapeamento objeto-relacional, do inglês *Object-relational mapping* (ORM). Através dele é possível os sistemas utilizarem a orientação a objetos representando as tabelas do banco de dados através de classes, as relações como objetos da instância correspondente, além de prover uma Interface de Programação de Aplicação, do inglês *Application Program-*

[14] https://docs.microsoft.com/pt-br/sql/ssms/
[15] https://dbeaver.io
[16] https://www.mysql.com/products/workbench/

Figura 2.5 – Exemplo do resultado de uma consulta SQL apresentada na GUI DBeavear.




Fonte: Autor.

*ming Interface* (API), que gerencia as transações e a persistência dos dados no banco de dados (O'NEIL, 2008). Exemplos de ORMs são o *Hibernate*[17], para Java, e o *Entity Framework*[18], para a plataforma .NET.

Outro exemplo é o ORM do *framework* Django[19] (para Python), que possibilita a estruturação e abstração do banco de dados através de *Models*. Com isso não importa qual SGBDR será utilizado, o framework se encarregará de criar e manter os dados e tabelas. Na Figura 2.6 está ilustrado a modelagem de uma tabela através da classe *Questao*, que é totalmente gerida pelo Django. Um exemplo de criação de um novo registro utilizando essa classe está apresentado na Figura 2.7.

Figura 2.6 – Classe Python que abstrai um modelo de dados Questão.

```
class Questao(models.Model):
    enunciado = models.TextField('Enunciado', max_length=500)
    difulcudade = models.CharField('Dificuldade', choices=DIFICULDADE, max_length=1)
    eh_ativa = models.BooleanField('Ativa', default=True)
    autor = models.ForeignKey(Autor, verbose_name='Autor', on_delete=models.PROTECT)
    criada_em = models.DateTimeField(auto_now_add=True)
    atualizada_em = models.DateTimeField(auto_now=True)
```

Fonte: Autor.

[17] http://hibernate.org/orm/
[18] https://docs.microsoft.com/pt-br/ef/
[19] https://www.djangoproject.com

Figura 2.7 – Exemplo de criação de uma novo registro utilizando o ORM do Django.

```
questao = Questao()
questao.enunciado = 'Enunciado da questão'
questao.autor = User.objects.get(pk=1)
questao.eh_ativa = True
questao.difulcudade = 'F'
questao.save()
```

Fonte: Autor.

## 2.8 Testes unitários

Segundo Mackinnon, Freeman e Craig (2000) os testes unitários são fundamentais em metodologias ágeis, onde através de programas ou *scripts* é possível testar trechos de código de uma aplicação, de forma a garantir seu funcionamento e comportamento. Silva e Moreno 2008 definem como unidade o menor trecho de código de uma aplicação que pode ser testado, podendo esse ser um método em programas orientados a objetos, uma função ou módulo em programas procedimentais etc. Dessa forma, o teste de uma unidade objetiva testar uma funcionalidade de forma isolada e independente, facilitando que erros sejam encontrados e que o sistema em desenvolvimento evolua sem afetar as funcionalidades já desenvolvidas.

O desenvolvimento dos testes unitários é, em muitas das vezes, de responsabilidade da equipe de desenvolvimento e podem ser elaborados antes, durante ou após o desenvolvimento de uma nova funcionalidade. No entanto, o recomendável é que eles sejam criados antes do início de um novo desenvolvimento, seguindo as práticas de TDD (Desenvolvimento Orientado a Testes) (SILVA; MORENO, 2012). Desse modo quando os métodos forem ser desenvolvidos esses deverão seguir um padrão e poderão ser validados quanto à sua corretude, possibilitando que a funcionalidade seja dada como pronta quando seu teste for aprovado. Outra vantagem do uso desse tipo de teste é quanto a validação automatizada do *software*, onde todos os casos de teste podem ser facilmente e rapidamente executados inúmeras vezes, a qualquer momento e com pouco esforço computacional (BERNARDO; KON, 2008).

Para isso existem diversos *frameworks* que auxiliam no desenvolvimento de testes, como JUnit[20] para Java, NUnit[21] para .NET e CppTest[22] para C++. Outro tipo de *framework* relevante na construção dos testes são aqueles que possibilitam a abstração da camada de persistência da

[20] https://junit.org
[21] https://nunit.org
[22] https://cpptest.sourceforge.io

aplicação, como Mockito[23] e Moq[24], de forma que as regras de negócio podem ser testadas independente de como será feito o acesso aos dados pela aplicação. Na Figura 2.8 está exemplificado o uso dos *frameworks* JUnit e Mockito para a construção de um teste unitário.

Figura 2.8 – Exemplo de uma classe de teste em Java utilizando Spring Boot, JUnit e Mockito.

```
@RunWith(SpringJUnit4ClassRunner.class)
public class MinhaServiceTest {

    @InjectMocks
    private MinhaService service;

    @Mock
    private MeuRepository repository;

    @Before
    public void initMocks() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    public void processamentoConcluidoComSucesso() {
        Mockito.when(repository.consultarDados(anyString())).thenReturn(true);

        ResultadoProcessamentoRecebivel resultado = service.processarEntrada(1122);

        assertNotNull(resultado);
        assertEquals(true, resultado.concluido);
    }
}
```

Fonte: Autor.

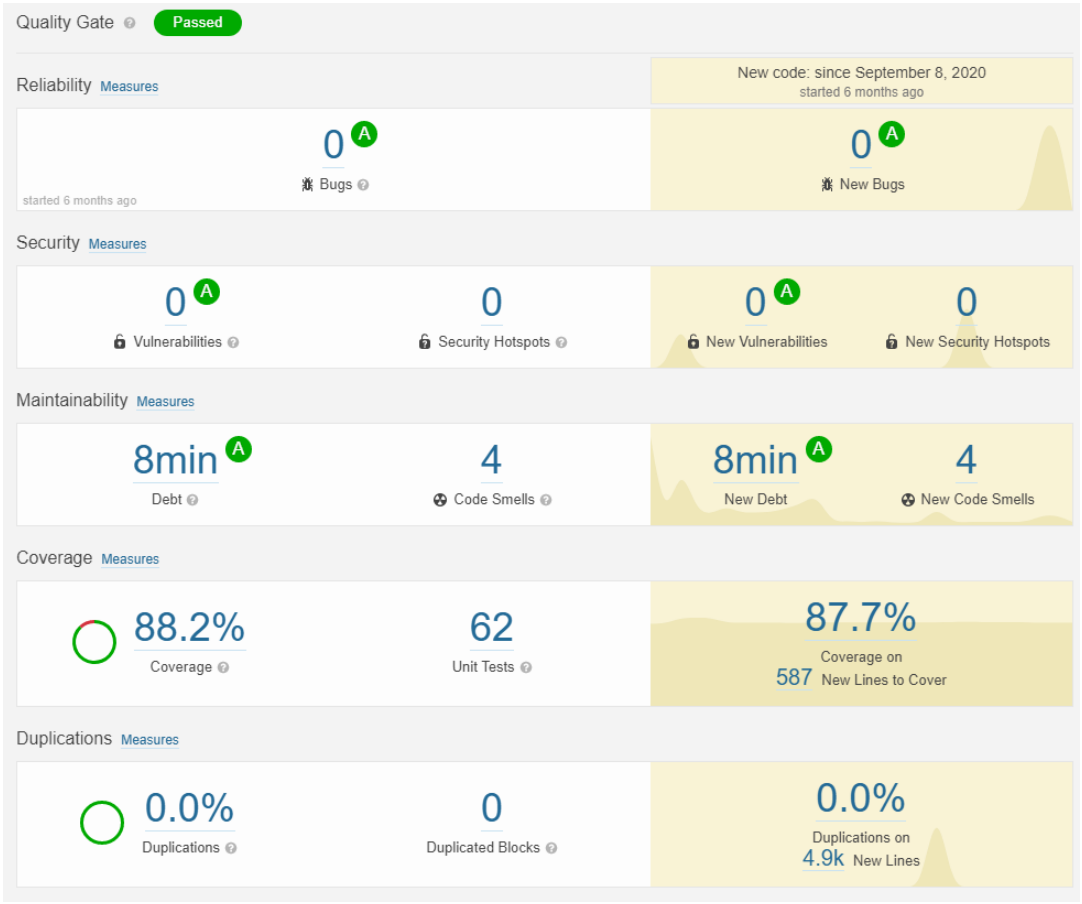
## 2.9 SonarQube

SonarQube é uma ferramenta automática utilizada para a revisão de código a fim de detectar *bugs*, vulnerabilidades e *code smells*[25]. É utilizada junto a um fluxo de desenvolvimento de *software* automatizado ou parcialmente automatizado, no qual, em determinada etapa, uma ferramenta de integração contínua verifica, compila e executa os testes unitários, em sequência o SonarQube analisa os resultados e notifica os envolvidos (SONARSOURCE, 2021).

Para gerar os indicadores o SonarQube calcula diversas métricas como o número de linhas de código, a complexidade do código e a conformidade do código frente a um conjunto de "regras de codificação", as quais são definidas para a maioria das linguagens de desenvol-

[23] https://site.mockito.org
[24] https://www.nuget.org/packages/Moq
[25] Característica identificada em um trecho de código que possivelmente indica um problema.

Figura 2.9 – Exemplo de relatório gerado pela ferramenta SonarQube.




Fonte: dti digital.

vimento utilizadas pelo mercado (LENARDUZZI et al., 2020). Na Figura 2.9 está apresentado um relatório gerado pela ferramenta, na qual é possível observar a evolução do *software* em desenvolvimento através da comparação entre versões.

Outro recurso importante provido pelo SonarQube são os chamados *Quality Gates*, que asseguram a qualidade e confiabilidade do código analisado, sendo ele julgado pronto ou não para entrega, através de um conjunto de regras definidas pela organização (SONARSOURCE, 2021). Tais regras são chamadas “Perfis de Qualidade”, que definem através de expressões booleanas os critérios de aceitação do novo código.

Um exemplo de “Perfil de Qualidade” poderia ser: (i) A Cobertura dos testes unitários no novo código deve ser superior a 80%; e (ii) As linhas duplicadas no novo código devem ser menores que 6%; e (iii) A avaliação de segurança do novo código deve ser “A”. Vale ressaltar que tais perfis são definidos por linguagem de programação na ferramenta SonarQube.

## 2.10 *Virtual Private Network*

*Virtual Private Network* (VPN) é definida como uma rede privada construída sobre a infraestrutura de uma rede pública, como a Internet. É também o modelo popular para fazer uso da Internet como infraestrutura de acesso a uma Intranet, por exemplo, a fim de prover a comunicação entre a matriz de uma empresas e suas filiais. Para isso é necessário um servidor VPN e uma aplicação VPN, chamado de cliente VPN, para se conectar à rede privada. Usar VPN em vez de redes físicas alugadas tem ao menos três vantagens, que são: (i) economia de recursos gastos para prover a comunicação; (ii) utilização da onipresença da Internet; e (iii) conveniência de uma construção e manutenção dinâmica (AQUN et al., 2000).

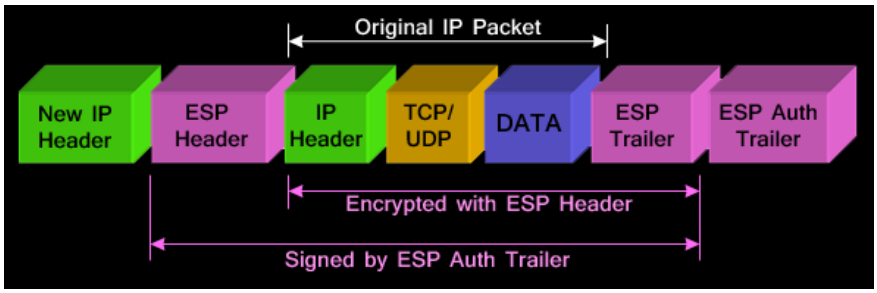
A ExpressVPN (2020), empresa que oferece planos de acesso a sua VPN, destaca os seguintes recursos oferecidos por VPNs: a privacidade da navegação, uma vez que o servidor VPN atua também como um *proxy*, camuflando o real endereço do usuário; a autenticação, de forma que somente usuários permitidos podem acessar a rede privada; o tunelamento, que proporciona uma camada a mais de abstração para a comunicação, dificultando a leitura dos dados encapsulados; e a segurança, através da criptografia dos dados trafegados utilizando a VPN.

Para que os recursos sejam disponibilizados por uma VPN é preciso que sua implementação seja feita por algum mecanismo de encapsulamento. Ou seja, um protocolo (protocolo X) é encapsulado dentro de outro protocolo (protocolo Y) durante o transporte, de modo que o protocolo X seja transparente para a rede pública. Um exemplo é utilizar o *Internet Protocol* (IP) como protocolo de encapsulamento para implementar a VPN baseada na Internet, que encapsula um pacote *IP Security Protocol* (IPSec) como um segundo protocolo de encapsulamento, que prove segurança e confidencialidade a comunicação, e por fim é encapsulado por outro pacote IP (AQUN et al., 2000). A Figura 2.10 exemplifica o resultado final de um encapsulamento utilizando VPN.

## 2.11 Ferramentas para acompanhamento de projetos

O Microsoft Teams é uma aplicação em nuvem, que oferece reuniões *online* por voz e vídeo, conversa por texto, compartilhamento de arquivos e mais, tudo isso em uma única plataforma a fim de integrar pessoas, conteúdos e ferramentas para uma equipe mais engajada e eficaz (MICROSOFT, 2021). Outras ferramentas que se assemelham ao Microsoft Teams são

Figura 2.10 – Encapsulamento utilizando IPSec com cabeçalho *Encapsulating Security Payload* (ESP).




Fonte: Firewall.cx (2012).

o Slack[26] e Rocket.Chat[27] que proveem recursos semelhantes, no entanto com custos, planos e integrações variadas. O Microsoft Teams geralmente é preferido por empresas que já utilizam outros serviços da Microsoft, como plano de *e-mail* e gerenciador de identidades.

Jira é uma ferramenta para o acompanhamento do desenvolvimento de *software*, muito utilizado por equipes ágeis, lançado em 2002 pela Atlassian Corporation (FISHER; KONING; LUDWIGSEN, 2013). Seu objetivo é fornecer aos membros de uma equipe de desenvolvimento de *software* meios para planejar, rastrear e lançar *softwares* com qualidade. Dentre suas principais funcionalidades estão a criação de fluxos de trabalho personalizados, planejamento de *sprints* e EUs, relatórios e integrações com outros produtos (ATLASSIAN CORPORATION, 2021).

[26] https://slack.com
[27] https://rocket.chat

# 3 O ESTÁGIO

Esta seção apresenta os processos utilizados antes e durante o desenvolvimento de uma nova demanda, os projetos de atuação e seus respectivos escopos. Atendendo ao pedido da dti e do cliente, nenhum código fonte, modelo ou regra de negócio serão apresentados.

## 3.1 A empresa

O estágio foi realizado na empresa dti digital, fundada em 2009, a qual possui sede em Belo Horizonte - MG com dois escritórios e uma filial em Lavras - MG. Hoje conta com mais de 800 colaboradores e um total de 38 clientes, sendo 3 deles internacionais, de diversas áreas e ramos de atuação. Sua estrutura organizacional é baseada em *squads*, sendo esse um desdobramento da metodologia ágil, que por sua vez permite resultados mais rápidos na gestão de projetos (SANTOS; FERREIRA, 2020).

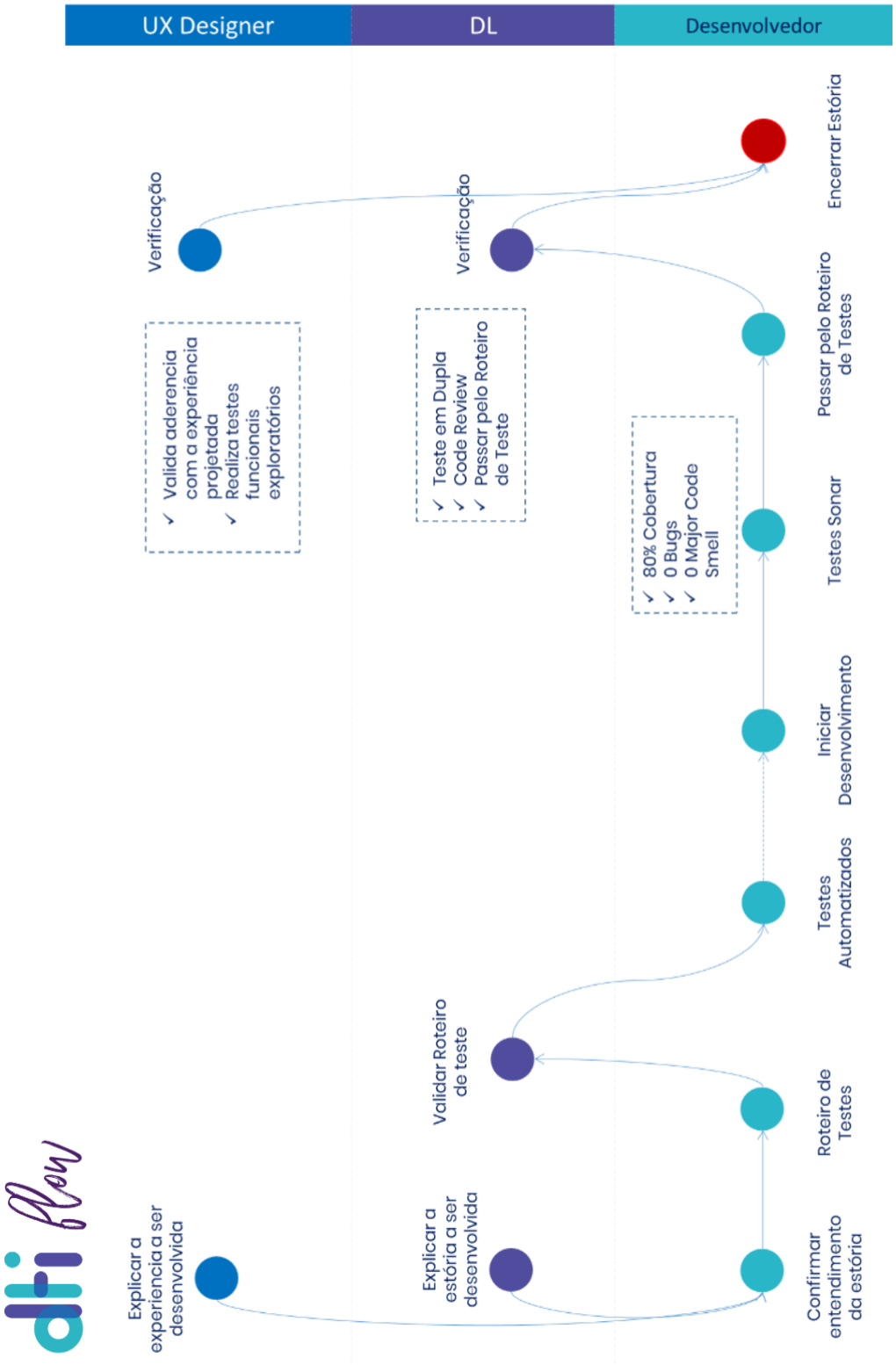
O estagiário fez parte dos colaboradores alocados na filial de Lavras, no entanto durante todo o período de estágio os trabalhos foram realizados de forma remota, devido aos desdobramentos da COVID-19. O cliente atendido pelo *squad* do estagiário se trata de uma instituição financeira (IF).

## 3.2 Processo de desenvolvimento

Após diversos projetos entregues, vários acertos e inúmeros erros, a dti desenvolveu um modelo (chamado dti flow) a ser seguido por todos os *squads* da empresa como processo de desenvolvimento de softwares. Esse modelo se trata de uma sequência de etapas definidas de forma empírica, que auxilia os membros das equipes a entregarem resultados satisfatórios e que atendam às necessidades e expectativas dos clientes. Na Figura 3.1 está apresentado este modelo, no qual se encontram os papéis dos membros dos *squads*, representados pelos retângulos, sendo eles o UX Designer, Desenvolvedor Líder (DL) e Desenvolvedor, e os círculos retratam as etapas do processo de forma sequencial.

Como os projetos nos quais o estagiário atuou se trataram de manutenção e desenvolvimento de novas funcionalidade para sistemas legados, que na maioria das vezes se tratavam de demandas relacionadas ao *back-end* das aplicações, o *squad* não contava com um membro responsável pelo UX Design. Dessa forma, as atividades atribuídas a tal cago não eram realizadas.

Figura 3.1 – dti flow




Fonte: dti digital.

Durante o período do estágio, a solicitação de um serviço até sua entrega, seguiu as seguintes etapas:

1. Uma nova demanda é encaminhada pelo cliente em forma de EU, então uma análise é realizada pelo time a fim de entender a solicitação e levantar dúvidas. Com isso, uma reunião é agendada com os representantes do cliente, responsáveis por acompanhar tal demanda, onde o entendimento da estória e as dúvidas são discutidas. Além disso, são tratados os detalhes técnicos da implementação, como complexidade e pontos de atenção.

2. Os membros do *squad* se reúnem para a quebra da demanda em atividades objetivas e realizáveis. Na sequência, cada atividade passa por uma análise de risco e estimativa de esforço, realizada através de *planning poker*. Com isso é possível que os membros priorizem as atividades de alto risco e estimem um prazo compatível com o conhecimento de todo o *squad*.

3. O Desenvolvedor se reúne com o DL antes do início de uma atividade para confirmar o entendimento, garantindo assim que o objetivo será atendido conforme o esperado. Em seguida, cabe ao Desenvolvedor realizar os roteiros de testes, englobando os cenários de erro e sucesso, e então os validar junto ao DL ou outro membro do *squad*. Tal ação busca evitar que o roteiro seja raso ou que algo tenha sido esquecido, além de nortear o desenvolvimento.

4. Quando possível, caso a tecnologia ou o projeto suporte/utilize, o Desenvolvedor cria os testes automatizados para então iniciar sua atividade.

5. Ao término de sua tarefa, quando aplicável, é necessário que o Desenvolvedor verifique o resultado da análise feita pelo SonarQube, buscando uma cobertura miníma de 80% do código, 0 *code smells* e 0 *bugs*.

6. Na sequência, é realizado o roteiro de testes junto a outro membro do *squad*. Para assim, identificar casos não abordados pelos testes e/ou melhorias possíveis.

7. Por fim, é preciso que o Desenvolvedor solicite o MR do ramo de trabalho, criado para o desenvolvimento da atividade, para o ramo *develop*. A revisão de código é realizada por outro membro do *squad*, que avalia a qualidade e identifica possíveis melhorias.

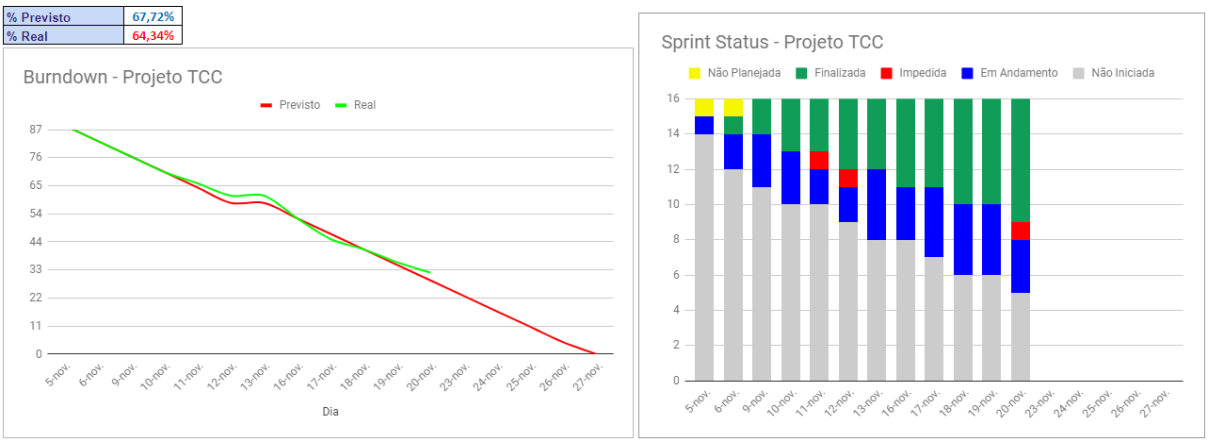
8. Terminada as etapas anteriores, o MR é realizado e os sistemas alterados estão prontos para homologação do cliente.

9. Durante a homologação, caso um erro seja identificado, os membros do *squad* investigam a causa, para então reexecutar as etapas 4, 5, 6 e 7, quando aplicáveis.

### 3.2.1 Acompanhamento das atividades

Durante o desenvolvimento das atividades, diferentes ferramentas são utilizadas para o acompanhamento e visualização das pendências. Internamente, é utilizado um *template* de planilha fornecido pela dti, onde no início de cada *sprint* ela é alimentada com as atividades, seus pontos de esforço e risco estimado. A partir dessa planilha, gráficos são gerados para o envio do acompanhamento diário via e-mail aos envolvidos no projeto. Na Figura 3.2 está exemplificado tais insumos gerados.

Figura 3.2 – Exemplo de gráficos gerados para o acompanhamento dos projetos




Fonte: Autor.

O cliente utiliza a ferramenta Jira para o acompanhamento de seus projetos, assim é preciso que as atividades a serem desenvolvidas sejam cadastradas e também atualizadas nessa plataforma.

O controle dos roteiros de testes escritos pelos Desenvolvedores, é realizado através de planilhas individuais para cada demanda, onde é inserido o objetivo do cenário de teste, o resultado esperado, observações caso necessário, o status do teste do Desenvolvedor e o status final após o teste em dupla. Na Figura 3.3 está exemplificado o uso dessa planilha.

Diariamente é realizada a *daily meeting*, nela participam todos os desenvolvedores envolvidos no projeto, mesmo que esses sejam de outras empresas, os testers - caso o projeto possua -, o responsável da área de negócios e um analista de sistemas da IF.

Figura 3.3 – Exemplo de planilha para o acompanhamento de testes

| Status Geral dos Fluxos de Teste | | | Aguardando Teste | | |
|---|---|---|---|---|---|
| **Fluxo** | **Alterar tela tal** | | **Status do Fluxo** | **OK** | |
| **1** | **Descrição** | **Resultado Esperado** | **Observação** | **Status Developer** | **Status Final** |
| 1.1 | O campo x deve aceitar no máximo 35 caracteres | Entrada limitada em 35 carecteres | | OK | OK |
| 1.2 | O campo y não deve aceitar caracteres diferentes de número | Apenas números são de fato inseridos no campo. | | OK | OK |
| 1.3 | O campo y deve realizar uma máscara de valor no formato ###.###,## | Máscara é aplicada para qualquer quantidade de números informado. | | OK | OK |
| **Fluxo** | **Ajustes funcionalidade Z** | | **Status do Fluxo** | Aguardando Teste | |
| **2** | **Descrição** | **Resultado Esperado** | **Observação** | **Status Developer** | **Status Final** |
| 2.1 | Tentar submeter formulário sem todos os campos preenchidos. | Ação não realizada. Mensagem de erro deve informar qual(is) campos estão faltantes. | | OK | OK |
| 2.2 | Tentar submeter formulário sem todos os campos válidos. | Ação não realizada. Mensagem de erro deve informar qual(is) campos estão inválidos. | | OK | Erro |
| 2.3 | Clicar no botão de "Pesquisar" sem nenhum campo de filtro. | Deverá apresentar a mensagem "Nenhum parâmetro de pesquisa informado." na seção "Registros Encontrados". | | OK | Aguardando Teste |

Fonte: Autor.

A cada quinzena, são realizados pelo membros do *squad* ritos estabelecidos pela dti. São eles:

1. *Check* de execução: Busca transparecer para a liderança qual a visão do *squad* em relação a execução do projeto, a aderência às práticas estabelecidas, e caso necessário, um plano de mitigação para possíveis riscos;

2. Reunião de retrospectiva das atividades: São levantados pontos positivos para o time e que devem continuar, bem como aqueles a serem melhorados. Para os tópicos “a melhorar”, são estipuladas ações a serem realizadas até a próxima reunião. Também é avaliado a evolução dos itens "a melhorar"apontados na última retrospectiva.;

3. Pré-*check*: Reunião, agendada conforme necessário, para avaliar a necessidade de se realizar um *check* arquitetural.

4. *Check* arquitetural: Encontro com especialistas externos ao time, a fim de verificar a aplicação em desenvolvimento ou a ser desenvolvida, sob o ponto de vista técnico, abordando ideias e propostas de soluções.

## 3.3 Os projetos

Durante o período de estágio o estagiário participou de diversos projetos para o mesmo cliente. Eles estão apresentados conforme o cronograma da Figura 3.4. A primeira coluna da tabela apresenta um código de identificação que será utilizado ao longo deste trabalho; a segunda coluna, por sua vez, apresenta a descrição da atividade de forma anonimizada; as

colunas seguintes apresentam o intervalo de tempo e então um gráfico mostra de forma visual tal período.

Inicialmente foi realizado um treinamento da linguagem Visual Basic (VB) [A1] onde algumas aplicações de console foram construídas a fim de conhecer e fixar a sintaxe e recursos da linguagem. Posteriormente esse treinamento se expandiu para o VB.NET, onde foi construída uma aplicação web de duas páginas, utilizando o modelo arquitetural *Model–View–Controller* (MVC), a fim de listar, paginar, filtrar e inserir novos registros em uma base de dados.

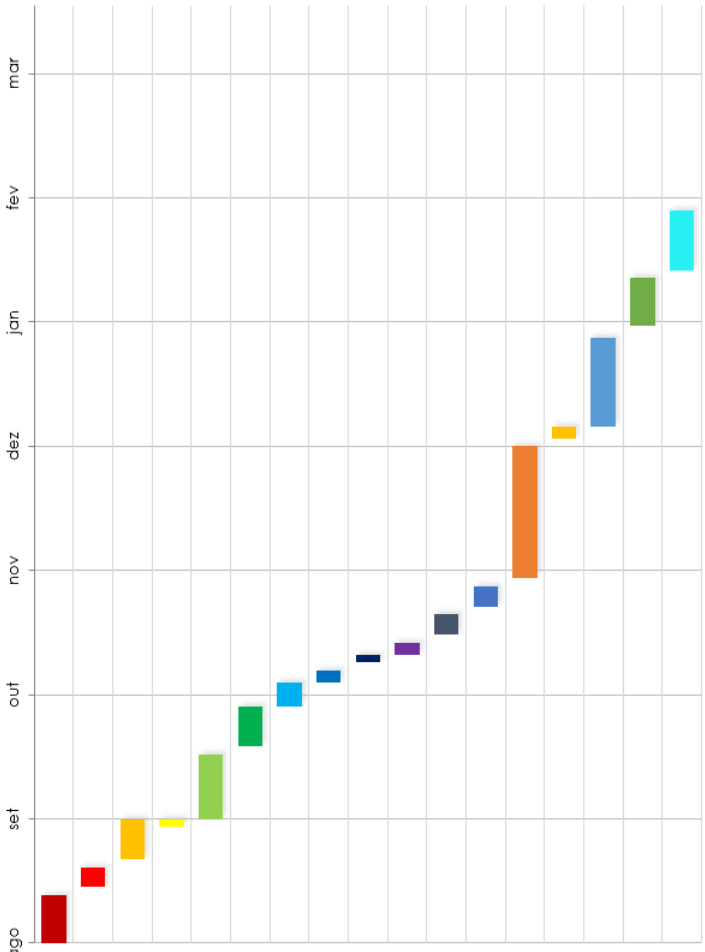
Posteriormente o estagiário acompanhou junto a outros desenvolvedores [A2] o desenvolvimento de algumas atividades do Sistema 1, no qual logo viria a atuar. Tal sistema se trata um boletador, onde diversas operações são inseridas e passam por fluxos de aprovação da IF. O projeto teve seu desenvolvimento iniciado em 2012, o qual utiliza uma arquitetura própria, semelhante ao MVC, e vem evoluindo desde então, realizando interação com diversos serviços e bases de dados, dessa forma tem uma alta complexidade e dependências para seu funcionamento.

Após um período de desenvolvimento de funcionalidades para o Sistema 1 [A3] e para a API 1 [A4], foi realizado a migração de tecnologia da Rotina 1 [A5], que se tratava de um processo desenvolvido em VB 6 que validava diversos dados e prosseguia com o fluxo de determinadas operações. A nova aplicação foi desenvolvida em C# utilizando .NET Framework, a pedido do cliente, de modo a se comportar da mesma forma que o seu antecessor. Posteriormente foi necessário implementar novas funcionalidades nessa aplicação [A6], de forma a possibilitar a comunicação com outro serviço que realiza validações e registro de operações.

Algumas atividades de curta duração [A8][A9][A10][A11] foram realizadas objetivando correções ou desenvolvimentos pontuais em determinadas aplicações. Com a atividade [A13] foi possível ter maior interação com os representantes do cliente, com algumas bases de dados SQL utilizadas pela IF, sendo elas o MS SQL Server[1], Sybase ASE[2] e Oracle Database[3], e também contato com alguns serviços utilizados pelo Sistema 1, como Web Services SOAP e REST. Tal atividade consistiu em implementar uma variação de um produto do Sistema 1, sendo necessário a integração com sistemas de outros fornecedores e o desenvolvimento de uma nova funcionalidade para a API 4, que consiste em um Web Service SOAP desenvolvido em VB.NET.

[1] https://www.microsoft.com/pt-br/sql-server
[2] https://www.sap.com/products/sybase-ase.html
[3] https://www.oracle.com/database/

Figura 3.4 – Atuação do estagiário nos projetos durante o estágio




| # | ATIVIDADE | INÍCIO | TÉRMINO | DIAS |
|---|---|---|---|---|
| A1 | Treinamento em VB.NET | 3/ago | 14/ago | 12 |
| A2 | Acompanhamento de atividades com outros desenvolvedores | 17/ago | 21/ago | 5 |
| A3 | Desenvolvimento demanda A - Sistema 1 | 24/ago | 2/set | 10 |
| A4 | Desenvolvimento de melhorias - API 1 | 1/set | 2/set | 2 |
| A5 | Migração de tecnologia Job 1 | 3/set | 18/set | 16 |
| A6 | Desenvolvimento demanda B - Job 1 | 21/set | 30/set | 10 |
| A7 | Análise demanda C - Sistema 1 | 1/out | 6/out | 6 |
| A8 | Desenvolvimento demanda A - Sistema 1 | 7/out | 9/out | 3 |
| A9 | Desenvolvimento demanda D - API 2 | 12/out | 13/out | 2 |
| A10 | Correções - Job 1 | 14/out | 16/out | 3 |
| A11 | Criação de scripts para tombamento de sistema legado | 19/out | 23/out | 5 |
| A12 | Análise e planejamento demanda C - Sistema 1 | 26/out | 30/out | 5 |
| A13 | Desenvolvimento demanda C - Sistema 1 | 2/nov | 4/dez | 33 |
| A14 | Análise demanda E - API 3 | 7/dez | 9/dez | 3 |
| A15 | Desenvolvimento demanda E - API 3 | 10/dez | 31/dez | 22 |
| A16 | Desenvolvimento demanda F - API 2 | 4/jan | 15/jan | 12 |
| A17 | Desenvolvimento demanda H - Sistema 1 | 18/jan | 1/fev | 15 |

Fonte: Autor.

Durante o período de homologação da atividade [A13] as atividades [A14] e [A15] foram executadas junto a outro membro da equipe. Foram desenvolvidas novas rotas para uma API existente, de forma a prover consulta aos dados utilizados e produzidos pelo Sistema 1. Tal aplicação foi desenvolvida em C# utilizando o *framework* .NET Core 2.1. Para essa aplicação foram construídos testes unitários utilizado o NUnit e o ReportGenerator[4] para analisar a cobertura de código dos testes.

Na sequência, com a atividade [A16], foi implementada uma nova rota na API 2, desenvolvida em Java utilizando o *framework* Spring Boot, de forma a prover a junção e sumarização de valores obtidos em diferentes consultas a uma API externa. Também foram desenvolvidos testes unitários de forma a cobrir todo código novo escrito. Por fim, a atividade [A17] se tratou do apoio no desenvolvimento de melhorias regulatórias no Sistema 1.

[4] https://danielpalme.github.io/ReportGenerator/

# 4 CONCLUSÃO

O presente trabalho buscou apresentar os processos e atividades desempenhadas durante o período de estágio na empresa dti digital, onde foram prestados serviços de desenvolvimento de *softwares* para uma instituição financeira. Foram apresentadas características das tecnologias, plataformas e ferramentas utilizadas, bem como seus casos de uso.

Diversas atividades foram executadas em diferentes contextos, como o desenvolvimento/melhora de um sistema web, de *web services* e de uma rotina automática, além do contato com representantes do cliente, negociação de prazos, acompanhamento de projetos e integração com outros membros da equipe. Com isso diversos conhecimentos teóricos obtidos durante a graduação foram postos em prática, em adição dos novos obtidos, em especial, em relação a sistemas distribuídos, metodologias ágeis, orientação a objetos, arquitetura de softwares e banco de dados.

O processo adotado pela dti para o desenvolvimento de softwares, o dti *flow*, se mostrou eficaz quanto as entregas realizadas pelo time durante o período de estágio. A preocupação com a qualidade do código, os cenários de uso e o entendimento do time em relação a demanda, fizeram com que as entregas atingissem seus objetivos e novas solicitações de projetos partissem do cliente, criando um vínculo de fidelidade com a dti.

Outro ponto relevante é em relação ao acompanhamento da execução dos projetos por parte da dti. Os diversos ritos, como retrospectivas de *sprints*, *checks* de execução e controle do estoque de *sprints*, geram insumos pertinentes e constantes para as lideranças intervirem quando necessário, apoiando os times mais necessitados e garantindo a qualidade dos serviços prestados.

Tais fatores somados a estrutura organizacional horizontal apoiada na cultura ágil, proporcionam para seus colaboradores resolução de problemas de forma descomplicada e um maior nível de responsabilidades em comparação com o modelo tradicional. Dessa forma a evolução, tanto profissional quanto pessoal dos colaboradores acontece de forma acelerada e compatível com as necessidades do mercado.

Por outro lado, o mesmo não acontece junto ao cliente atendido pelo time, visto que se trata de uma empresa tradicional e do ramo bancário, que apresenta processos burocráticos e verticais. Impedimentos e pendências, em alguns casos, demoram a serem resolvidos, gerando perda de produtividade e sobrecarga da equipe. Estes contratempos fazem com que a equipe inicie outras tarefas, mantendo diferentes atividades em andamento e poucas concreti-

zadas. Outro ponto recorrente se dá em relação as demandas, que por englobarem produtos complexos e que abrangem diversas áreas internas, acarretam definições realizadas em paralelo com desenvolvimento das soluções, gerando retrabalho e dilatação de prazos.

Todas essas experiências possibilitaram ao estagiário a vivência de formas de trabalho diferentes, cada uma com seus benefícios e impactos. De um lado a agilidade na comunicação e execução de processos, alta responsabilidade e acompanhamento constante de progresso, do outro a rigidez de processo com pouca adaptabilidade, solução de problemas de forma vertical, escopos fechados e etc.

Com a alta competitividade do mercado, novas tecnologias surgindo diariamente e alta demanda de soluções que agreguem valor ao negócio, a cultura ágil se mostra cada vez mais compatível e adaptável a tais fatores, não sendo uma exclusividade de companhias do ramo tecnológico. Vale ressaltar que as metodologias ágeis não são a solução para todos os tipos de problemas, mas sim um recurso para as necessidades do mercado e uma alternativa aos métodos tradicionais.

# REFERÊNCIAS


ALSHAHWAN, F.; MOESSNER, K. Providing soap web services and restful web services from mobile hosts. In: IEEE. **2010 Fifth International Conference on Internet and Web Applications and Services**. [S.l.], 2010. p. 174–179.

ANDRADE, A. P. de. **O que é uma IDE (Ambiente de Desenvolvimento Integrado)?** 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-uma-ide-ambiente-de-desenvolvimento-integrado>. Acesso em: 01 mar. 2021.

AQUN, Z. et al. Research on tunneling techniques in virtual private networks. In: IEEE. **WCC 2000-ICCT 2000. 2000 International Conference on Communication Technology Proceedings (Cat. No. 00EX420)**. [S.l.], 2000. v. 1, p. 691–697.

ATLASSIAN CORPORATION. **Jira | Software para acompanhamento de itens e projetos**. 2021. Disponível em: <https://www.atlassian.com/br/software/jira>. Acesso em: 08 mar. 2021.

BECK, K. et al. **Manifesto for agile software development**. 2001. Disponível em: <http://agilemanifesto.org>. Acesso em: 18 jan. 2021.

BERNARDO, P. C.; KON, F. A importância dos testes automatizados. **Engenharia de Software Magazine**, v. 1, n. 3, p. 54–57, 2008.

BOSCARIOLI, C. et al. Uma reflexão sobre banco de dados orientados a objetos. In: **Congresso de Tecnologias para Gestão de Dados e Metadados do Cone Sul, Paraná, Brasil**. [S.l.: s.n.], 2006.

CHACON, S.; STRAUB, B. **Pro git**. [S.l.]: Springer Nature, 2014.

CHAMBERLIN, D. D. et al. A history and evaluation of system r. **Commun. ACM**, ACM, New York, NY, USA, v. 24, n. 10, p. 632–646, 1981.

CODD, E. F. A relational model of data for large shared data banks. **Commun. ACM**, ACM, New York, NY, USA, v. 13, n. 6, p. 377–387, 1970.

CRUZ, F. **Scrum e PMBOK unidos no Gerenciamento de Projetos**. [S.l.]: Brasport, 2013.

DAHM, M. Byte code engineering. In: **JIT'99**. [S.l.]: Springer, 1999. p. 267–277.

DRIESSEN, V. **A successful Git branching model**. 2010. Disponível em: <http://nvie.com/posts/a-successful-git-branching-model>. Acesso em: 25 jan. 2021.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011.

ENGWALL, K.; ROE, M. Git and gitlab in library website change management workflows. **Code4Lib Journal**, n. 48, 2020.

EXPRESSVPN. **VPN: o que é?** 2020. Disponível em: <https://www.expressvpn.com/what-is-vpn>. Acesso em: 01 mar. 2021.

FARRELL, J. **Java programming**. [S.l.]: Cengage Learning, 2011.

FERRIS, C.; FARRELL, J. What are web services? **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 46, n. 6, p. 31, jun. 2003.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. [S.l.]: University of California, Irvine Irvine, 2000. v. 7.

Firewall.cx. **Understanding VPN IPSec Tunnel Mode and IPSec Transport Mode - What's the Difference?** 2012. Disponível em: <http://www.firewall.cx/networking-topics/protocols/870-ipsec-modes.html>. Acesso em: 09 mar. 2021.

FISHER, J.; KONING, D.; LUDWIGSEN, A. **Utilizing Atlassian JIRA for large-scale software development management**. [S.l.], 2013.

GITLAB. **All GitLab Features**. 2020. Disponível em: <https://about.gitlab.com/features>. Acesso em: 25 jan. 2021.

GRENNING, J. Planning poker or how to avoid analysis paralysis while release planning. **Hawthorn Woods: Renaissance Software Consulting**, v. 3, p. 22–23, 2002.

LENARDUZZI, V. et al. Are sonarqube rules inducing bugs? In: **2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)**. [S.l.: s.n.], 2020. p. 501–511.

MACKINNON, T.; FREEMAN, S.; CRAIG, P. Endo-testing: unit testing with mock objects. **Extreme programming examined**, p. 287–301, 2000.

MAHNIČ, V.; HOVELJA, T. On using planning poker for estimating user stories. **Journal of Systems and Software**, Elsevier, v. 85, n. 9, p. 2086–2095, 2012.

MAPLE, S.; BINSTOCK, A. **JVM Ecosystem report 2018 – About your Platform and Application**. 2018. Disponível em: <https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/>. Acesso em: 27 fev. 2021.

MICROSOFT. **Introdução ao .NET**. 2020. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/core/introduction>. Acesso em: 15 fev. 2021.

MICROSOFT. **.NET Standard**. 2020. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/standard/net-standard>. Acesso em: 15 fev. 2021.

MICROSOFT. **Microsoft Teams – Group Chat software**. 2021. Disponível em: <https://www.microsoft.com/pt-br/microsoft-teams/group-chat-software/>. Acesso em: 08 mar. 2021.

O'NEIL, E. J. Object/relational mapping 2008: Hibernate and the entity data model (edm). In: **Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2008. (SIGMOD '08), p. 1351–1356.

RED HAT. **O que é IDE?** 2019. Disponível em: <https://www.redhat.com/pt-br/topics/middleware/what-is-ide>. Acesso em: 01 mar. 2021.

ROB, P.; CORONEL, C. **Sistemas de banco de dados: projeto, implementação e administração**. 8. ed. São Paulo: Cengage Learning, 2011.

SABBAGH, R. **Scrum: Gestão ágil para projetos de sucesso**. São Paulo: Editora Casa do Código, 2014.

SANTOS, G. L. dos; FERREIRA, L. Estruturação de equipes em squads em uma empresa do setor de telecomunicações. **CIÊNCIA DINÂMICA**, v. 18, n. 2, p. 66–88, 2020.

SCHÖN, E.-M.; THOMASCHEWSKI, J.; ESCALONA, M. J. Agile requirements engineering: A systematic literature review. **Computer Standards & Interfaces**, Elsevier, v. 49, p. 79–91, 2017.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide**. 2020. Disponível em: <https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>. Acesso em: 18 jan. 2021.

SEMEDO, M. J. Ganhos de produtividade e de sucesso de metodologias ágeis vs metodologias em cascata no desenvolvimento de projectos de software. Universidade Lusófona de Humanidades e Tecnologias, Lisboa, 2012.

SILVA, M. F da; MORENO, A. G. Automaçao em testes ágeis. **Revista de Sistemas e Computação-RSC**, v. 1, n. 2, 2012.

SOARES, M. dos S. Metodologias ágeis extreme programming e scrum para o desenvolvimento de software. **Revista Eletrônica de Sistemas de Informação**, Conselheiro Lafaiete, v. 3, n. 1, 2004.

SONARSOURCE. **SonarQube Documentation**. 2021. Disponível em: <https://docs.sonarqube.org/latest/>. Acesso em: 02 mar. 2021.

SPINELLIS, D. Git. **IEEE software**, IEEE, v. 29, n. 3, p. 100–101, 2012.

TECNICON. **Metodologia Scrum para a gestão de processos ágeis na indústria**. 2019. Disponível em: <https://www.tecnicon.com.br/blog/411-Metodologia_Scrum_para_a_gestao_de_processos_ageis_na_industria>. Acesso em: 09 mar. 2021.

THAI, T.; LAM, H. **. NET framework essentials**. [S.l.]: O'Reilly Media, Inc., 2003.

VERSIONONE. **14th Annual State of Agile Survey**. 2020. Disponível em: <https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>. Acesso em: 08 fev. 2021.

WALLS, C. **Spring Boot in action**. [S.l.]: Manning Publications, 2016.