GUSTAVO MAGNO SANTOS

PROCESSO DE TESTE DE SOFTWARE APLICADO AO PROGRAMA DE REGULARIZAÇÃO AMBIENTAL - PRA

LAVRAS-MG

2019

GUSTAVO MAGNO SANTOS

PROCESSO DE TESTE DE SOFTWARE APLICADO AO PROGRAMA DE REGULARIZAÇÃO AMBIENTAL - PRA

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do curso de Sistemas de Informação, para a obtenção do título de Bacharel.

Prof.ª Dr.ª Renata Teles Moreira

Orientadora

LAVRAS-MG

2019

GUSTAVO MAGNO SANTOS

PROCESSO DE TESTE DE SOFTWARE APLICADO AO PROGRAMA DE REGULARIZAÇÃO AMBIENTAL – PRA

SOFTWARE TESTING PROCESS APPLIED TO THE ENVIRONMENTAL REGULARIZATION PROGRAM - PRA

Relatório técnico apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Sistemas de Informação, para a obtenção do título de Bacharel.

APROVADO em 26 de novembro de 2019.
Dr. André Grützmann - UFLA
Dr. Maurício Ronny de Almeida Souza – UFLA

Profa. Dra. Renata Teles Moreira
Orientadora

LAVRAS – MG

2019

RESUMO

Este trabalho aborda as atividades realizadas e metodologias utilizadas em um projeto de *software*, desenvolvido no Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF). As atividades realizadas tiveram como objetivo, melhorar a qualidade de um *software* que auxilia a administração pública e proprietários de imóveis rurais no processo de regularização ambiental. As atividades consistiram na realização de análises e testes de *software* em meio à uma equipe que utiliza a abordagem ágil, *Scrum*, para gestão e planejamento de projetos de *software*. Este relatório apresenta as ferramentas, tecnologias e metodologias utilizadas durante o desenvolvimento das atividades. Após o término do projeto, constatou que as atividades realizadas tiveram um valor significativo para o aluno adquirir experiência prática dentro de uma organização, assimilando com o conhecimento teórico obtido durante a graduação.

Palavras-chave: Qualidade de *Software*. Teste de *Software*. *Scrum.*

ABSTRACT

This paper discusses the activities and methodologies used in a software project, developed in the Laboratory of Studies and Projects in Forest Management (LEMAF). The activities were aimed at improving the quality of software that assists public administration and rural property owners in the process of environmental regularization. The activities consisted of software analysis and testing among a team that uses the agile approach, Scrum, for management and planning software projects. This report presents the tools, technologies and methodologies used during the development of the activities. Upon completion of the project, it found that the activities performed had significant value for the student to gain practical experience within an organization, assimilating with the theoretical knowledge obtained during undergraduate course.

Keywords: *Software Quality. Software Testing. Scrum.*

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo, demonstrar os conceitos, metodologias e ferramentas envolvidas durante o projeto do Programa de Regularização Ambiental (PRA), aplicado principalmente às atividades que compõem o processo de teste de *software*.

O processo de teste de *software* tem como objetivo principal revelar defeitos que foram introduzidos durante o desenvolvimento de *softwares*, que resultam em falhas e em seu mal funcionamento. Atividades de planejamento, execução e documentação de testes fazem parte desse processo, assim como o gerenciamento de todos os defeitos encontrados. Todas essas atividades foram realizadas nos sistemas que compõem o PRA, com objetivo final de entregar um *software* com a qualidade desejada pelos clientes.

O PRA é um instrumento criado a partir do novo Código Florestal brasileiro (Lei Federal nº 12.651) com o intuito de regularizar a situação ambiental de propriedades e posses rurais em todo território nacional, através de um conjunto de ações a serem adotadas pelos proprietários rurais e fiscalizadas pelos órgãos competentes.

As atividades descritas neste trabalho foram desenvolvidas no Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF), com objetivo de complementar e desenvolver os conhecimentos adquiridos ao longo do curso de graduação em Sistemas de Informação, realizado na Universidade Federal de Lavras, além de obter experiência dentro do mercado de trabalho. As atividades foram realizadas no período de setembro de 2017 até o final do mês de julho de 2018, tendo este trabalho, orientado ao que foi realizado e vivenciado durante este período.

1.1 Estrutura do trabalho

Além deste capítulo de introdução, o trabalho está organizado da seguinte forma: O capítulo 2 contém o referencial teórico, que aborda as definições e conceitos utilizados, apresentando trabalhos relacionados que oferecem suporte para a discussão dos assuntos tratados neste trabalho.

No capítulo 3, Empresa e Sistema, foi realizado a descrição da empresa (LEMAF), a forma de organização das equipes e processos utilizados durante o desenvolvimento do projeto, além da descrição detalhada sobre o *software* desenvolvido.

O capítulo 4, apresenta a descrição das atividades desenvolvidas, relacionadas à função de teste de *software.*

O capítulo 5, apresenta as considerações finais sobre todas as atividades realizadas no decorrer do projeto, bem como sugestões de melhoria.

O capítulo 6, contém as referências bibliográficas utilizadas no trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo aborda as definições e conceitos utilizados, apresentando trabalhos relacionados que oferecem suporte para a discussão dos assuntos tratados neste trabalho.

2.1 Qualidade de *Software*

Sommerville (2011) relata que os problemas com a qualidade de *software* foram inicialmente descobertos na década de 1960 e que, em resposta à insatisfação com os defeitos existentes nos *softwares* produzidos até o final do século XX, passaram a se adotar técnicas formais de gerenciamento da qualidade de *software*. Essas técnicas de gerenciamento de qualidade, em conjunto de novas tecnologias de *software* e melhores testes de *software*, conduziram a melhorias significativas no nível geral de qualidade de *software*.

Ao agregar valor tanto para o fabricante quanto para o usuário de um produto de *software*, um *software* de alta qualidade gera benefícios para a empresa de *software* e para os usuários finais. A empresa ganha valor agregado pelo fato de um *software* de alta qualidade exigir menos manutenção, menos correções de erros e menos suporte ao cliente. Os usuários ganham valor agregado pelo fato da aplicação fornecer a capacidade de agilizar algum processo de negócio. (PRESSMAN, 2011).

Segundo Sommerville (2011), o ideal seria que equipes de gerenciamento de qualidade sejam independentes de qualquer grupo em particular de desenvolvimento, pois caso haja problemas, não corra risco de comprometer a qualidade do produto em nome do cumprimento do cronograma, por exemplo. No entanto, em pequenas empresas isso é praticamente impossível, onde o gerenciamento de qualidade e de desenvolvimento de *software* está inevitavelmente interligado com pessoas com responsabilidades de desenvolvimento e de qualidade.

2.2 Testes de Software

De acordo com Pressman (2011), “teste de software é um elemento de um tópico mais amplo, muitas vezes conhecido como verificação e validação (V&V).” O objetivo da verificação é checar se o *software* atende a seus requisitos funcionais e não funcionais.

Validação, no entanto, é um processo mais geral e tem o objetivo de garantir que o *software* atenda às expectativas do cliente. A validação é essencial porque as especificações de requisitos nem sempre refletem os desejos ou necessidades dos clientes e usuários do sistema. (SOMMERVILLE, 2011).

Os testes de *software* são uma função de controle de qualidade com um objetivo principal - descobrir erros. O papel da *SQA* é garantir que os testes sejam planejados apropriadamente e conduzidos eficientemente de modo que se tenha a maior probabilidade possível de alcançar seu objetivo primário. (PRESSMAN, 2011).

Segundo Delamaro, Maldonado e Jino (2007), o teste de *software* é uma atividade dinâmica, seu intuito é executar o programa ou modelo utilizando algumas entradas em particular e verificar se o comportamento está de acordo com o esperado. Caso a execução apresentar resultados que não foram especificados, dizemos que um erro ou defeito foi identificado. Além disso, os dados obtidos com os testes, podem servir como fonte de informação para a localização e correção de tais defeitos.

Conforme *International Software Testing Qualifications Board* (ISTQB, 2018), um equívoco comum do teste de *software* é que ele consiste apenas em executar testes, ou seja, executar o *software* e verificar os resultados. O teste de *software* é um processo que inclui muitas atividades diferentes, e a execução do teste (incluindo a verificação dos resultados) é apenas uma dessas atividades. O processo de teste também inclui atividades como planejamento de teste, análise, modelagem e implementação dos testes, relatórios de progresso e resultados de testes e avaliação da qualidade de um objeto de teste.

De acordo com Pressman (2011), o papel de um grupo independente de teste é remover problemas inerentes associados ao fato de deixar o próprio desenvolvedor testar a coisa que ele mesmo criou, como por exemplo, demonstrar que o programa é isento de erros e funciona de acordo com os requisitos do cliente. Dessa forma, o teste independente remove o conflito de interesse que poderia estar presente. No entanto, o desenvolvedor e a equipe de teste trabalham juntos durante todo o projeto de *software* para garantir que testes completos sejam realizados.

Enquanto o teste está sendo realizado, o desenvolvedor deve estar disponível para corrigir os erros encontrados.

2.4 Níveis de teste

De acordo com ISTQB (2018), níveis de teste são grupos de atividades de teste que são organizados e gerenciados juntos. Cada nível de teste é uma instância do processo de teste, incluindo suas atividades, que são executadas em relação ao *software* em um determinado nível de desenvolvimento, desde os componentes até os sistemas completos. Os níveis de teste estão relacionados com outras atividades dentro do ciclo de vida de desenvolvimento de *software*. Os níveis de teste abordados neste trabalho estão descritos a seguir.

2.4.1 Teste de componente

Pressman (2011) diz que o teste de componente, também chamado de teste de unidade, focaliza o esforço de verificação na menor unidade de projeto do *software*, o componente ou módulo de *software*. Esse teste enfoca a lógica interna de processamento e as estruturas de dados dentro dos limites de um componente.

Segundo ISTQB (2018), o teste de componente geralmente é realizado isoladamente do resto do sistema, e pode exigir objetos simulados, virtualização de serviços, equipamentos, simuladores e controladores. O teste de componente pode cobrir funcionalidades (como correção de cálculos), características não funcionais (como busca de vazamentos de memória) e propriedades estruturais (como teste de decisão).

Segundo Pressman (2011), o desenvolvedor do *software* é sempre o responsável pelo teste dos componentes do programa, garantindo que cada um deles executa a função ou apresenta o comportamento para o qual foi projetado.

2.4.2 Teste de integração

Conforme ISTQB (2018), o teste de integração se concentra nas interações entre componentes ou sistemas, existindo dois níveis diferentes de teste de integração, o de componentes e o de sistema, ambos descritos a seguir:

a) o teste de integração de componentes foca nas interações e interfaces entre componentes integrados. O teste de integração de componentes é executado após o teste do componente e geralmente é automatizado.
b) O teste de integração do sistema se concentra nas interações e interfaces entre sistemas, pacotes e microsserviços. O teste de integração do sistema também pode abranger interações e interfaces fornecidas por entidades externas (como serviços da *Web*).

Ainda segundo ISTQB (2018), os testes de integração devem focar na comunicação entre os componentes e os sistemas, e não na funcionalidade dos módulos individualmente.

"O teste de integração de componentes geralmente é de responsabilidade dos desenvolvedores, e o teste de integração do sistema geralmente é de responsabilidade dos testadores." (ISTQB, 2018).

2.4.3 Teste de sistema

Conforme Pressman (2011), teste de sistema é uma série de diferentes testes com a finalidade primária de exercitar totalmente o sistema.

Segundo ISTQB (2018), o teste de sistema se concentra no comportamento e capacidade de todo um sistema ou produto, seja funcional ou não, considerando as execuções das tarefas de ponta a ponta do sistema como um todo. O teste de sistema deve usar as técnicas mais apropriadas de acordo com os aspectos do sistema a ser testado.

2.4.4 Teste de validação

De acordo com Sommerville (2011), o teste de validação ou *release*, é o processo de testar um *release* específico de um sistema que se destina para uso fora da equipe de desenvolvimento. Geralmente, o *release* de sistema é para uso dos clientes e usuários. O objetivo do teste de validação é verificar se o sistema atende a seus requisitos e é bom o suficiente para uso externo, não centrando na descoberta de defeitos.

2.4 Tipos de teste

Conforme ISTQB (2018), um tipo de teste é um grupo de atividades de teste destinado a testar características específicas de um sistema de *software*, ou parte de um sistema, com base em objetivos de teste específicos. Tais objetivos podem incluir:

a) avaliar as características de qualidade funcional, como integridade, correção e adequação;
b) avaliar as características de qualidade não funcionais, como confiabilidade, eficiência de desempenho, segurança, compatibilidade e usabilidade;
c) avaliar se a estrutura ou arquitetura do componente ou sistema está correta, completa e especificada;
d) avaliar os efeitos das alterações, como a confirmação da correção dos defeitos (teste de confirmação) e procurar alterações não intencionais no comportamento como resultado de alterações no *software* ou no ambiente (teste de regressão).

2.4.1 Teste funcional

Também chamado de teste comportamental ou teste caixa-preta, o teste funcional de um sistema envolve testes que avaliam as funções que o sistema deve executar. Os requisitos funcionais podem ser descritos como especificações de requisitos, de negócios, épicos, histórias de usuários, casos de uso ou especificações funcionais, podendo ainda não estarem documentados. As funções são “o que” o sistema deve fazer. (ISTQB, 2018).

De acordo com Pressman (2011), o teste funcional tenta encontrar erros em funções incorretas ou faltando, erros de interface, erros em estruturas de dados ou acesso a bases de dados externas, erros de comportamento ou desempenho, e erros de inicialização e término.

“Os testes funcionais devem ser realizados em todos os níveis de teste.” (ISTQB, 2018).

2.4.2 Teste não funcional

Segundo ISTQB (2018), os testes não funcionais de um sistema, avaliam as características de sistemas e de *softwares*, como usabilidade, eficiência de desempenho ou segurança. O teste não funcional é o teste de "quão bem" o sistema se comporta.

Ao contrário das percepções errôneas comuns, o teste não funcional pode e geralmente deve ser realizado em todos os níveis de teste, e feito o mais cedo possível. A descoberta tardia de defeitos não funcionais pode ser extremamente perigosa para o sucesso de um projeto. (ISTQB, 2018).

2.4.3 Teste estrutural

Também chamado de teste caixa-branca, os testes estruturais focalizam a estrutura de controle do programa. São criados casos de teste para assegurar que todas as instruções no programa foram executadas ao menos uma vez e que todas as condições lógicas foram exercitadas. (PRESSMAN, 2011).

Para ISTQB (2018), a eficácia dos testes estruturais pode ser medida através da cobertura estrutural. A cobertura estrutural é a extensão em que algum tipo de elemento estrutural foi testado e é expresso como uma porcentagem do tipo de elemento a ser coberto.

2.4.4 Teste relacionado à mudança

De acordo com ISTQB (2018), testes relacionados à mudança devem ser realizados quando são feitas alterações em um sistema, seja para corrigir um defeito ou por causa de uma funcionalidade nova. O intuito é confirmar se as alterações corrigiram o defeito ou implementaram a funcionalidade corretamente, e não causaram consequências adversas imprevistas.

Ainda segundo ISTQB (2018), especialmente quando se utiliza de desenvolvimento iterativo e incremental, como ágil, novos recursos, alterações nos recursos existentes e recompilação de código resultam em alterações frequentes no código, o que também requer testes relacionados a estas alterações. Devido à natureza evolutiva do sistema, testes como os de confirmação e regressão são muito importantes, além de serem realizados em todos os níveis de teste.

2.4.4.1 Teste de confirmação

Conforme ISTQB (2018), após um defeito ser corrigido, a nova versão do *software* deve ser testada novamente com todos os casos de teste que falharam devido ao defeito encontrado

anteriormente. O software também pode ser testado com novos testes se, por exemplo, para um defeito estiver faltando uma funcionalidade. No mínimo, as etapas para reproduzir as falhas causadas pelo defeito descoberto, devem ser executadas novamente na nova versão do *software*. A finalidade de um teste de confirmação é confirmar se o defeito original foi corrigido com sucesso.

2.4.4.2 Teste de regressão

Segundo Pressman (2011), cada vez que um novo módulo é acrescentado como parte do teste de integração, o *software* muda, assim como após a correção de algum erro, algum aspecto da configuração do *software* também é alterado. Dessa forma, o teste de regressão, que é a reexecução do mesmo subconjunto de testes que já foram executados, ajuda a garantir que as alterações não introduzam comportamento indesejado ou erros adicionais.

2.5 *Framework Scrum*

De acordo com Sommerville (2011), a insatisfação com abordagens pesadas da engenharia de *software*, levou um grande número de desenvolvedores de *software* a proporem na década de 1990, novos métodos ágeis. Métodos ágeis baseiam-se em uma abordagem incremental para especificação, desenvolvimento e entrega de *software*. Eles são mais adequados quando os requisitos de um *software* mudam rapidamente durante o processo de desenvolvimento, e destinam a entregar rapidamente versões funcionais do *software* ao cliente, que pode propor alterações e novos requisitos a serem incluídos posteriormente. O objetivo é reduzir a burocracia do processo, evitando qualquer trabalho de valor duvidoso de longo prazo e qualquer documentação que nunca será utilizada.

Schwaber e Sutherland (2017) definem *Scrum* como um *framework* estrutural usado para gerenciar o trabalho em produtos complexos, dentro do qual se pode empregar vários processos ou técnicas, de modo que você possa continuamente melhorar o produto, o time e o ambiente de trabalho. Além disso o *Scrum* emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

Conforme Schwaber e Sutherland (2017) , "o *framework Scrum* consiste de times *Scrum* associados a papéis, eventos, artefatos e regras. Cada componente dentro do *framework* serve a um propósito específico e é essencial para o uso e sucesso do *Scrum*."

2.5.1 Artefatos e eventos

Segundo Carvalho e Mello (2012), o ponto inicial do *Scrum* é o *Backlog* do Produto. Este artefato é definido por meio de reuniões com todos os clientes e envolvidos no projeto, e são apontadas todas as necessidades do negócio e as funcionalidades a serem desenvolvidas. Dessa forma, o *Backlog* do Produto é uma lista de funcionalidades, ordenadas por prioridade, que provavelmente serão desenvolvidas durante o projeto. Essas funcionalidades são descritas na forma de histórias de usuário (*user stories* ou *US*).

Carvalho e Mello (2012), consideram que a *Sprint* é considerada a principal prática do *Scrum*, e que esse evento é o período de tempo no qual são implementados os itens de trabalho definidos no *Backlog* do Produto pela equipe *Scrum*.

De acordo com Schwaber e Sutherland (2017), as *Sprints* devem ter um tempo de duração (*time-box)* de um mês ou menos, durante o qual um incremento de produto potencialmente liberável é criado. Além disso, as *Sprints* devem ter durações consistentes ao longo de todo o esforço de desenvolvimento, e uma nova *Sprint* inicia imediatamente após a conclusão da *Sprint* anterior.

Ainda segundo Schwaber e Sutherland (2017), a *Sprint*, além de consistir no trabalho de desenvolvimento, também é composta por outros eventos, como a reunião de planejamento da *Sprint* (*Sprint Planning*), reuniões diárias (*Daily Scrum*), reunião de revisão da *Sprint* (*Sprint Review*) e uma reunião de retrospectiva da *Sprint* (*Sprint Retrospective*).

Os eventos que compõem a *Sprint* serão abordados posteriormente na seção 3.1.3, onde serão descritos em conjunto com sua aplicação no processo de desenvolvimento utilizado na empresa.

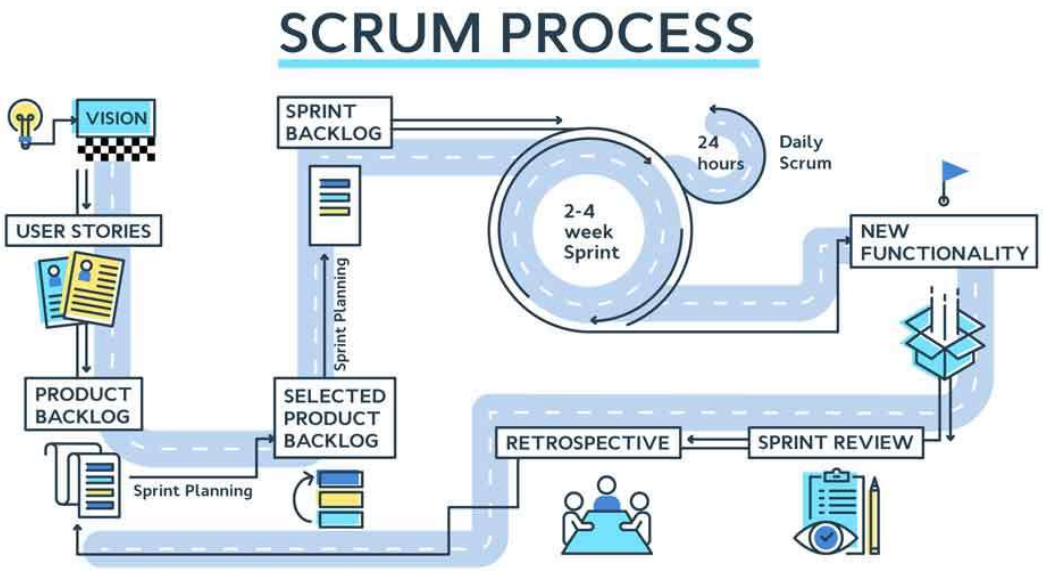
Além do *Backlog* do Produto, outro artefato existente no *Scrum* é o *Backlog* da *Sprint*, que de acordo com Carvalho e Mello (2012), é um subconjunto do *Backlog* do Produto, sendo uma lista composta pelas atividades a serem desenvolvidas durante a *Sprint*. Sua definição acontece durante a reunião de planejamento da *Sprint* (*Sprint Planning*).

De acordo com Carvalho e Mello (2012), o último artefato do *Scrum* é o gráfico *Burndown*. Este artefato trata-se de uma representação gráfica do trabalho restante em

comparação com o trabalho já realizado em uma *Sprint*. Geralmente, coloca-se a quantidade de trabalho no eixo vertical e o tempo no eixo horizontal. Sua utilidade é predizer quando todo o trabalho será completado e também para alarmar o time em caso de atraso.

O ciclo *Scrum* incluindo os artefatos e eventos citados, é representado na Figura 1.

Figura 1 - Ciclo *Scrum*



Fonte: Alona Brailovska - Dreamstime.com

2.5.2 Equipe *Scrum*

Conforme Schwaber e Sutherland (2017), o time *Scrum* é formado por um *Product Owner* (*PO*), o time de desenvolvimento e um *Scrum Master*. Os times *Scrum* escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora do time, e também possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe. O modelo de time no *Scrum* é projetado para aperfeiçoar a flexibilidade, criatividade e produtividade.

Segundo Carvalho e Mello (2012), o *Product Owner* é o membro do time que representa o cliente (interno ou externo). Ele define quais são os requisitos e qual é o grau de importância e prioridade de cada um deles. Este membro necessita conhecer muito bem as regras de

negócios do cliente, de forma que ele possa tirar qualquer dúvida que o time de desenvolvimento possa ter em relação às funcionalidades do produto. O *Product Owner* que é o responsável pelo *Backlog* do Produto.

Conforme Carvalho e Mello (2012), o *Scrum Master* trabalha para que o processo *Scrum* aconteça e para que não existam impedimentos para os membros da equipe realizarem seu trabalho. É seu dever remover os obstáculos apontados na reunião diária, de modo que o time de desenvolvimento se concentre apenas nas questões técnicas.

Schwaber e Sutherland (2017), definem o time de desenvolvimento como o time de profissionais que realizam o trabalho de entregar um incremento potencialmente liberável do produto ao final de cada *Sprint*, ou seja, é ele que executa o trabalho da *Sprint*. Além disso, definem também que o time de desenvolvimento deve ter de 3 a 9 membros, sendo pequeno o suficiente para se manter ágil e grande o suficiente para completar um trabalho significativo dentro da *Sprint*.

2.6 Kanban

De acordo com Brod (2013), o Kanban é definido como um quadro de avisos ou tarefas, e a forma como o utilizamos hoje em abordagens ágeis, vem de um estudo iniciado em 1940 pela Toyota. Na prática, se divide um quadro em três colunas: a fazer (*to do*); em execução (*doing*); feito (*done*). Nessas colunas se cola as etiquetas adesivas com as tarefas e suas anotações relativas. Essas etiquetas representam as *user stories* do *Backlog* da *Sprint* avançando no quadro à medida que as tarefas progridem, e servem também para expor a evolução do projeto durante a *Daily Scrum* (Figura 2).

Figura 2 - Exemplo de um Kanban

Fonte: Brod (2013)

3 EMPRESA E SISTEMA

Este capítulo tem como objetivo apresentar a descrição da empresa (LEMAF), a forma de organização das equipes e processos utilizados durante o desenvolvimento do projeto, além da descrição detalhada sobre o *software* desenvolvido.

3.1 LEMAF / FUNDECC

Fundado em 2004 e inserido no Departamento de Ciências Florestais (DCF) da Universidade Federal de Lavras (UFLA), o Laboratório de Estudos e Projetos em Manejo Florestal (LEMAF) surgiu como uma organização que tem como objetivo, efetuar pesquisa, ensino e extensão, conduzindo diversos projetos em parceria e convênio com órgãos estaduais e federais, bem como a iniciativa privada. Seus projetos englobam as áreas de manejo florestal, geoprocessamento, sensoriamento remoto e tecnologia da informação, sendo referência tecnológica em soluções de gestão ambiental.

O LEMAF responde à Fundação de Desenvolvimento Científico e Cultural (FUNDECC), que tem por finalidade apoiar o desenvolvimento de atividades de ensino, pesquisa e extensão bem como os desenvolvimentos institucionais, científicos e tecnológicos da Universidade Federal de Lavras, mediante assessoramento à elaboração de projetos e administração dos recursos financeiros. A FUNDECC é reconhecida como entidade cuja atuação sirva de base para que as ideias desenvolvidas na Universidade Federal de Lavras possam se transformar em projetos com resultados imediatos, produtivos, levando a Universidade, além da sua função primordial, a produção de conhecimento e inteligência.

3.1.1 Processo de desenvolvimento baseado no *Scrum*

Com o objetivo de entregar algo de valor para os clientes, de forma mais rápida, iterativa e incremental, além de obter um feedback rápido sobre o andamento do projeto, o LEMAF utiliza de abordagem ágil como forma de gerenciamento e desenvolvimento de seus projetos, sendo adotado o *framework Scrum* como referência.

Para atender aos princípios do *Scrum*, a forma organizacional adotada pelo LEMAF tem as definições do *framework* como base, em que seus colaboradores são divididos em times

ágeis, ou *squads* (como é chamado no LEMAF). *Squads* (“esquadrões” em inglês) são grupos responsáveis por todos os aspectos relacionados ao desenvolvimento de um determinado projeto ou produto, são autogerenciáveis e tem autonomia para tomar decisões e definir prioridades, desde que estas estejam alinhadas aos objetivos da empresa. As *squads* no LEMAF são compostas da seguinte forma (times de T.I.):

a) *Product Owner* (*PO*);
b) Desenvolvedores;
c) Analistas de Qualidade de *Software;*
d) *Scrum Master*;
e) Administradores de Banco de Dados.

Cada *squad* é composto por 1 *Product Owner*, 1 *Scrum Master*, 1 administrador de banco de dados e o time de desenvolvimento. Contendo de 3 a 9 integrantes, o time de desenvolvimento é composto pelos desenvolvedores e analistas de qualidade de *software*. Geralmente, um dos integrantes do time de desenvolvimento exerce, também, o papel de *Scrum Master*.

Por convenção, cada *squad* é responsável por um sistema em desenvolvimento, podendo no entanto, serem necessários outros *squads* para novos projetos. Esses projetos podem inclusive, estarem também em um mesmo contrato já firmado com o cliente. Quando isso ocorre, times que atuam em projetos de um mesmo contrato, ou então, times que compartilham de uma mesma tecnologia utilizada, formam um grupo maior, chamado de Tribo.

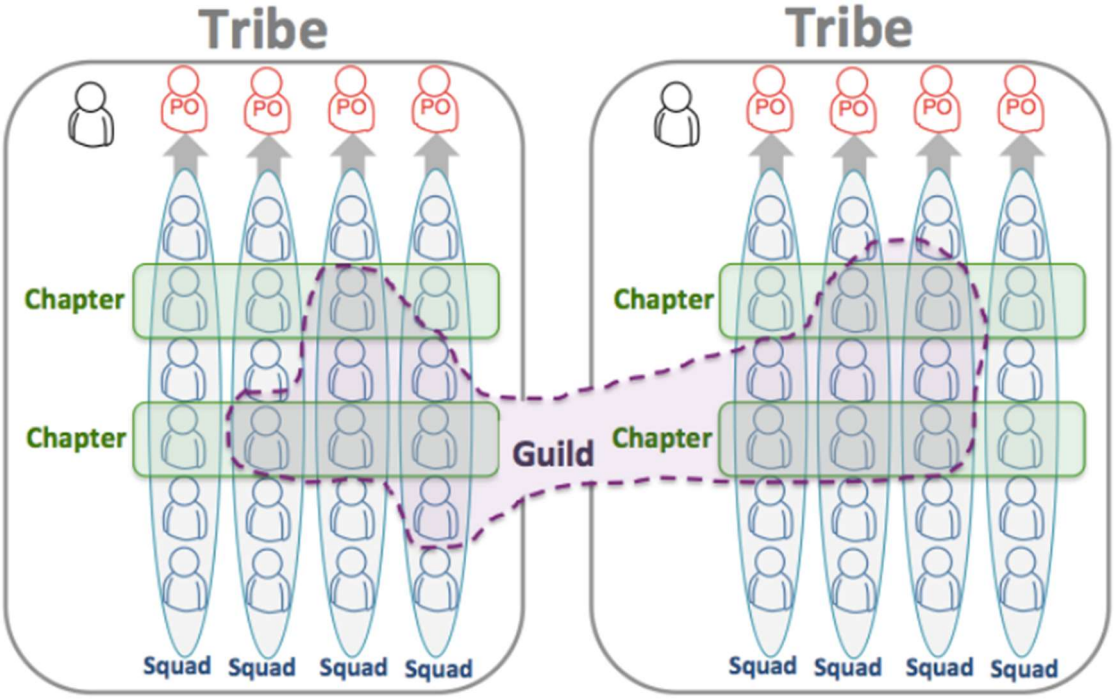
Conforme Kniberg e Ivarsson (2012), cada tribo tem um líder responsável por fornecer o melhor ambiente possível para os *squads* pertencentes a essa tribo. Os *squads* de uma tribo estão todos fisicamente no mesmo escritório, normalmente um ao lado do outro e, a razão de estarem próximos, é promover a colaboração entre os times.

No LEMAF, esse líder é o Gerente de Projetos, que além da responsabilidade citada anteriormente, tem também a função de planejar e coordenar o desenvolvimento dos projetos, suprindo as necessidades e atuando na resolução de eventuais problemas, com foco na meta e prazo dos projetos.

Kniberg e Ivarsson (2012) citam também a existência dos *Chapters* (capítulos) e *Guilds* (guildas). Em que os capítulos são pequenos grupos de pessoas com habilidades semelhantes e que trabalham em uma mesma área de competência, na mesma tribo, com por exemplo, um capítulo de desenvolvedores. De forma semelhante ocorre com as guildas, no entanto, as guildas envolvem toda a organização, podendo neste grupo, além de estarem presentes pessoas de uma mesma área de competência, mas que pertençam a tribos diferentes, participar também outras pessoas que tenham interesse. Como exemplo, podemos citar uma guilda de analistas de qualidade de *software*.

O objetivo desses grupos é discutir assuntos em comum daquela área e compartilhar conhecimento, mantendo todos atualizados e envolvidos no mesmo propósito (Figura 3).

Figura 3 - Organização das tribos



Fonte: Kniberg e Ivarsson (2012)

No LEMAF, estão presentes também uma tribo da área de geoprocessamento e setores de apoio que são vitais para o correto andamento dos projetos, como os setores de infraestrutura, design, suporte e pesquisa e desenvolvimento.

Como parte do processo de desenvolvimento, todos os eventos prescritos pelo *Scrum* são utilizados para criar uma regularidade e minimizar a necessidade de reuniões não planejadas. A seguir, estão listados os eventos Scrum realizados no LEMAF.

a) *Sprint*;
b) *Sprint Planning* (Planejamento da *Sprint*);
c) *Daily Scrum* (Reunião diária);
d) *Sprint Review* (Revisão da *Sprint*);
e) *Sprint Retrospective* (Retrospectiva da *Sprint*).

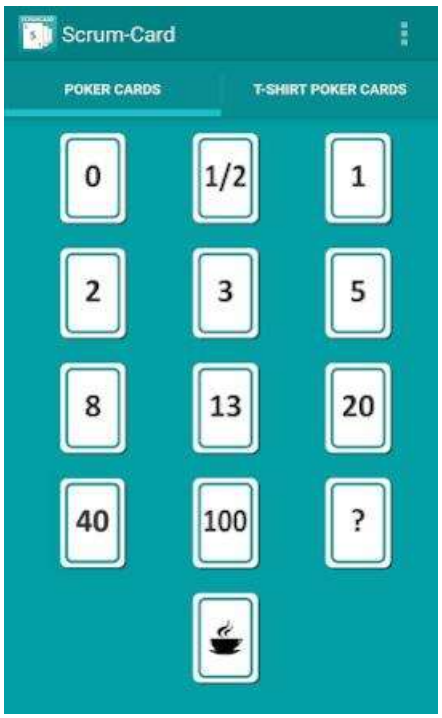
Por convenção, as *Sprints* de todos times no LEMAF tem um *time-box* definido de 2 semanas. No primeiro dia de uma nova *Sprint*, a primeira atividade a ser realizada com todos integrantes do *squad*, é o *Sprint Planning*. Evento com *time-box* de 4 horas, conduzido usualmente pelo *Scrum Master* e *Product Owner*, onde é discutido o trabalho a ser realizado durante toda a *Sprint*.

Inicialmente, o *PO* apresenta as primeiras funcionalidades a serem desenvolvidas, nomeadas de "histórias de usuário" (*user stories* ou *US*). As *US*, que já se encontram priorizadas pelo *PO* em seu *backlog* de produto, são então, uma a uma (em ordem de prioridade) explicadas em detalhes e "quebradas" em atividades menores pelo time. Posteriormente, é feito a estimativa do esforço necessário para cada US. Esse esforço é estimado utilizando um aplicativo *Scrum Poker* (ou *Planning Poker*).

O *Scrum Poker* é uma técnica baseada em gamificação[1], onde existe um baralho com as cartas contendo certos valores, nos quais se referem às possíveis estimativas de esforço que podem ser definidas por cada membro da equipe de desenvolvimento (somente), a cada *US* (Figura 4).

[1] Uso de características e elementos de jogos para melhorar o engajamento e facilitar o aprendizado em situações que não se restringem ao entretenimento.

Figura 4 - Cartas do *Scrum Poker*



Fonte: Captura de tela do aplicativo Scrum-Card

Dessa forma, no momento da estimativa, cada integrante define o esforço que acredita demandar a funcionalidade discutida, escolhe a respectiva carta secretamente e, todos "jogam" na mesa ao mesmo tempo. Se todas as cartas contêm os mesmos valores, esse valor é definido como a estimativa da *US*. Caso as cartas jogadas tenham valores diferentes, as pessoas que jogaram a maior e a menor carta devem explicar para o restante do time, os motivos de suas escolhas. Posteriormente, ocorrem novas rodadas até que o time entre em consenso com o valor de esforço a ser definido para aquela *US*. Esse processo é repetido para todas *US* apresentadas pelo *PO*, até atingir ou ultrapassar o valor de esforço médio que a equipe consegue entregar por *Sprint*, fechando o *backlog* da *Sprint* e encerrando a *Sprint Planning*.

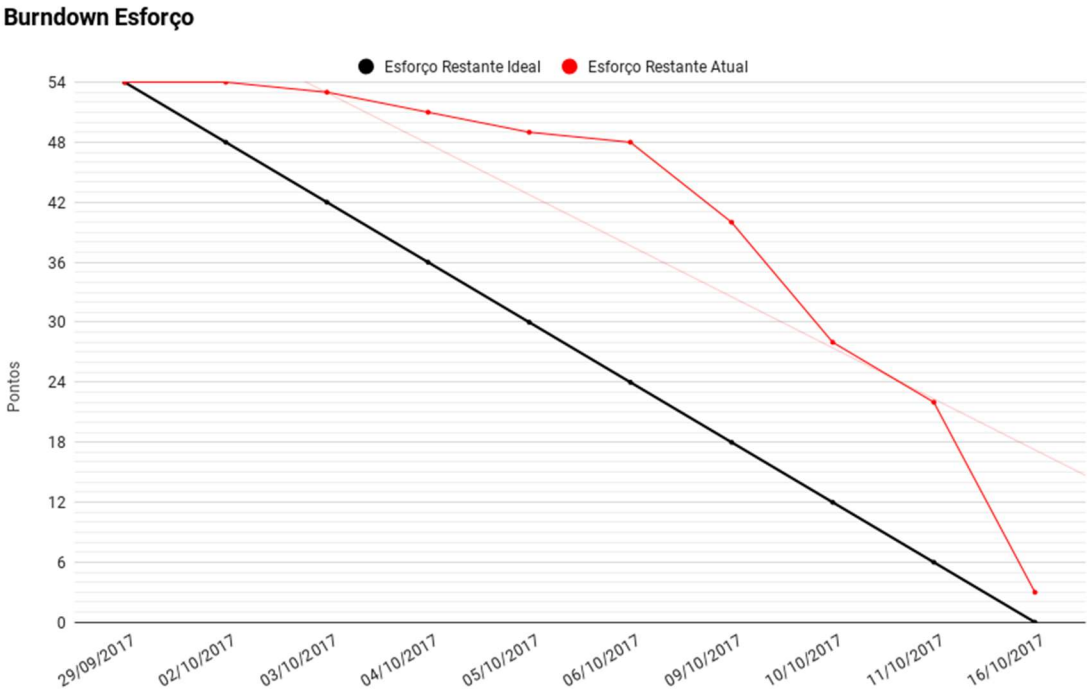
As *US* e suas respectivas atividades que compõem o *backlog* da *Sprint*, ficam registradas no Kanban virtual de cada *Sprint* cadastrada na plataforma de gerenciamento de projetos, Taiga[2]. Essas atividades são dispostas no Kanban na forma de etiquetas.

Cada etiqueta representa uma atividade a ser desenvolvida, e a medida que as atividades vão sendo realizadas, a equipe move as etiquetas para a coluna que representa o estado atual daquela atividade, proporcionando organização, controle e clareza para toda a equipe sobre o andamento da *Sprint.*

A partir dessa técnica, é possível gerar um gráfico, chamado de *Burndown*, com a finalidade de monitorar o progresso da *Sprint*. O *Burndown* indica o trabalho finalizado pelo tempo da *Sprint*, onde o trabalho finalizado é representado pelos pontos de esforço já definidos na *Planning*, e o tempo é representado pelos dias da *Sprint* (Figura 5).

[2] https://taiga.io/

Figura 5 - Gráfico *Burndown*



Fonte: Dados internos do LEMAF

A cada dia de desenvolvimento da *Sprint*, é realizado a *Daily Scrum*, reunião com duração máxima de 15 minutos em que o time de desenvolvimento discute sobre o estado atual das atividades realizadas, onde geralmente cada integrante diz o que fez no dia anterior, o que fará em seguida, e se tem algum obstáculo que impeça atingir a meta da *Sprint*.

Durante o desenvolvimento das atividades, a equipe utiliza a ferramenta de versionamento de código, GitLab[3], que além de armazenar o repositório com os arquivos de desenvolvimento do projeto em um servidor da empresa, inclui também as instruções de configuração dos ambientes de desenvolvimento.

No último dia da *Sprint*, ocorrem a *Sprint Review* e *Sprint Retrospective*, reuniões com *time-box* de 2 horas, cada, onde na *Sprint Review* é inspecionado o que foi realizado e adaptado

[3] https://about.gitlab.com/

o *backlog* do produto (se necessário). Na *Sprint Retrospective* é inspecionado como foi a *Sprint* em relação aos processos utilizados, pessoas e relacionamentos, com objetivo de implementar melhorias para a próxima *Sprint*.

Ao final de cada *Sprint*, um incremento é gerado ao produto, podendo ser feito o *deploy* de uma nova versão do *software* a ser entregue para o cliente.

3.2 PRA

Em vigor desde maio de 2012, o novo Código Florestal brasileiro (Lei Federal nº 12.651) criou como instrumento de controle e monitoramento, um processo de regularização ambiental de propriedades e posses rurais, com base no Cadastro Ambiental Rural (CAR) e nos Programas de Regularização Ambiental (PRA) estaduais.

Conforme Serviço Florestal Brasileiro (SFB, 2016), o primeiro passo para obtenção da regularidade ambiental do imóvel, é a inscrição do imóvel e seus dados no CAR. Os dados exigidos contemplam os dados do proprietário, possuidor rural ou responsável direto pelo imóvel rural, os dados sobre os documentos de comprovação de propriedade e ou posse, informações georreferenciadas do perímetro do imóvel, das áreas de interesse social e das áreas de utilidade pública, com a informação da localização dos remanescentes de vegetação nativa, das Áreas de Preservação Permanente (APP), das Áreas de Uso Restrito (AUR), das áreas consolidadas (áreas ocupadas antes de 22 de julho de 2008 com atividades agrossilvipastoris, ecoturismo ou turismo rural) e das Reservas Legais (RL).

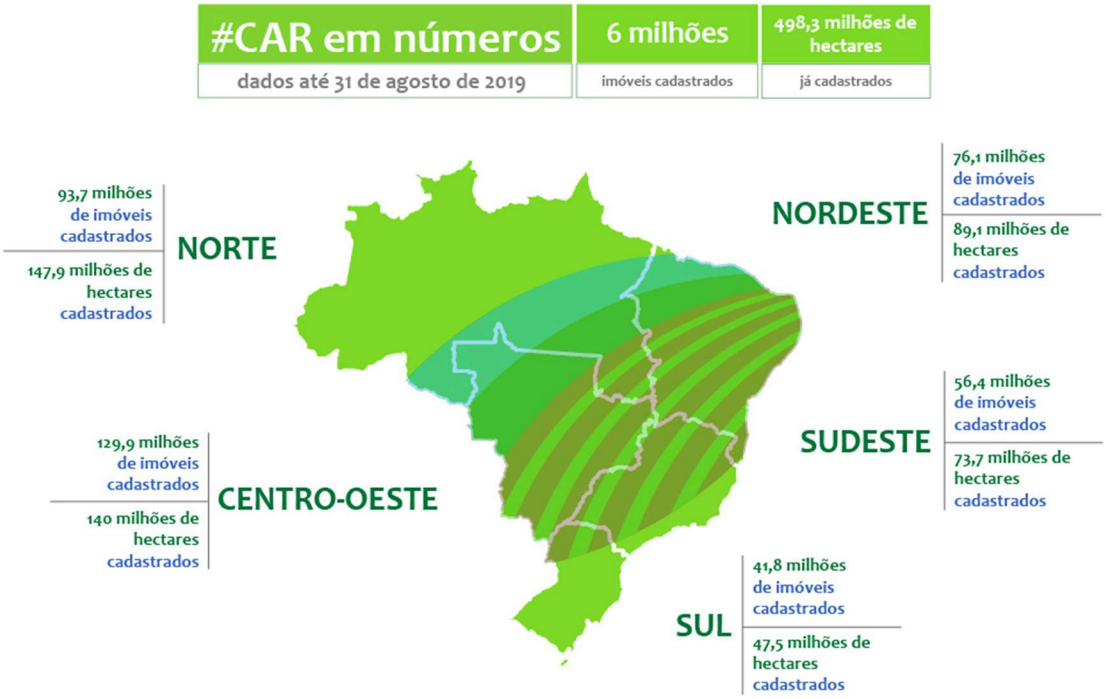
Segundo Lima (2016), o Brasil nunca teve um banco de dados como o CAR, sendo um instrumento fundamental para auxiliar no processo de regularização ambiental de imóveis rurais. Até 31 de maio de 2016, 90,39% da área passível de cadastramento foi inserida no SICAR, que é o sistema eletrônico nacional de cadastramento ambiental rural, o que representa 3,5 milhões de imóveis (Figura 6).

Figura 6 - Módulo de cadastro do CAR

Fonte: Captura de tela do módulo de cadastro do CAR

Em consulta recente aos números do CAR divulgados pelo Serviço Florestal Brasileiro (SFB), até a data de 31 de agosto de 2019, já foram cadastrados 6 milhões de imóveis rurais, totalizando uma área de 498.337.308 hectares inseridos na base de dados do sistema (Figura 7).

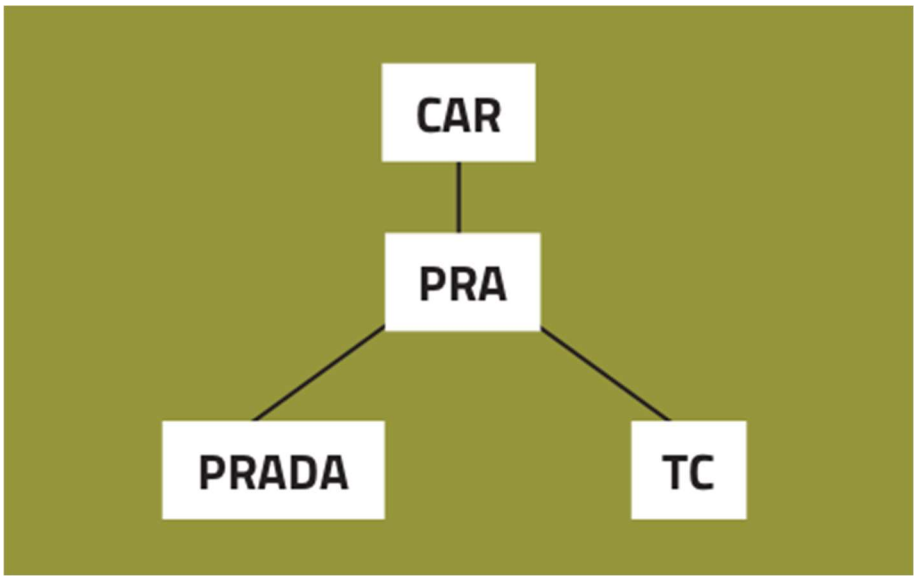
Figura 7 - Números do CAR



Fonte: Serviço Florestal Brasileiro (2019)

Lima e Munhoz (2016) definem o Programa de Regularização Ambiental (PRA) como um conjunto de regras sobre o processo de regularização perante o novo Código Florestal. O PRA tem como base o Cadastro Ambiental Rural (CAR), que irá determinar os passivos de APP e RL a regularizar, além de prever que o produtor deverá propor um Projeto de Recomposição de Áreas Degradadas e Alteradas (PRADA) que, uma vez aprovado pelo órgão ambiental, será a base de um Termo de Compromisso assinado pelo produtor (Figura 8). Os PRAs devem ser claros sobre a regularização das áreas desmatadas antes e depois de 22 de julho de 2008.

Figura 8 - Instrumentos do PRA



Fonte: Lima e Munhoz (2016)

O PRADA é o instrumento de planejamento em que o proprietário ou possuidor de imóvel rural mostra as ações que vai fazer para adequar seu imóvel, que contém o detalhamento das metodologias, cronograma e insumos utilizados para regularização das áreas degradadas da propriedade, quando é necessário.

Segundo o SFB (2017), os Programas de Regularização Ambiental, restringem-se à regularização das Áreas de Preservação Permanente, de Reserva Legal e de uso restrito desmatadas até 22 de julho de 2008, ocupadas por atividades agrossilvipastoris. A regularização poderá ser efetivada através de recuperação, recomposição, regeneração ou compensação das áreas do imóvel rural. A compensação aplica-se exclusivamente às áreas de Reserva Legal consolidadas. Após realizar a inscrição no CAR, os proprietários ou os possuidores de imóveis rurais com passivo ambiental relativo às APP, AUR e RL, poderão solicitar de imediato a adesão ao PRA dos Estados e do Distrito Federal para proceder à regularização ambiental do seu imóvel rural.

A adesão ao PRA traz benefícios aos proprietários, como suspensão de multas ambientais, processos administrativos e criminais, possibilita acesso ao crédito rural, possibilidade de compensação de Reserva Legal, manutenção das atividades agrossilvipastoris

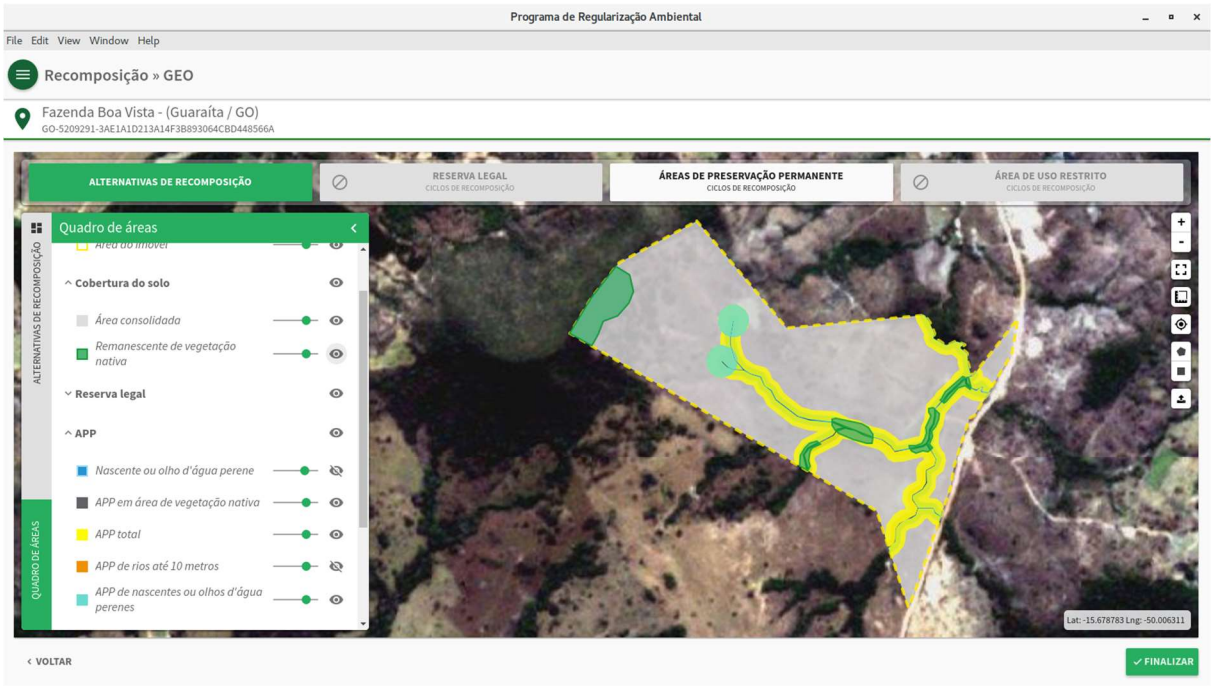
em áreas consolidadas e regularização da situação ambiental do imóvel rural, garantindo segurança jurídica para a atividade produtiva. A Figura 9 representa o fluxo geral do PRA.

Figura 9 - Fluxo geral do PRA

Fonte: Lima e Munhoz (2016)

Finalizado a inscrição do imóvel rural e sua análise no SICAR, o imóvel possuindo a condição que permita sua regularização através do PRA, já é permitido o início da elaboração do projeto de regularização através do módulo de cadastro do PRA (Figura 10).

Figura 10 - Menu “GEO” do PRA

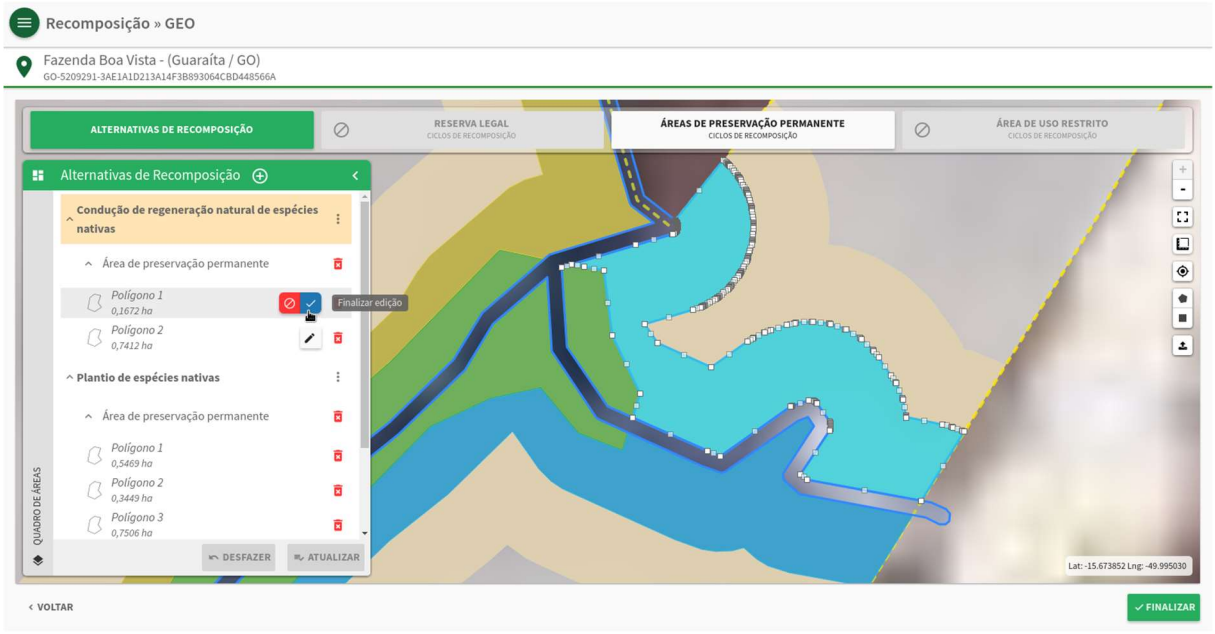


Fonte: Captura de tela do Módulo de Cadastro do PRA (Federal)

Através da ação “Recomposição” presente no módulo de cadastro do PRA, é possível definir o tempo de recomposição de cada tipo de área através do cronograma (dividido em

ciclos), ver todo detalhamento da área do imóvel, conforme cadastro no CAR, bem como elaborar todo projeto de recomposição, definindo as alternativas que serão utilizadas, suas respectivas áreas através da criação de polígonos e as metodologias utilizadas (Figura 11 e Figura 12).

Figura 11 - Demarcação de polígono no PRA



Fonte: Captura de tela do Módulo de Cadastro do PRA (Federal)

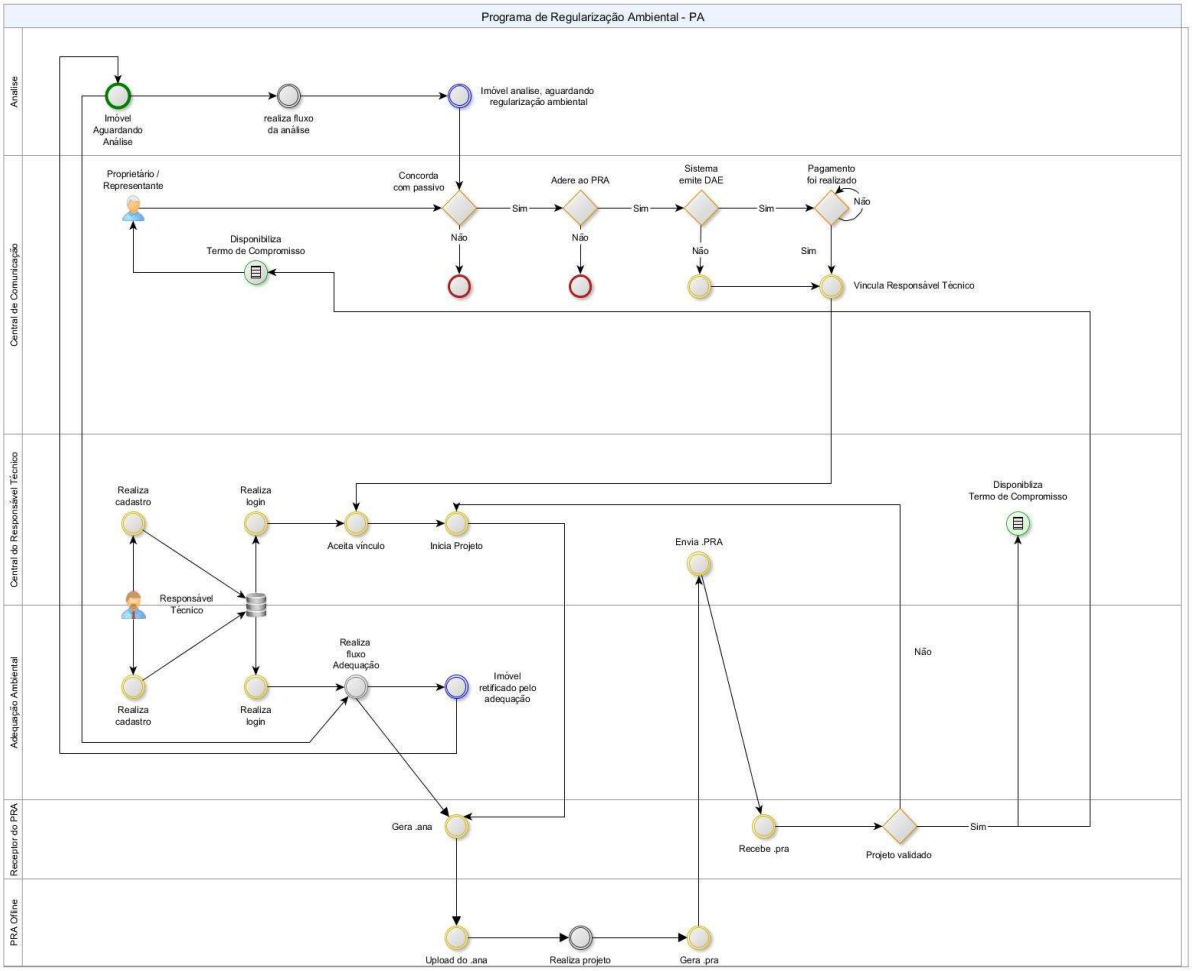
Figura 12 - Definição de metodologias de recomposição no PRA

Fonte: Captura de tela do Módulo de Cadastro do PRA (RO)

Além da ação “Recomposição”, o módulo de cadastro também dispões das ações de cadastro de áreas compensadas e cadastro de sanções administrativas.

Finalizado e validado o projeto de regularização, é dado andamento às etapas seguintes, como geração do PRADA e assinatura do Termo de Compromisso (Figura 13).

Figura 13 - Fluxo do PRA do estado do Pará



Fonte: Dados internos do LEMAF

4 ATIVIDADES DESENVOLVIDAS

Neste capítulo, será descrito a forma como ocorreu o recrutamento e seleção, o treinamento inicial e serão apresentadas as descrições das principais atividades desenvolvidas durante a atuação como analista de qualidade de *software* na organização. Todas atividades foram realizadas nos sistemas que integram o Programa de Regularização Ambiental das unidades federativas do Acre, Pará, Rondônia e da versão do governo federal.

A seleção para atuação como analista de qualidade de *software* no LEMAF se deu através do envio de currículo e posterior realização de testes de raciocínio lógico e entrevista com uma empresa terceirizada de recrutamento e seleção. Posteriormente, foi realizada a entrevista final no LEMAF com uma equipe de 3 pessoas, sendo uma da área de recursos humanos, uma gestora e uma do cargo de atuação, na qual avaliou os conhecimentos técnicos para a função.

O treinamento se iniciou imediatamente após a comunicação de seleção para a vaga e alguns dias antes do início das atividades. Ele iniciou através do estudo de um documento criado pelo conselho internacional de qualificações de teste de software (ISTQB) que é a base de conhecimento para a obtenção das certificações de nível fundamental (CTFL) e nível avançado (CTAL) em testes de *software*. Durante a primeira semana de atuação na organização, o treinamento foi dedicado a aprender sobre as ferramentas utilizadas e adaptação ao processo de gestão e desenvolvimento de *software* utilizado na equipe. A partir da segunda semana foi aprofundado o conhecimento sobre o projeto e sistema desenvolvido, já ocorrendo o início das atividades de teste de *software* (testes de sistema). O início formal das atividades ocorreu durante a terceira semana, com o início de uma nova *Sprint*. O principal contribuidor durante o período de treinamento foi um colega de equipe e função, que era o responsável pelo treinamento. Após sua aprovação perante a equipe, foi permitido o início das atividades, que estão descritas a seguir.

4.1 Planejamento, execução e documentação de testes de *software*

Iniciado o período de desenvolvimento de uma *Sprint*, o analista de qualidade aprofunda os estudos a partir das histórias de usuário que compõem o *Backlog* da *Sprint*, analisando as regras de negócios, requisitos do sistema, protótipos de tela e contexto em que o sistema ou

funcionalidades desenvolvidas na *Sprint* serão utilizadas. Geralmente, nesta etapa ocorre um contato maior com o *PO* para esclarecer possíveis dúvidas e fazer questionamentos sobre as atividades.

Após entender de forma bem clara sobre as funcionalidades e o que deve ser entregue com as atividades em desenvolvimento, o analista de qualidade inicia a produção dos cenários e casos de teste para cada atividade que está sendo realizada pelos desenvolvedores de *software*.

Como é impossível se testar um *software* com perfeição, é de grande importância definir os cenários e casos de teste com inteligência, encontrando todos os casos em que se tem a maior probabilidade de encontrar os defeitos, maximizando esse número e minimizando o custo e tempo gasto nos testes.

Um cenário de teste é um documento que descreve as situações de teste, ou seja, ele ajuda no teste, descrevendo "o que" deve ser testado em cada atividade ou história de usuário desenvolvida (Figura 14).

Figura 14 - Exemplo de um cenário de teste

Sprint 13 - História de usuário #668 - Análise

Descrição do Teste	Tempo Estimado	Critério de saída	Resultado
Verificar se na conclusão do módulo de análise, a opção disponível para indeferir portarias está disponível para seleção como "Indeferir" com o perfil Analista.	Baixo	A opção disponível para indeferir portarias deve ser alterada apenas pela palavra "Indeferir" como resultado do parecer da análise.	OK
Verificar se na conclusão do módulo de análise, a opção disponível para indeferir portarias está disponível para visualização como "Indeferir" com o perfil Gestor.	Baixo	A opção disponível para indeferir portarias deve ser alterada apenas pela palavra "Indeferir" como resultado do parecer da análise.	OK
Conferir se a opção foi alterada no banco de dados analise.tipo_resultado_analise	Baixo	A opção disponível para indeferir portarias deve ser alterada apenas pela palavra "Indeferir" como resultado do parecer da análise.	ERRO

Fonte: Dados internos do LEMAF

Um caso de teste é um documento com um conjunto de condições usadas para se realizar o teste. Nele são definidas as entradas de teste e as saídas esperadas. Diferentemente dos cenários de teste, os casos de teste definem "como" o *software* deve ser testado (Figura 15).

Figura 15 - Exemplo de um caso de teste

Id	Passo	Valor	Ação	Resultado Esperado	Resultados Falhos	Evidências
1	Entrar com a palavra vazia no campo de Busca	""	Clicar em Buscar	Exibir mensagem: "Necessário entrar com a Palavra Chave"	Não exibiu o a mensagem, mas exibiu um erro descohecido	Ver arquivo "CasoTeste 1.JPG"
2	Entrar com número de caracteres acima do permitido (maior que 20)	aaaaaaaaaaaaaaaaaaaa	Clicar em Buscar	Exibir mensagem: "O número de caracteres ultrapassou o limite máximo de 20"		
3	Entrar com uma palavra **válida** no campo Busca, juntamente com uma data válida	célula	Clicar em Buscar	Listar até 2 vídeos com o título, Autor, Veículo, Categoria,	Listou mais de 2 vídeos, não encontrou o resultado de busca e exibiu um erro desconhecido	Ver arquivo "CasoTeste 3.JPG"
4	Entrar com uma palavra **não** válida no Campo Busca, juntamente com uma data **válida**	carro	Clicar em Buscar	Exibir mensagem: "Busca não encontrada"		
5	Entrar com uma palavra **não** válida no Campo Busca, juntamente com uma data **não** válida	carro e 01/24/2012	Clicar em Buscar	Exibir mensagem: "Busca não encontrada"		
6	Entrar com a data inicial igual a data final	01/02/2012 e 01/02/2012	Clicar em Buscar	Listar até 2 vídeos com o título, Autor, Veículo, Categoria,		
7	Entrar com a data Inicial maior que a data final	01/02/2012 e 01/02/2011	Clicar em Buscar	Apresentar Erro: "A data inicial deve ser menor que a data final"	Apresentou erro indefinido	Ver arquivo "CasoTeste 7.JPG"

Fonte: Braga (2015)

Esses casos de teste devem englobar tanto os casos que geram saídas incorretas, quanto os casos que geram saídas corretas pelo sistema.

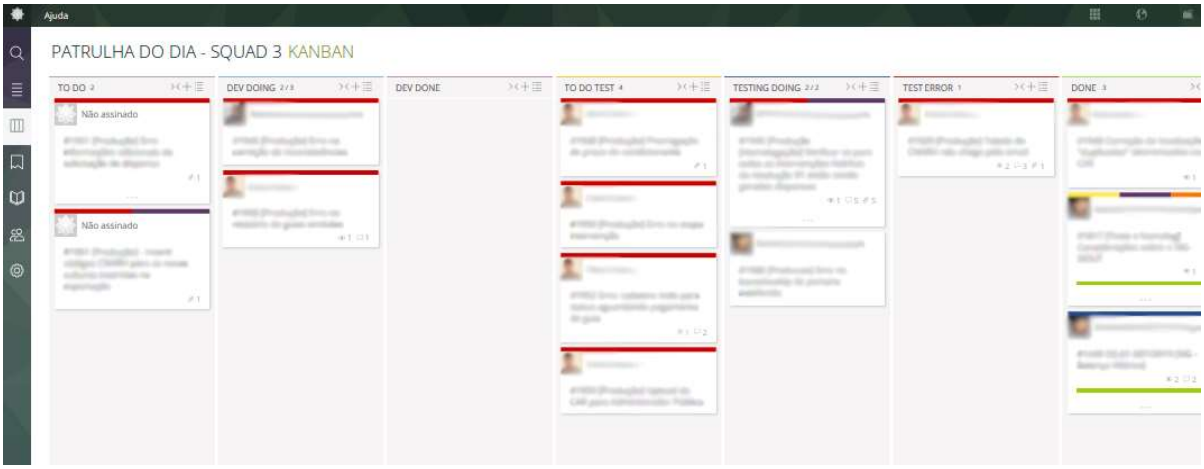
Existem estratégias comumente utilizadas para se definir melhor os casos de teste e guiar sua realização. Citando um simples exemplo, em um determinado teste que tem o objetivo de verificar um formulário com campos de entrada de dados numéricos inteiros positivos, podem ser utilizadas estratégias de dividir o domínio das possíveis entradas em partições menores com algo em comum (partições de equivalência), como partições de números inteiros, decimais, positivos e negativos. A partir dessas partições, pode se utilizar outra estratégia para escolher os valores que serão incluídos nos casos de teste, e que tenham uma probabilidade maior de encontrar defeitos. Uma delas, é a partir da análise dos valores de borda, que escolhe os valores na fronteira entre as partições de equivalência já definidas. No caso do exemplo citado, poderíamos escolher valores entre -1 e 1, incluindo números decimais. Outra estratégia, seria escolher valores aleatórios, como se fosse um palpite baseado em experiências anteriores.

Com os cenários e casos de teste elaborados, o próximo passo é realizar os testes nas atividades desenvolvidas.

Quando um desenvolvedor termina uma atividade (*Dev Done*) da *Sprint*, seus respectivos testes unitários (responsabilidade do desenvolvedor) e ela é aprovada na revisão de código por outro desenvolvedor, a etiqueta desta atividade no Kanban é passada para a coluna

"*To Do Test"*, o que quer dizer que ela está pronta para a etapa de teste. A partir desse momento, o analista de qualidade escolhe dentre essas atividades, aquela que tem a maior prioridade, e transfere a etiqueta dela para a coluna "*Test Doing*" (Figura 16).

Figura 16 - Kanban da *Sprint*



Fonte: Captura de tela do Taiga

Após a realização do teste de uma atividade, caso a atividade passar sem erros, a etiqueta dela é passada para a coluna "*Done*" no Kanban e marcada como concluída. Caso for encontrado um defeito, o analista de qualidade cria um registro do defeito no sistema de rastreamento e controle de *bugs*, vinculando esse registro ao desenvolvedor, incluindo informações como o resumo e descrição do defeito, história de usuário relacionada, qual sistema testado, passos para reproduzir o erro, e anexos com capturas de tela e arquivos que precisam ser utilizados ou podem ser úteis para a correção do defeito.

Em seguida, a etiqueta da atividade deve ser movida para a coluna de "*Test Error*", onde o desenvolvedor pode visualizar de forma mais clara o estado da atividade e realizar as correções, retornando a etiqueta para a coluna de "*Dev Doing*". Após a aprovação na revisão de código (*code review*) por outro desenvolvedor, era feito um teste de confirmação e demais testes necessários. Esse processo é repetido até que a atividade passe nos testes.

Foram realizados testes envolvendo tanto o *front end* como o *back end* das aplicações. No caso dos testes que envolviam a interface das aplicações, além dos testes funcionais, sempre era realizado também o teste não funcional, como o de usabilidade. Nesses tipos de teste, frequentemente era utilizado mais de um navegador, devido a comportamentos diferentes

encontrados durante os testes, que ocasionalmente geravam erros. Durante os testes envolvendo o *back end* das aplicações, era utilizado bastante a ferramenta *Postman*[4], principalmente no módulo "Receptor" do sistema, testando os serviços por meio do envio de diferentes tipos de requisições, analisando seu retorno e a consistência e durabilidade dos dados no banco, através do sistema gerenciador de banco de dados PostgreSQL[5].

Devido a característica do sistema, envolvendo também sistemas de informação geográfica (*GIS*), muitos testes exigiam também a criação de cadastros de imóveis no sistema do CAR e *shapefiles* de imóveis e polígonos para diferentes testes. Esses *shapefiles* eram criados através do *software* QGIS[6], que também era bastante utilizado nos testes que envolviam verificações utilizando *shapefiles*. Esses testes envolviam as localizações, áreas e perímetros dos imóveis, como por exemplo, testes de sobreposição de áreas não permitidas. Posteriormente, todos esses dados eram utilizados para os testes nas aplicações do PRA.

Existia um processo envolvendo o sistema de versionamento de código (GIT), em que todos os testes eram realizados inicialmente no ambiente de desenvolvimento local, através da respectiva *branch* referente às atividades da *US* desenvolvida e, caso o teste passasse, era feito o *merge* com a *branch master* local (atualizada) e testado novamente. No caso das aplicações *back end*, como os serviços, havia mais uma etapa de testes, em que após os testes utilizando o ambiente de desenvolvimento local, a aplicação também era carregada em um servidor de testes que atendia toda equipe e, então, era testada novamente. Neste momento, caso houvessem *scripts* SQL (*evolutions)* com alterações a serem aplicadas na base de dados do servidor de teste, deveria ser indicado sua execução através de uma planilha de controle de *evolutions*. Após aprovado nas etapas de teste, a *branch* remota da *US* era sincronizada com a *branch master* do projeto no GitLab.

No momento de realizar o *deploy* de uma nova versão (*release*) do sistema no servidor de homologação, os analistas de qualidade também faziam um teste de validação antes do cliente, para verificar se a aplicação estava funcionando corretamente para uso externo.

---

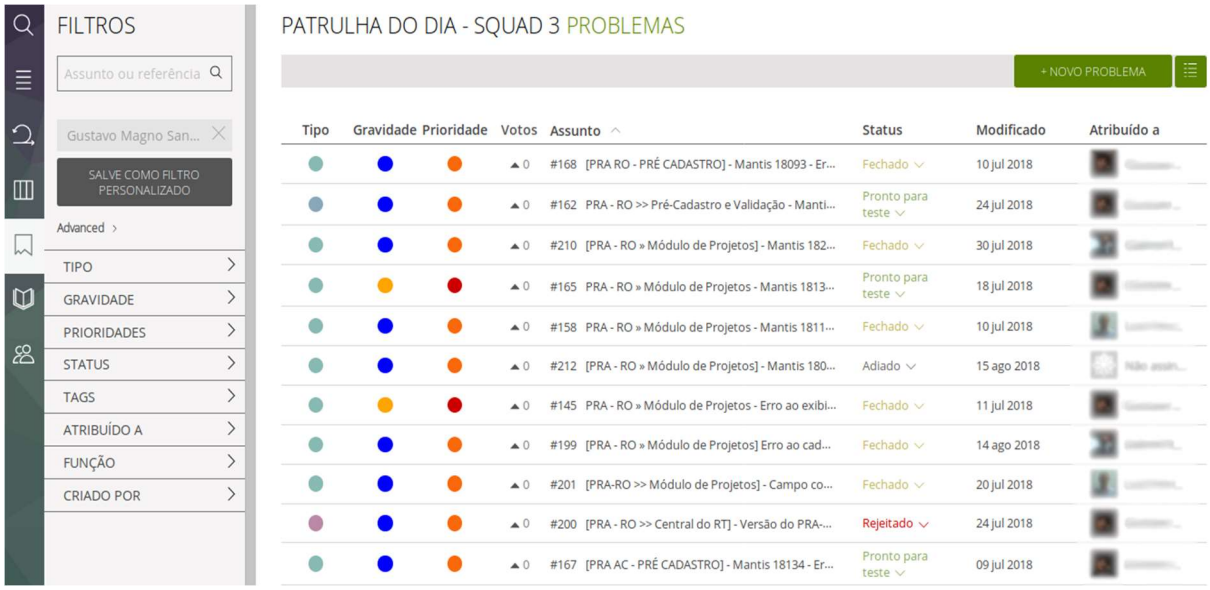
[4] https://www.getpostman.com/
[5] https://www.postgresql.org/
[6] https://www.qgis.org/pt_BR/site/

Dentre todas atividades de teste, foram realizados testes funcionais, não funcionais, testes relacionados à mudança, principalmente nos níveis de testes de integração, sistema e validação.

4.2 Gerenciamento dos defeitos encontrados e propostas de melhoria

A partir da realização dos testes, todos os defeitos encontrados ou propostas de melhoria sugeridas pelos analistas de qualidade, são registrados no sistema de rastreamento e controle de *bugs*, que pode ser o Mantis[7] ou então o próprio Taiga, que tem uma ferramenta para isso (Figura 17).

Figura 17 - Defeitos registrados no Taiga



Fonte: Captura de tela do Taiga

Os problemas registrados no Taiga podem ser vinculados a sua respectiva *US*, a outros problemas e ao desenvolvedor que realizou a atividade, definindo também filtros, como tipo (erro ou melhoria), gravidade, prioridade, status e sistema. A descrição do problema e todas informações relacionadas que foram criadas no momento do teste, estão também disponíveis para consulta.

[7] https://www.mantisbt.org/

É papel do analista de qualidade fazer o monitoramento e gerenciamento de todos registros, finalizando os registros já corrigidos, alterando e replicando o que for necessário, bem como levar os registros com maiores prioridades para discussão com o *PO* e possível apresentação na próxima *Sprint Planning*.

4.3 Elaboração e revisão de manuais de usuário e documentos relativos ao *software*

Quando existe alguma atividade na *Sprint* que necessita de documentação para ser apresentada e entregue ao cliente, é papel do analista de qualidade elaborar esses documentos.

Algumas dessas documentações, são os manuais de usuário do módulo de cadastro do sistema. No início do trabalho no LEMAF, somente o manual do PRA do governo federal havia começado o desenvolvimento, portanto, foram realizadas as atividades de revisão e finalização do manual, abrangendo as funcionalidades de todo o sistema. Posteriormente foi elaborado também os manuais específicos para o módulo de cadastro do PRA dos estados do Acre, Pará e Rondônia. Foi realizado também o manual das aplicações Adequação Ambiental e Central do Responsável Técnico, que compõem o sistema do PRA do estado do Pará.

Depois que os manuais foram elaborados, foi implementado o processo de atualização constante de todos documentos, juntamente com o desenvolvimento de atividades que alterassem ou implementassem funcionalidades novas nos sistemas. Nesse caso, uma das tarefas que compunham as *US,* era a atualização da documentação, quando necessário.

Por fim, outras atividades realizadas foram, a elaboração de documentos de casos de uso, elaboração de *changelogs* no momento de *deploy* de novas versões do sistema, revisões ortográficas e posteriores correções em todos os sistemas do PRA.

5 CONSIDERAÇÕES FINAIS

Após a experiência vivenciada e todo conhecimento adquirido durante as atividades realizadas como analista de qualidade de *software* no LEMAF, bem com durante as aulas do curso de Sistemas de Informação, foi possível identificar que o processo de desenvolvimento utilizado no LEMAF, baseado no *Scrum*, funciona de forma eficiente e conforme planejado pela organização. Porém, foi identificado também, situações que podem ser discutidas e trabalhadas para melhorar ainda mais o ambiente e o processo de desenvolvimento da organização, como por exemplo, incentivar o estudo e aplicação de testes automatizados por parte dos analistas de qualidade, que em geral, só realizam testes manuais. Esses testes automatizados poderiam reduzir muito o custo e tempo de execução de determinados tipos de teste, como os testes de regressão. Outro ponto observado que pode ser melhorado, se refere a um melhor planejamento e organização das *squads*, para que haja equilíbrio de trabalho entre as diferentes funções exercidas e o fluxo de desenvolvimento ocorra de forma mais fluida, evitando sobrecarga excessiva para alguns membros da equipe de desenvolvimento ao final da *Sprint*. A última sugestão, seria ampliar a utilização do *Docker*[8] para a configuração dos ambientes de desenvolvimento de todos projetos do LEMAF em *containers*, permitindo a portabilidade do ambiente e reduzindo o tempo que um colaborador leva para configurar os ambientes de desenvolvimento necessários durante o trabalho.

Uma das contribuições das atividades realizadas pelo aluno foi em relação ao conhecimento e correto uso das ferramentas durante o dia a dia de trabalho. No período foram utilizadas várias ferramentas amplamente requisitadas em empresas da área de tecnologia da informação. Estas ferramentas são responsáveis por otimizar determinadas atividades dentro da organização, seja em relação a comunicação entre as equipes, gestão dos projetos, gerenciamento de documentos, controle e versionamento de código e realização das atividades de teste de *software*. As principais ferramentas utilizadas foram: Taiga, GitLab, Visual Studio Code, Rocket.Chat, Alfresco, Mantis Bug Tracker, PostgreSQL, Postman, QGIS, Asciidoctor e Docker.

[8] https://www.docker.com/

Outra contribuição foi em relação ao aprendizado prático e teórico adquirido sobre teste de *software*, além da oportunidade de realizar cursos específicos sobre a área em outras instituições de ensino. Esse aprendizado foi importante para complementar o conhecimento teórico e introdutório visto durante algumas disciplinas da graduação.

Uma das contribuições mais relevantes do estágio foi em relação a experiência prática geral adquirida com a atuação no mercado de trabalho. Na maior parte do tempo, durante a graduação, o aluno tem pouco ou nenhum contato com o mercado de trabalho, ficando um pouco perdido sobre o que fazer e como agir após a conclusão do curso e ingresso no mercado. Após o período de trabalho realizado, o aluno sai muito mais preparado nesse aspecto, tendo conhecimento sobre como funciona um processo de recrutamento e seleção, sobre como funciona o dia a dia de trabalho em uma organização que tem orçamentos, prazos e compromissos a cumprir, sobre o relacionamento profissional com colegas e comportamento adequado no ambiente de trabalho, bem como a experiência de atuar em um projeto de porte nacional.

Por fim, após o trabalho realizado no LEMAF, foi percebido a melhora da qualidade do *software* produzido e a importância do papel do analista de qualidade de *software* no processo de desenvolvimento de um *software*. Além disso, a experiência obtida foi muito importante para o aluno assimilar alguns conteúdos teóricos aprendidos em sala de aula, conteúdos que também foram úteis para realização das atividades no LEMAF. As principais disciplinas do curso de Sistemas de Informação que auxiliaram durante as atividades realizadas foram:

a) Engenharia de *Software* I:

A disciplina de engenharia de *software* possibilitou entender sobre como é o processo de construção de um *software*, desde sua especificação de requisitos junto ao cliente até a entrega, e os diferentes modelos de ciclo de vida de *software*. A disciplina também permitiu aprender e realizar a modelagem de alguns sistemas, a entender sobre as boas práticas de programação, a importância de se preocupar com a qualidade e além disso, proporcionou o primeiro contato sobre o que são os testes de s*oftware* e suas técnicas utilizadas.

b) Gerência de Projetos de *Software*:

A disciplina Gerência de Projetos de *Software* permitiu a compreensão de como um projeto de *software* é gerenciado, sobre os ciclos de vida dos projetos e as áreas do conhecimento em gerência de projetos: Integração, Escopo, Cronograma, Custo, Qualidade, Recursos, Comunicações, Riscos, Aquisições e Partes Interessadas (*stakeholders*). Durante a disciplina foi demonstrado também sobre as diferentes abordagens adotadas na gerência de projetos, entre elas, a preditiva, ágil e híbrida. Em relação à abordagem ágil, foi visto principalmente sobre o *Scrum*, que foi muito útil durante o período de trabalho no LEMAF.

c) Banco de Dados I e Sistemas Gerenciadores de Banco de Dados:

Os conhecimentos adquiridos nas disciplinas de banco de dados ajudaram durante a realização de testes que envolviam a consistência e durabilidade de dados nas bases de dados, assim como na realização de consultas, operações e criação de *scripts* SQL de acordo com a necessidade das atividades. Durante as disciplinas, foi visto principalmente sobre a linguagem SQL e as arquiteturas dos Sistemas Gerenciadores de Banco de Dados (SGBD), no qual foi visto também sobre o PostgreSQL e sua possibilidade de integração com os Sistemas de Informações Geográficas (SIG), ferramentas que eram muito utilizados durante as atividades realizadas.

d) Qualidade de *Software:*

O conhecimento obtido através da disciplina de Qualidade de *Software* possibilitou entender sobre os fundamentos, gestão, normas e modelos de qualidade de *software*, bem como as métricas de avaliação de qualidade.

e) Interação Humano-Computador:

A disciplina de Interação Humano-Computador permitiu compreender melhor sobre como as interfaces afetam os usuários, sobre o design, prototipação e construção de interfaces e sua avaliação baseadas em heurísticas. Foi visto e exercitado também diferentes técnicas utilizadas em testes de usabilidade e sua importância no projeto de interfaces de *software*.

6 REFERÊNCIAS BIBLIOGRÁFICAS

BRAGA, J. C. **Objetos de aprendizagem, volume 2:** metodologia de desenvolvimento. Santo André, SP : Editora da UFABC, 2015.

BROD, C**. Scrum Guia Prático para Projetos Ágeis**. 1. ed. São Paulo, SP : Novatec, 2013.

CARVALHO, B. V. de; MELLO, C. H. P. **Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica**. Diálogos & Ciência: Revista Eletrônica da Faculdade de Tecnologia e Ciências de Feira de Santana, v. 19, n. 3, p. 557–573, dez. 2012. Disponível em: <http://www.scielo.br/pdf/gp/v19n3/09.pdf>.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. Rio de Janeiro, RJ : Elsevier, 2007.

INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD. **Certified Tester Foundation Level Syllabus**. [S.l.], 2018. Disponível em: <https://www.bstqb.org.br/uploads/syllabus/syllabus_ctfl_2018br.pdf>. Acesso em: 11 nov. 2019.

KNIBERG, H.; IVARSSON, A. **Scaling Agile Spotify with Tribes, Squads, Chapters & Guilds**. [S.l.], 2012. Disponível em: <http://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf>. Acesso em: 20 out. 2019.

LIMA, R. C. A.; MUNHOZ, L.; KFOUR, A. (Colab.). **Programas de Regularização Ambiental (PRAs) :** um guia para orientar e impulsionar o processo de regulamentação dos PRAs nos estados brasileiros. São Paulo, SP : Agroicone, 2016.

PRESSMAN, R. S. **Engenharia de Software:** uma abordagem profissional. 7. ed. Porto Alegre, RS : AMGH, 2011.

SCHWABER K.; SUTHERLAND J. **Um guia definitivo para o Scrum:** As regras do Jogo. [S.l.], 2017. Disponível em: <https://getscrum.com/wp-content/uploads/2017/11/NovoGuiaScrumEmPortuguesDownload2017.pdf>. Acesso em: 23 out. 2019.

SERVIÇO FLORESTAL BRASILEIRO. **Cadastro Ambiental Rural**. [S.l.], 2016. Disponível em: <http://www.car.gov.br/#/sobre>. Acesso em: 08 nov. 2019.

SERVIÇO FLORESTAL BRASILEIRO. **Etapas do CAR e Regularização Ambiental**. [S.l.], 2017. Disponível em: <http://www.florestal.gov.br/regularizacao-ambiental>. Acesso em: 08 nov. 2019.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo, SP : Pearson, 2011.