UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ
CAMPUS LUIZ MENEGHEL - CENTRO DE CIÊNCIAS TECNOLÓGICAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO HENRIQUE MORALES DE MENDONÇA

UMA PROPOSTA DE UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE DISTRIBUÍDO FOCADO PARA SISTEMAS MÓVEIS

BANDEIRANTES–PR

2017

MENDONÇA, G. M.. **Uma Proposta de um Processo de Desenvolvimento de Software Distribuído Focado para Sistemas Móveis**. 39 p. Trabalho de Conclusão de Curso – Projeto (Bacharelado em Ciência da Computação) – Universidade Estadual do Norte do Paraná, Bandeirantes–PR, 2017.

RESUMO

O desenvolvimento de software para dispositivos móveis vem ganhando espaço e tem crescido de forma significativa, proporcionando um novo impulso à área de desenvolvimento de software, devido à alta demanda e às tecnologias novas que surgem com o passar do tempo. As empresas de desenvolvimento estão cada vez mais aumentando sua produtividade utilizando times de desenvolvimento geograficamente distribuídos. Com o objetivo de facilitar o desenvolvimento e a integração entre o desenvolvedor e o cliente, este trabalho apresenta um processo de desenvolvimento de aplicativos móveis com o intuito de auxiliar as equipes distribuídas. Também é mostrado o desenvolvimento um aplicativo para dispositivos móveis para a empresa COPEL, utilizando o processo proposto e por fim a disponibilização do aplicativo para a empresa. Este trabalho apresenta quais foram as percepções da aplicação do processo, mostrando quais pontos fortes e fracos do mesmo.

Palavras-chave: Processo de Software, Desenvolvimento Distribuído de Software, Desenvolvimento de Aplicações Móveis, Arquitetura de Software.

MENDONÇA, G. M.. **Distributed Software Development Process**. 39 p. Final Project – Draft Version (Bachelor of Science in Computer Science) – State University Northern of Parana , Bandeirantes–PR, 2017.

ABSTRACT

The software development for mobile devices has been gaining momentum and has grown in a significant way, giving a boost to the software development area, because of the high demand and the new technologies that appears over time. Development companies are increasingly boosting their productivity by using geographically distributed development teams. In order to facilitate the development and integration between the client and the developer, this work presents a development process for mobile applications with the purpose of assisting the distributed teams. Also, this work presents a developement of a mobile application for the COPEL company, using the proposed process and lastly making avaiable the application for the company. This works presents which were the perceptions of aplying the proccess, showing the strongest and weakest points of the process.

Keywords: Software Development Process, Distributed Software Development, Mobile Application Development, Software Architecture.

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização

De acordo com Herbsleb e Moitra [2001] software tem se tornando um componente vital de qualquer tipo de negócio e as últimas décadas tem vivenciado a globalização para negócios, em particular, negócios utilizando softwares intensivos e de alta tecnologia. A competição pelo mercado global faz com que as empresas passem os limites nacionais e comecem a pensar em um escopo global.

Pautado nessa ideia, com o intuito de melhorar a forma com que desenvolvimento de software distribuídos são feitos, a motivação é em construir um processo de desenvolvimento de software utilizando estudos de modelos de processos já existentes e de acordo com as características peculiares do nosso escopo, como gerenciar equipes fisicamente distribuídas de forma simples e rápida, propor um processo que atenda essas necessidades. Vale ressaltar que nenhum processo é o ideal para o desenvolvimento, mas o mais vantajoso de ser usado no contexto do projeto.

Portanto, norteados com pesquisas em desenvolvimento global de software, processos de software e arquitetura de software, é proposto o desenvolvimento de um processo de software para aplicações de dispositivos móveis utilizando um trabalho em parceria com uma empresa a qual é aplicado o processo proposto, testado e coletado os resultados.

1.2 Formulação e Escopo do Problema

Neste trabalho é realizado um processo de desenvolvimento em parceria com a empresa COPEL. Diante disso, surge alguns problemas e riscos no desenvolvimento. É necessário que os requisitos sejam bem definidos, o sistema validado pela parte do cliente e a integração com o sistema já existente localizado fisicamente na empresa. Deste modo, a utilização de um modelo de processo já existente não é o ideal, já que este é um trabalho pequeno com equipes fisicamente distribuídas e que não se encaixam nas descrições de modelos de processos já propostos.

1.3 Justificativa

Devido à magnitude deste trabalho ser relativamente pequeno comparado com grandes empresas de desenvolvimento de software, foi optado por definir um processo de desenvolvimento de aplicativos móveis que objetiva melhorar a comunicação entre as partes de desenvolvimento e cliente, o entendimento dos requisitos necessitados pela parte

do cliente e agilizar a etapa de desenvolvimento.

Portanto a melhor alternativa foi criar um processo de desenvolvimento de aplicativos móveis que se encaixe melhor ao escopo do trabalho realizado. Para o processo proposto, é utilizado características de modelos ágeis, tais como os *sprints* do modelo de processo *Scrum* e histórias de usuário, utilizado no modelo de processos *XP*. Também é utilizado conceitos do modelo de processo prescritivo Prototipação Evolucionária, para facilitar na validação do sistema pela parte do cliente.

1.4 Objetivos

1.4.1 Objetivo geral

O objetivo deste trabalho é desenvolver um processo de desenvolvimento de software para desenvolvimento distribuído de dispositivos móveis a fim de agilizar futuros desenvolvimentos do mesmo escopo.

1.4.2 Objetivos específicos

Este trabalho tem como objetivos específicos:

- Criar um processo de desenvolvimento de software distribuído para dispositivos móveis;
- Descrever este processo;
- Desenvolver um aplicativo para a empresa COPEL utilizando o processo proposto.
- Apresentar os resultados obtidos após a aplicação do processo.

2 FUNDAMENTAÇÃO

Este capítulo apresenta alguns temas significativos à elaboração deste trabalho. Primeiramente são apresentados os conceitos e características de Processos de Software, posteriormente serão abordados os seguintes assuntos: Desenvolvimento Global de Software, Desenvolvimento de Aplicativos Móveis e Arquitetura de Software.

2.1 Processos de Software

A engenharia de software é uma disciplina pertencente à engenharia que se ocupa de todos os aspectos da produção de um software, desde o início da especificação do sistema, até a manutenção deste sistema depois do mesmo já estar em operação [SOMMERVILLE et al., 2003].

Para Pressman e Maxim [2016], processos de software é um roteiro de passos previsíveis, o qual é usado quando se trabalha na elaboração de um produto ou sistema. É utilizado também para se ter um resultado de alta qualidade e dentro do prazo estabelecido. Portanto, um processo de software é constituído por um conjunto de atividades:

- Interdependentes;
- Com responsáveis;
- Com entradas e saídas definidas.

Então, processo é um conjunto de regras que definem como um projeto será executado. Um projeto é algo que ocorre em um tempo determinado, e na sua execução concreta, visa a criação de um produto específico. Processo não deve ser confundido com um modelo de processo, que é um conjunto de regras mais abstratas que especificam a forma geral dos processos [WAZLAWICK, 2013].

Segundo Wazlawick [2013], há várias vantagens em definir o desenvolvimento como um processo, entre elas:

- **O tempo de treinamento pode ser reduzido**: é mais fácil encaixar novos membros na equipe quando se tem processos bem definidos e documentados;
- **Produtos podem ser mais uniformizados**: o processo em si não garante uniformidade nos produtos, porém uma equipe que tem um processo definido tende a ser mais previsível do que a mesma equipe sem processo;

- **Possibilidade de capitalizar experiências**: pode-se imaginar que um processo bem definido inibe a criatividade do desenvolvedor, porém isso não é verdade. Um processo bem gerenciado deve prover mecanismos para sua melhoria, portanto se um desenvolvedor descobrir uma maneira melhor de fazer algo do que proposto pelo processo, deve ser possível incorporar essas alterações ao processo.

Normalmente nos modelos de processo há grandes divisões dos processos, que são chamados de fases. Uma fase é um período de tempo em que atividades com objetivos bem específicos são realizadas.

Em alguns processos, como no Modelo Cascata, as fases são sequenciais, ou seja, depois de um ponto específico ele passa de uma fase para outra, como de requisitos para análise, depois para a programação, testes, implantação, etc.

Outros processos tem fases cíclicas. Neste modelo, o desenvolvimento passa de uma fase para outra, repetidamente, até que o projeto seja finalizado. Exemplos desse modelo são o Modelo Espiral e o Modelo Prototipação Evolucionária, além dos modelos ágeis [WAZLAWICK, 2013].

Na maioria dos processos, sua organização é feita em torno de tarefas, as vezes chamada de atividades. Cada atividade tem um objetivo principal que visa criar uma mudança visível nos artefatos durante a execução de um projeto [WAZLAWICK, 2013].

Cada atividade deve ter um artefato de entrada e saída. A atividade toma um artefato de entrada, modifica ou cria um novo artefato para gerar sua saída.

Artefatos podem variar, sendo desde diagramas, documentos de texto, até programas, contratos e projetos. Alguns modelos determinam que cada artefato deve ter um dono e que só o mesmo pode alterar, mas outros modelos já dizem o oposto, que não deve haver donos e que qualquer um está livre para alterá-los. Segundo Wazlawick [2013], em qualquer um dos casos, é importante que os artefatos sejam submetidos a um sistema de controle de versão, pois caso alguma mudança indevida seja feita, é possível ser desfeita.

2.1.1 Modelos de Processo

Segundo Wazlawick [2013], modelos de processo aplicados ao desenvolvimento de software também são chamados de "ciclos de vida". Existem duas grandes famílias de modelos, os prescritivos e os ágeis.

2.1.1.1 Modelos de Processo Prescritivos

O modelo prescritivo mais emblemático é o Cascata, cuja sua característica é ser sequencial e ter fases bem definidas. Segue abaixo a explicação de alguns dos modelos de processo prescritivos mais conhecidos.

2.1.1.1.1 Prototipação Evolucionária

Segundo Wazlawick [2013], distinguem-se duas abordagens de prototipação:

1. *throw-away*, que são protótipos feitos apenas para estudar os aspectos do sistema e entender melhor seus requisitos, reduzindo os riscos, porém depois que o protótipo é usado, ele é descartado;

2. *cornerstone*, aqui o propósito do protótipo é o mesmo que o *throw-away*, porém no final, o protótipo não é descartado, ao invés disso ele vai sendo evoluído até se tornar um sistema que pode ser entregue.

A prototipação evolucionária se baseia na técnica de prototipação *cornerstone*, fazendo com que seja necessário um planejamento mais cuidadoso do protótipo do que no *throw-away*. Este modelo é interessante quando é difícil fazer com o que o cliente informe os requisitos do sistema ou quando o cliente não sabe muito bem os requisitos. Porém, este modelo não é muito bom na questão à previsão de tempo para desenvolvimento ou em relação à gerência de projeto, pois muitas vezes o engenheiro de software apressa o desenvolvimento para que o protótipo entre em produção, causando problemas.

2.1.1.2 Modelos Ágeis

Em geral, os modelos ágeis têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos envolvidos [WAZLAWICK, 2013]. Diferente dos modelos prescritivos, que são como "receita de bolo", os modelos ágeis focam em valores humanos e sociais. Abaixo é apresentado alguns dos modelos ágeis mais famosos.

2.1.1.2.1 Scrum

Para Wazlawick [2013] o *Scrum* é um modelo ágil para a gestão de projetos. Dentro do *Scrum*, um dos conceitos mais importantes é o *sprint*, que consiste em um ciclo de desenvolvimento que dura de duas a três semanas.

No *Scrum*, as funcionalidades que precisam ser implementadas ficam em uma lista chamada *product backlog*. O *product backlog* não precisa ser completo já no início do projeto, já que o *Scrum* é uma metodologia ágil e utiliza o planejamento constante. Então pode-se iniciar apenas com as funcionalidades mais evidentes e relevantes e de acordo com o andamento do projeto e identificação de novos requisitos, o *product backlog* é atualizado.

Como dito, os *sprints* são um dos conceitos mais importantes do *Scrum*. Cada *sprint* é um ciclo de desenvolvimento de poucas semanas de duração sobre o qual se estrutura o *Scrum*. No início de cada *sprint* é feito uma reunião no qual a equipe prioriza

os elementos do *product backlog* e passam esses elementos para o *sprint backlog*, que é a lista de funcionalidades para serem feitas durante aquele *sprint* [WAZLAWICK, 2013].

Normalmente é utilizado alguma ferramenta para o auxílio da criação e manutenção do *product backlog* e do *sprint backlog*, fazendo com que todos da equipe saibam tudo o que já foi feito e o que está sendo feito no *sprint* atual.

2.1.1.2.2 XP - eXtreme Programming

O XP (*eXtreme Programming*) ou programação extrema surgiu por volta dos anos 90 nos Estados Unidos e é um modelo ágil recomendado para equipes pequenas e médias, seguindo uma série de valores, princípios e regras.

Dentro dos valores do XP, [WAZLAWICK, 2013] cita alguns, como: simplicidade, respeito, comunicação, *feedback* e coragem. Então, seguindo esses valores, o XP prega que a equipe deve ter uma forte ligação entre os membros, com uma ótima comunicação e respeito. Na parte de desenvolvimento, a simplicidade é utilizada para sugerir que a equipe deve se concentrar nas funcionalidades mais importantes do sistema, entre outros pontos. Como o projeto de software é reconhecido como um empreendimento de alto risco, deve-se obter o *feedback* o quanto antes para evitar eventuais falhas que podem vir a ocorrer.

Então, segundo [WAZLAWICK, 2013] o XP preconiza mudanças incrementais e *feedback* rápido, além de considerar mudanças algo positivo e parte do processo. Também pode-se adicionar que o XP prioriza as funcionalidades mais importantes do sistema, de forma que se o trabalho não puder ser totalmente concluído, pelo menos as partes mais importantes terão sido.

2.2 Desenvolvimento global de software

Segundo [HERBSLEB; MOITRA, 2001], as últimas décadas têm testemunhado uma tendência constante e irreversível da globalização dos negócios, e negócios intensivos de software de alta tecnologia em particular. As forças econômicas estão transformando os mercados nacionais em mercados globais e com isso, gerando novas formas de concorrência e cooperação que vão além dos limites nacionais. Essa mudança está tendo um profundo impacto não só no mercado e na distribuição mas também no jeito em que os produtos são concebidos, desenhados, construídos, testados e entregues para o cliente [O'HARA-DEVEREAUX; JOHANSEN, 1994]

Ao longo dos últimos anos, softwares têm se tornado um componente essencial para quase todos os negócios. O sucesso de um negócio aumenta dependendo do uso do software como uma arma competitiva. De acordo com [HERBSLEB; MOITRA, 2001],

organizações têm procurado acesso a recursos qualificados e começaram a experimentar desenvolvimento remotamente localizados. Vários fatores tem acelerado essa tendência:

- A necessidade de capitalizar o conjunto de recursos globais usando recursos escassos com sucesso e competitivamente custoso, onde quer que esteja localizado;
- A vantagem do negócio estar próximo do mercado, incluindo conhecimento de cliente e condições locais, como o bem será gerado pelo investimento local;
- A rápida formação de corporações virtuais e times virtuais para explorar as oportunidades do mercado;
- Muita pressão para melhorar o tempo para o mercado usando diferentes fuso horários em um desenvolvimento ininterrupto;
- A necessidade por flexibilidade para capitalizar a fusão de oportunidades adquiridas onde quer que estejam presentes.

Porém, segundo Herbsleb e Moitra [2001], baseado tanto em estatísticas, modelagem de desenvolvimento e pesquisas, tarefas em desenvolvimentos distribuídos fisicamente demoram mais para serem realizadas do que em desenvolvimentos colocalizados e que a comunicação e coordenação são as principais causas desse atraso. Mas também não se pode negligenciar os benefícios em potencial da dispersão geográfica. Um bom exemplo é que se uma organização administrar bem os dias de folga de suas equipes e trabalhar entre os locais distribuídos, é possível dar foco para o prazo de entrega e os caminhos críticos do desenvolvimento usando os fuso horários. Teoricamente é possível expandir as 8-10 horas de trabalho diárias para 24 horas.

2.2.1 Dimensões do problema

A separação dos membros do projeto tem diversos efeitos em vários níveis [HERBSLEB; MOITRA, 2001]. Dentre esses efeitos, tem-se:

- **Problemas estratégicos**: Depois de definido quais locais participará de um projeto, é difícil decidir como dividir o trabalho entre todos os locais. As soluções é avaliar os recursos disponíveis de cada local, os níveis de perícia nas tecnologias, suas infraestruturas, etc. Uma organização ideal seria deixar os locais operarem independentes enquanto proverem uma comunicação fácil, flexível e efetiva entra os locais. Um número de modelos são possíveis e apropriados sob diferente circunstâncias e requerem mecanismos diferentes de coordenação [GRINTER; HERBSLEB; PERRY, 1999].

- **Problemas culturais**: Culturas diferem em várias dimensões críticas, como a necessidade de estrutura, atitudes através de hierarquias, senso de tempo e estilo de comunicação e em um desenvolvimento global de software, a cooperação entre indivíduos com diferentes culturas é essencial [HOFSTEDE; HOFSTEDE, 1997]. Por exemplo, um email para alguém de uma cultura onde a comunicação tende a ser direta pode parecer rude, já para outros pode ser normal.

- **Comunicação inadequada**: Desenvolvimento de software requer muita comunicação, principalmente nos estágios iniciais [FILHO, 2003]. Projetos de software tem basicamente dois tipos de necessidades de comunicação: o primeiro, a comunicação formal que precisa ser oficial, clara e de fácil entendimento; o segundo é uma comunicação vital e muito pouco formal, como se fossem "conversas de corredores"e que ajudam as pessoas estarem cientes do que está acontecendo, em que as pessoas estão trabalhando, quais estados as partes do projeto estão, quem tem mais perícia na área, etc [HERBSLEB; GRINTER, 1999].

- **Problemas no gerenciamento do conhecimento**: Sem uma informação efetiva e mecanismos de compartilhamento de conhecimento, gerentes não podem explorar os benefícios do desenvolvimento global de software. Quando líderes de projetos disseminam informações de *status* inadequadamente, os times não podem determinar quais tarefas estão no caminho crítico. Uma documentação pobre também pode causar ineficiência no desenvolvimento colaborativo.

- **Problemas no gerenciamento de projeto e processos**: A falta de sincronização quando times distribuem processos entre locais, pode ser crítico. Por exemplo, se em um time de desenvolvimento em um local define "código de unidade testado"diferente de outro time de outro local. Então, a sincronização depende de uma entrada e saída de dados entre as equipes de uma forma muito clara.

- **Problemas técnicos**: Nesta parte, geralmente os problemas são causados por incompatibilidade entre o formato dos dados e diferentes versões das ferramentas. Mas também há problemas nas configurações, gerenciamento na transmissão de dados, entre outros.

2.3 Desenvolvimento de Aplicações Móveis

A popularidade de dispositivos móveis faz com que a demanda de aplicações específicas sejam maiores, no entanto, os dispositivos móveis diferem daquelas encontradas em computadores pessoais [FORMAN; ZAHORJAN, 1994].

Normalmente, os dispositivos móveis apresentam uma série de restrições com o que o desenvolvedor precisa se ater na hora de desenvolver, dentre elas estão:

- Poder de processamento menor;
- Memória com capacidade reduzida;
- Fonte de energia de baixa duração;
- Tela de tamanho bastante reduzida;
- Mecanismos de dados diferentes do que os de um computador pessoal;
- Conexão de rede tem uma largura de banda reduzida.

Quando uma equipe vai desenvolver uma aplicação para um dispositivo móvel, uma das grandes dificuldades, segundo Johnson [2007] é projetar as telas da aplicação tendo em vista o tamanho reduzido, fazendo com que seja demandado um bom planejamento como o dispor dos elementos na tela. Também é necessário projetar a aplicação de forma com que seja bem eficiente em termos de recursos, usando somente o necessário, pois caso cause estresse no processador, pode causa congelamento da interface.

2.4 Arquitetura de Software

A arquitetura de software surgiu por volta dos anos 60, mas só se tornou popular na década de 90. Sua importância aparece quando os softwares começam a se tornar mais complexos e consequentemente é necessário tomar decisões de quais elementos precisarão existir e como estes serão organizados de forma que o software se torne escalável. Segundo Shaw e Garlan [1996], a arquitetura de software pode ser definida como um sistema em termos de componentes computacionais e, os relacionamentos entre estes componentes e os padrões que guiam a sua composição e restrições.

Então, na arquitetura de software é preciso definir a estruturação e organização do software. Dentro disso, é preciso definir as estruturas que formarão o sistema, os protocolos de comunicação, a sincronização e o acesso a dados, escalabilidade e desempenho e alguns outros atributos de qualidade. Com isso, é possível afirmar que a arquitetura de software é uma abstração do sistema, é ela que define a organização fundamental dos componentes em um sistema.

Para a escolha da arquitetura que irá ser usada, é feita com base, entre outros fatores, em cima dos requisitos do sistema. Portanto, é extremamente importante ter um alto conhecimento dos requisitos do sistema para fazer a escolha certa da arquitetura que será usada, levando em conta os aspectos técnicos, sociais e de negócio. É importante conhecer estes aspectos pois facilita o entendimento da dificuldade, dos riscos e do impacto da implantação da arquitetura.

Uma das responsabilidades do arquiteto de software é a escolha das tecnologias que serão usadas para desenvolver o produto. A escolha das tecnologias envolve os seguintes aspectos:

- Determinar se as ferramentas disponíveis para suportar estas escolhas de tecnologia (IDEs, simuladores, ferramentas de teste, etc.) são adequadas para o desenvolvimento prosseguir.
- Determinar a extensão da familiaridade interna, bem como o grau de apoio externo disponível para a tecnologia (tais como cursos, tutoriais, exemplos, entre outros) e decidir se isso é adequado para prosseguir;
- Determinar se uma nova tecnologia é compatível com as outras tecnologias já existentes. Por exemplo, a nova tecnologia pode ser utilizada em conjunto com as outras já existentes?

2.4.1 Atributos de Qualidade

Quando um novo projeto de software é iniciado, é esperado que este consiga atender as qualidades requeridas. Então, a arquitetura escolhida precisa ser condizente com as qualidades requeridas e para entender como alcançar estas qualidades, é necessário expressar e entender os atributos de qualidade.

Para Bass, Clements e Kazman [2013], um atributo de qualidade é uma propriedade mensurável ou testável de um sistema e é usado para indicar quão bem o sistema satisfaz as necessidades dos seus *stakeholders*.

Ainda segundo Bass, Clements e Kazman [2013], para definir os requisitos existem várias técnicas, tais como caso de uso, *user stories*, sistemas já existentes, documentação, entre outros. Mas independente da origem, todos os requisitos abrangem as seguintes categorias:

1. **Requisitos funcionais**: descreve como o sistema deve ser comportar e reagir a estímulos em tempo de execução;
2. **Requisitos de atributo de qualidade (Requisitos não funcionais)**: é um complemento de um requisito funcional ou qualificação do produto global. Exemplos de requisitos não funcionais são: a rapidez em que uma função deve ser executada, o tempo para implementar o produto, uma limitação nos custos operacionais, etc;
3. **Restrições**: as restrições são decisões que o arquiteto de software não tem liberdade de escolha. Por exemplo, definir a linguagem de programação que será usada é uma restrição e cabe ao arquiteto de software definir, porém fatores externos pode fazer

com que o arquiteto tenha que aceitar restrições (caso um não há tempo de treinar a equipe em uma nova linguagem).

Já que os requisitos funcionais não são o suficiente para definir a arquitetura que será usada, os atributos de qualidade tem papel fundamental na definição da mesma. Depois de formado os requisitos funcionais, pode-se dividir em conjuntos e subconjuntos de funcionalidades para facilitar a atribuição a diferentes elementos arquitetônicos.

Portanto, a capacidade de realizar tarefas para o qual foi planejado, é chamado de funcionalidade. Todavia, é necessário que estas funções sejam relacionadas aos atributos de qualidade.

2.4.2 Padrões arquiteturais

Na engenharia de software, os padrões são utilizados por desenvolvedores que buscam soluções já existentes para solucionar problemas comuns em desenvolvimento de software. Para Bass, Clements e Kazman [2013], os padrões ajudam a promover boa prática de projeto, utilizando experiências existentes e comprovadas em desenvolvimento de software. Um padrão arquitetural estabelece uma relação entre:

- **Contexto**: é uma situação recorrente, comum no mundo que dá origem a um problema;
- **Problema**: o problema, adequadamente generalizado, que surge no contexto dado. A descrição do padrão descreve tanto o problema e suas variantes, quanto quaisquer forças complementares ou opostas. Já a descrição do problema geralmente inclui atributos de qualidade que devem ser atendidos;
- **Solução**: uma solução arquitetônica bem-sucedida para o problema, devidamente abstraída. Esta solução descreve as estruturas arquitetônicas que solucionam o problema, incluindo o equilíbrio das forças atuantes. A solução para um padrão é determinada e descrita por:
  - Um conjunto de tipos de elementos (por exemplo, repositório de dados, processos e objetos);
  - Um conjunto e mecanismos de interação ou conectores (por exemplo, chamadas de método, eventos ou barramento de mensagens);
  - Um *layout* topológico dos componentes;
  - Um conjunto de restrições semânticas abrangendo topologia, comportamento de elementos e mecanismos de interação.

Sistemas complexos em geral utilizam vários padrões ao mesmo tempo. Um sistema móvel, por exemplo, pode empregar um padrão de arquitetura cliente-servidor de três camadas, mas dentro desse padrão também pode haver *caches*, *firewalls*, MVC e assim por diante. Além disso, cada um destes padrões pode empregar mais padrões.

2.4.3 Arquitetura em Camadas

O padrão em camada define a organização do software em serviços agrupados em camadas de abstração. As camadas são organizadas de forma com que cada camada só possa comunicar com a camada de cima ou de baixo de forma unidirecional. Isso significa então que a camada de cima provê serviços para a camada de baixo, fazendo com que as camadas têm dependência entre si. Geralmente as camadas inferiores oferecem serviços mais básicos como de infraestrutura, compondo os serviços das camadas superiores.

Algumas das diversas vantagens dessa arquitetura são:

- Facilita a compreensão, uma vez que os projetos são baseados em níveis crescentes de abstração, permitindo dividir um problema complexo em um conjunto de problemas menores;
- Facilita a manutenção, uma vez que as camadas são fracamente acopladas, assim alterações em uma camada afetam apenas a camada adjacente superior;
- Facilita a reutilização, uma vez que as camadas podem ser reutilizadas em diferentes contextos, desde que se respeitem as interfaces.

Porém, como qualquer outro padrão, essa arquitetura apresenta desvantagens também, dentre elas estão:

- Algumas das vezes é difícil estruturar o sistema na arquitetura de camadas, pois geralmente é difícil encontrar os níveis de abstração corretos. E mesmo se isso for possível, o desempenho do sistema pode ser afetado, sobretudo para o uso de uma arquitetura fechada [SHAW; GARLAN, 1996].
- Definir o número de camadas pode não ser fácil. O número de camadas não depende apenas das funcionalidades do sistema, mas também dependem dos requisitos não funcionais do mesmo.

2.4.4 Arquitetura Cliente Servidor

Nesta arquitetura, a aplicação é executada fisicamente distribuída em dois tipos de estruturas:

- Os clientes, que requerem um serviço;
- Os servidores, que são os responsáveis por proverem os serviços.

Essa arquitetura é geralmente utilizada quando um número grande de clientes requisitam serviços compartilhados simultaneamente. Porém, é necessário gerenciar os recursos ou serviços compartilhados, de forma com que tenha uma política de acesso. A principal vantagem dessa arquitetura é permitir a escalabilidade e disponibilidade do serviço. A arquitetura cliente servidor pode ter variações, como por exemplo um servidor central ou múltiplos servidores distribuídos. Esta arquitetura só é possível de ser utilizada se ambos o cliente e o servidor estiverem na mesma rede.

Nessa arquitetura os clientes precisam saber a identidade do serviço do servidor para realizar a requisição do mesmo, porém o servidor não sabe a identidade do cliente até que o cliente faça a requisição. Portanto, o fluxo dessa arquitetura é assimétrico, o que significa que os clientes iniciam interações invocando serviços de servidores. Além disso, um componente do servidor pode ser um cliente de servidor, um exemplo disso é um servidor usar um serviço de consulta de CEP de um outro servidor externo, neste caso esse componente do sistema é um cliente de outro sistema.

As principais desvantagens são que o servidor pode apresentar um gargalo no desempenho caso receba muitas requisições de uma só vez e se caso o servidor não tenha uma réplica do serviço, caso ele falhe, todos os clientes pararão de funcionar.

Então, nessa arquitetura, é possível utilizar réplicas do servidor para aumentar a escalabilidade ou a disponibilidade. Já que os aplicativos do cliente são separados do servidor, é muito fácil adicionar novos clientes a um sistema.

3 PROCESSO PROPOSTO

Nesta seção é explanado o processo proposto, descrevendo as etapas do processo em si, a finalidade e objetivos do mesmo, o escopo dos projetos que abrange, etc.

O objetivo deste processo é agilizar todo o desenvolvimento de um aplicativo para dispositivos móveis, onde as equipes estão distribuídas fisicamente. Este processo foi criado para ser utilizado em projetos de desenvolvimento para dispositivos móveis, já que outros tipos de desenvolvimento requerem atenções diferentes na hora de desenvolver. Um foco importante deste processo é o gerenciamento das equipes distribuídas e como ambas devem fazer para que se obtenha o melhor resultado final sem que a distância física cause grandes problemas.

Este processo tenta tratar todos os aspectos de um desenvolvimento de aplicativos para dispositivos móveis, desde como será a comunicação entre as equipes até a prototipação e o aplicativo final.

Para facilitar na explicação de como são as etapas deste processo, é utilizado a figura 1 como auxílio para que seja fácil de visualizar e compreender a explicação. Como a figura 1 mostra, o processo é dividido em etapas.

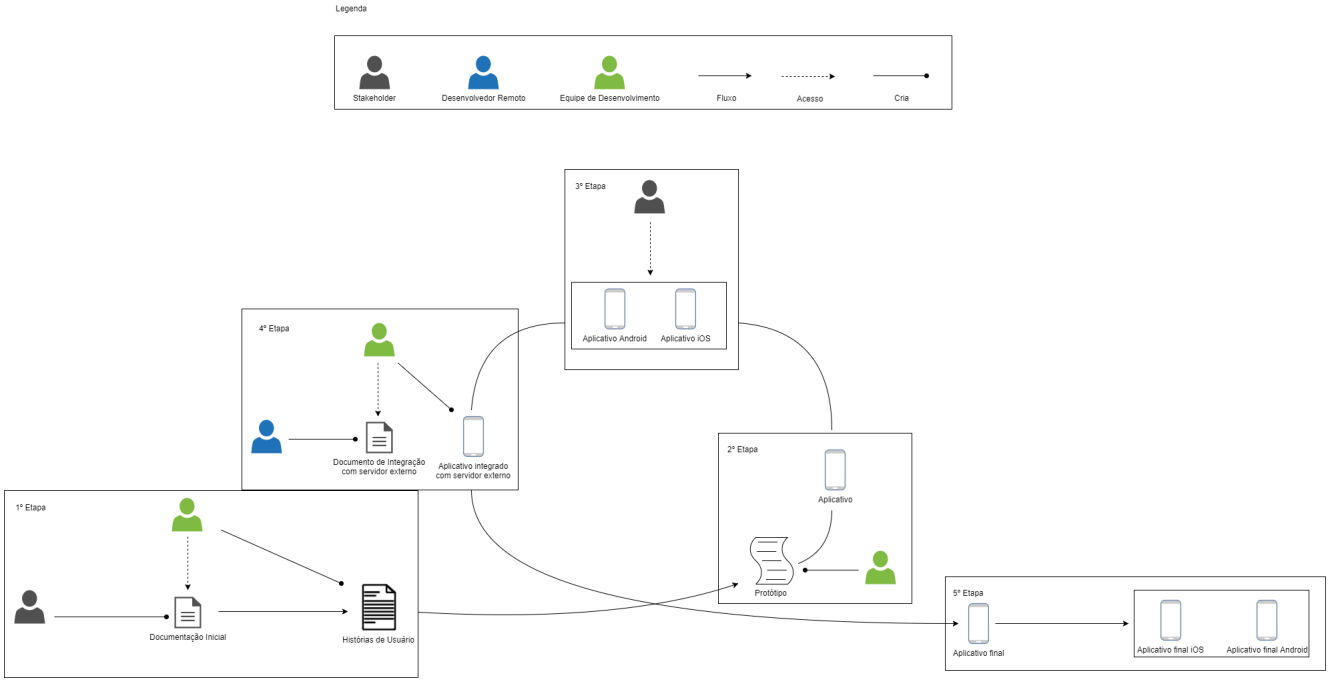


Figura 1 – Fluxo do processo proposto.

3.1 Etapas do Processo

Em projetos de desenvolvimento de software, normalmente se têm duas partes, a equipe de desenvolvimento e a parte interessada ou o *stakeholder*. Uma das características deste processo é de ter sido feito para essas mesmas equipes estarem fisicamente distribuídas, porém sem isso ser um problema ao longo do desenvolvimento. Então, a partir disto, antes de utilizar o processo, é necessário ambas as equipes definirem como será feita a comunicação entre ambas, definir os meios de comunicação.

Como já dito, o processo supõe que as equipes estão fisicamente distribuídas, o que dificulta no encontro pessoalmente de ambas. Por isso, é de extrema importância definir os meios de comunicação que serão utilizados ao longo de todo o projeto. Hoje em dia há diversas maneiras de ter uma comunicação a distância, como ligações telefônicas, emails, ferramentas on-line, etc. Um grande detalhe que a equipe deve ficar atenta é que na maioria das vezes o conteúdo da conversa entre ambas deve ser registrada e arquivada, pois muito das vezes serve como base para outras partes do desenvolvimento, por exemplo um email contendo informações do que deve ser mudado no aplicativo móvel que serve como base para a equipe de desenvolvimento fazer os ajustes.

Há casos em que uma comunicação mais rápida e até mesmo informal é melhor ou a comunicação em si não tem necessidade de ter seu conteúdo registrado, as equipes podem escolher mais de um meio de comunicação. Por exemplo, para comunicações rápidas, apenas para sanar dúvidas, ligações telefônicas podem ser melhores, porém, para solicitar mudanças no sistema, uma ferramenta on-line pode se encaixar melhor, já que fica registrado o que é para ser feito e ambas as equipes tem ciência do que está sendo e do que já foi feito.

Um ponto importante é que ambas as equipes estejam sincronizadas com os meios de comunicação. Se uma das partes não tem o domínio ou não utiliza corretamente o meio que foi definido, isso pode causar problemas na comunicação podendo afetar futuramente outras partes do processo como a implementação do aplicativo móvel, por exemplo.

Com os meios de comunicação definidos, o processo parte para as etapas do desenvolvimento.

3.2 Etapa 1 - Requisitos

O processo começa pela à etapa de requisitos do aplicativo móvel. Com as equipes distribuídas, é complicado realizar uma reunião de algumas horas para a parte interessada explicar como será o funcionamento do aplicativo, as funcionalidades, o objetivo e o que esperam do aplicativo final, para a equipe de desenvolvimento. Baseado nisso, é necessário que a parte interessada crie um documento inicial contendo informações sobre o aplicativo

móvel.

Este documento servirá como uma explicação do que o aplicativo deverá se tornar, suas funcionalidades, finalidades e características. Espera-se que a documentação inicial apresente alguns tópicos importantes para auxiliar a equipe de desenvolvimento na hora da extração de requisitos. Estes tópicos são:

- **Introdução sobre o aplicativo**: Neste tópico deve ser apresentado uma introdução sobre a área em que o aplicativo móvel irá atuar, uma contextualização do mesmo;
- **Justificativa**: Este tópico deve apresentar a justificativa do aplicativo existir;
- **Caracterização do problema**: Este tópico tem como finalidade descrever qual o problema que o aplicativo deve tratar;
- **Objetivo e funcionalidades**: Este tópico deve conter o objetivo do aplicativo móvel, junto com a descrição das funcionalidades do mesmo;
- **Necessidades**: Este tópico deve conter as necessidades que o aplicativo móvel deverá ter, como integração com servidor externo, etc.

Este documento inicial deve ser entregue para a equipe de desenvolvimento através do meio definido. A equipe de desenvolvimento tem a tarefa de ler e analisar todos os pontos importantes deste documento, juntamente com a descrição das funcionalidades e a partir desta base extrair os requisitos que o aplicativo móvel terá.

Como este processo tem como uma das características agilizar o desenvolvimento de um aplicativo móvel e auxiliar na comunicação entre ambas as equipes, ele se baseia em modelos de processos ágeis, tais como *XP* e *Scrum*.

Para facilitar, documentar e descrever os requisitos do aplicativo móvel, este processo utiliza uma modificação das histórias de usuário, que é comumente utilizada em conjunto com o modelo ágil *XP*.

Cada história de usuário, segundo Pressman e Maxim [2016], similar aos casos de uso, é feita a partir de conversas com o cliente ou parte interessada, onde o mesmo conta uma história de como irá funcionar uma parte do sistema e esta história é escrita pelo cliente e colocada em uma ficha. O cliente então atribui um valor à história se baseando no valor global do recurso, ou seja, priorizando as que tem maior valor.

Porém, diferente da forma como as histórias de usuário são feitas, este processo faz uma modificação em como cada história é criada. Neste processo, ao invés do cliente ou a parte interessada escrever as histórias, a equipe de desenvolvimento que as escreve. O intuito disto foi tentar agilizar o desenvolvimento, aliviando o cliente de ter de escrever as

histórias de usuário tendo em vista que já forneceram uma documentação inicial, e ao fator da comunicação não ser algo fácil, não podendo der conversas de fato com o cliente para a equipe de desenvolvimento ouvir as histórias de usuário do cliente. Mas, como a equipe de desenvolvimento pode não ter entendido exatamente o que cada funcionalidade do sistema deve fazer, o *stakeholder* deve analisar as histórias de usuário e validá-las, dizendo à equipe desenvolvedora se cada história de usuário está de acordo com o esperado ou não, e no caso de não estar, o *stakeholder* deve apresentar os motivos de não estar de acordo e a equipe de desenvolvimento fazer os ajustes na história para ser validade novamente pelo *stakeholder* e caso esteja em conformidade, o processo segue para outra etapa, porém, se ainda haver alguma inconformidade, deverá repetir esta etapa até que tudo esteja correto.

As histórias de usuário servirão como base para a equipe de desenvolvimento organizar a sequência de funcionalidades a ser implementadas, por este motivo elas devem estar de acordo com que o *stakeholder* quer, evitando ao máximo problemas futuros. Com as histórias de usuário feitas e validadas pelo *stakeholder*, é hora de começar a implementação do aplicativo móvel.

Antes da implementação do aplicativo móvel, deve-se levar em considerações alguns pontos. O primeiro deles é que mesmo o *stakeholder* validando todas as histórias de usuário, há a chance de ter alguma funcionalidade não condizente com o que o cliente realmente quer ou precise. Outro ponto importante é o acompanhamento do desenvolvimento do aplicativo móvel pela parte interessada. Como as partes são distribuídas fisicamente e o desenvolvimento é para aplicativos móveis, se torna complicado a equipe de desenvolvimento fornecer versões do aplicativo móvel para a parte interessada para os mesmos conseguirem usar, testar e validar as funcionalidades. Por estes motivos, este processo utiliza da prototipação evolucionária, fazendo dela um meio para o *stakeholder* validar novamente se a funcionalidade está de acordo com o esperado e testar ou usar o aplicativo móvel antes do mesmo estar totalmente implementado.

Pautado na ideia de prototipação evolucionária, este processo define que a implementação do aplicativo móvel deve prover protótipos utilizáveis e *cross-platform* (termo utilizado para dizer que o aplicativo móvel funcionará em várias plataformas com apenas uma implementação). Então, a partir deste ponto o processo segue para a segunda etapa, a etapa de prototipação.

3.3 Etapa 2 - Prototipação

Um dos pontos importantes do desenvolvimento neste processo é a escolha das tecnologias. Hoje em dia há muitas tecnologias para desenvolver aplicativos móveis *cross-platform*, porém há a necessidade de saber se a tecnologia que será utilizada tem suporte para prototipação de alguma forma.

Então nesta etapa do processo começa a parte de implementação do aplicativo móvel em si.

Como no desenvolvimento haverá várias histórias de usuário, para a implementação, este processo utiliza dos *sprints* retirados do modelo ágil *Scrum* para facilitar e organizar a equipe de desenvolvimento em qual funcionalidade deve ser implementada em sequência. O *Scrum* consiste em dividir o desenvolvimento em fases e cada uma dessas fases possui um tempo definido que pode ser um ciclo com uma duração pré-definida. Cada fase deste ciclo é chamado de *sprint*. Portanto, para cada história de usuário, temos um *sprint* para a implementação da mesma.

Em cada um desses *sprints* temos primeiro a codificação do protótipo. A equipe de desenvolvimento deve codificar a funcionalidade definida no *sprint* em questão e a partir do código feito, é gerado o protótipo. Portanto, o protótipo é uma versão utilizável do aplicativo móvel, o que facilitará nas próximas etapas do desenvolvimento e ao final do processo todo, este mesmo protótipo se tornará o aplicativo final.

No entanto, nem sempre a funcionalidade desenvolvida está de acordo com o que a parte interessada deseja. Por este motivo, na sequência da iteração do *sprint*, vem a parte de verificação e validação da funcionalidade desenvolvida neste *sprint* para se certificar que as funcionalidades estão sempre de acordo com o que o cliente ou a parte interessada deseja.

3.4 Etapa 3 - Verificação e Validação

Com o protótipo em mãos, nesta etapa, é hora do *stakeholder* testar e validar se a funcionalidade desenvolvida naquele *sprint* está de acordo com o que ele espera. Então a finalidade desta etapa e se certificar que cada implementação da equipe de desenvolvimento está de acordo com os requisitos e que no futuro o aplicativo final não acabe se tornando algo que não era o esperado.

O *stakeholder* também deve testar se a funcionalidade está de acordo em ambas as plataformas móveis. Caso o *stakeholder* ache algo que não está de acordo, ou que precisa ser melhorado, o mesmo informa a equipe de desenvolvimento apontando tudo que deve ser alterado para que na próxima iteração a equipe de desenvolvimento coloque estas alterações para serem desenvolvidas e consequêntemente testadas e validadas novamente pelo *stakeholder*, garantindo assim que sempre o protótipo estará de acordo com o esperado.

Caso o *stakeholder* dê a validação positiva e que está de acordo com o esperado, o processo segue para a próxima etapa.

3.5 Etapa 4 - Integração com Servidor Externo

Está etapa não é necessária em alguns desenvolvimentos. Isto porquê há aplicativos móveis que não requerem integração com um servidor externo, o mesmo atua apenas no dispositivo móvel. Porém, em muitos casos, o aplicativo necessita uma integração com um servidor externo para processamento de dados, análise de dados, busca de dados, etc.

Caso o aplicativo móvel não necessite de integração com um servidor externo, então o processo parte para a próxima etapa, caso contrário, deverá seguir as especificações desta etapa.

Esta etapa assume que o cliente ou a parte interessada já tem um servidor externo funcionando e que há uma equipe especialista para fornecer a documentação necessária para que a equipe de desenvolvimento possa realizar a integração.

Então, esta equipe especializada deve criar um documento descrevendo como será a integração, qual arquitetura será utilizada, acesso ao servidor e alguma forma de testes para que a equipe de desenvolvimento teste a integração e que não afete o servidor externo. Esta documentação deve ser muito bem feita e guardada em sigilo para que a parte interessada não seja prejudicada caso alguma informação de acesso seja divulgada, portanto, ambas as equipes devem se atentar à qual meio de comunicação esta informação deverá ser passada.

Com a documentação, a equipe de desenvolvimento é responsável por integrar a funcionalidade desenvolvida no *sprint* com o servidor externo e testar a integração. Após esta etapa, as iterações continuam até que todas as funcionalidades do aplicativo móvel estejam implementadas ao protótipo, validadas e integradas, e só assim o processo parte para a última etapa.

3.6 Etapa 5 - Aplicativo Final

Nesta etapa o processo já sai das iterações dos *sprints* e acontece apenas uma vez. Portanto, nesta etapa a equipe de desenvolvimento é responsável por transformar o protótipo no aplicativo final de fato. Então o protótipo que foi sendo desenvolvido ao longo do processo se transformará de fato no aplicativo final. O jeito que o protótipo se tornará o aplicativo final depende da tecnologia, há tecnologias que necessitam exportar para as plataformas móveis e outras o próprio protótipo já está nos formatos para as plataformas móveis.

Nesta etapa também se encerra o processo por completo.

4 PROCESSO APLICADO

Neste capítulo será mostrado como o processo proposto foi aplicado em um desenvolvimento real entre duas equipes, mostrando passo passo o que foi feito, como foi feito, o que foi utilizado para o desenvolvimento e o resultado do aplicativo final.

Neste trabalho foi desenvolvido um aplicativo móvel para a empresa COPEL (Companhia Paranaense de Energia Elétrica). Este aplicativo tem como objetivo inovar o banco de dados de cadastro de empregados de empresas terceirizadas já utilizado pela COPEL. Este aplicativo móvel visa otimizar as vistorias realizadas pelos técnicos de segurança do trabalho aos funcionários das empresas terceirizadas, durante o Programa Preservando a Vida (PPV), inclusive podendo ser utilizado pelos fiscais de obra da empresa em suas inspeções anuais de segurança ou a qualquer momento, por exemplo, durante a realização de uma tarefa.

Então antes de aplicar o processo, foi necessário realizar algumas tarefas. A primeira delas era entrar em contato com a COPEL e identificar quem era o responsável por dar as informações que a equipe de desenvolvimento precisava. Após uma conversa telefônica, foi identificado o *stakeholder* responsável por auxiliar a equipe de desenvolvimento durante todo o processo.

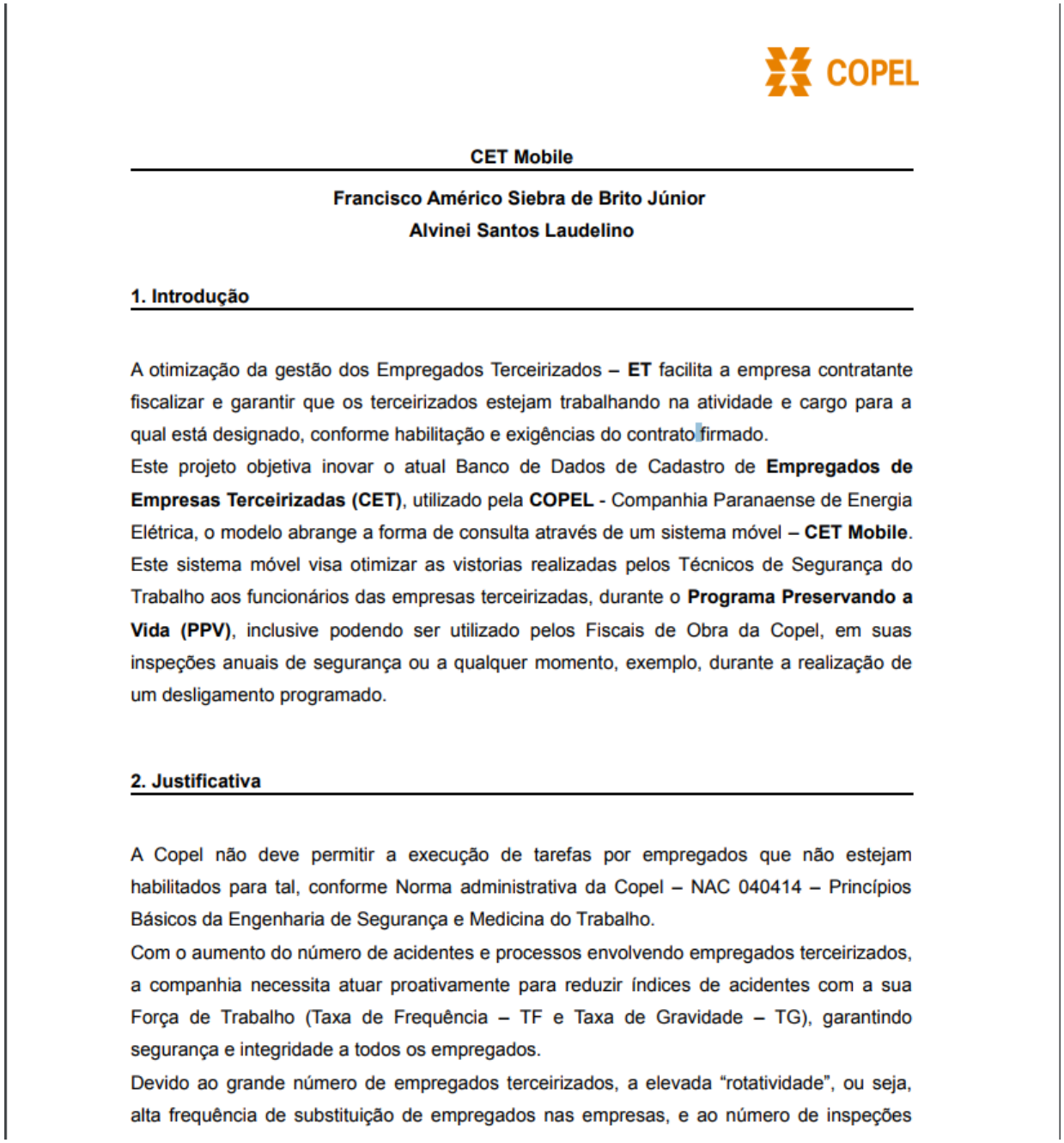
Com o *stakeholder* definido, ambas as equipes discutiram sobre os meios de comunicação que iriam utilizar. Para facilitar no gerenciamento das tarefas e análise em qualquer hora do que estava sendo feito, foi definido que seria utilizado a ferramenta Trello. Esta ferramente permite que várias pessoas possam ter acesso à algum projeto criado, definindo tarefas, prazos, anexando arquivos ou imagens, de uma maneira fácil. Também foi definido que para alguns assuntos, seria utilizados emails.

4.1 Etapa 1 - Requisitos

No início do processo, o *stakeholder* da COPEL mandou para a equipe de desenvolvimento um arquivo contendo as informações descritas pelo processo, como na figura 2. Este documento foi a base para a equipe de desenvolvimento conseguir extrair as histórias de usuário.

A partir deste documento inicial, a equipe de desenvolvimento conseguiu extrair as histórias de usuário para o desenvolvimento. Foi definido que seria colocado as histórias de usuário no Trello e a partir daí o *stakeholder* da COPEL acessaria o Trello, analisaria as histórias de usuário e validava se estava tudo de acordo com o esperado.

Para uma melhor organização, foi criado três quadros no Trello. Um para as



COPEL

CET Mobile

Francisco Américo Siebra de Brito Júnior
Alvinei Santos Laudelino

1. Introdução

A otimização da gestão dos Empregados Terceirizados – **ET** facilita a empresa contratante fiscalizar e garantir que os terceirizados estejam trabalhando na atividade e cargo para a qual está designado, conforme habilitação e exigências do contrato firmado.
Este projeto objetiva inovar o atual Banco de Dados de Cadastro de **Empregados de Empresas Terceirizadas (CET)**, utilizado pela **COPEL** - Companhia Paranaense de Energia Elétrica, o modelo abrange a forma de consulta através de um sistema móvel – **CET Mobile**.
Este sistema móvel visa otimizar as vistorias realizadas pelos Técnicos de Segurança do Trabalho aos funcionários das empresas terceirizadas, durante o **Programa Preservando a Vida (PPV)**, inclusive podendo ser utilizado pelos Fiscais de Obra da Copel, em suas inspeções anuais de segurança ou a qualquer momento, exemplo, durante a realização de um desligamento programado.

2. Justificativa

A Copel não deve permitir a execução de tarefas por empregados que não estejam habilitados para tal, conforme Norma administrativa da Copel – NAC 040414 – Princípios Básicos da Engenharia de Segurança e Medicina do Trabalho.
Com o aumento do número de acidentes e processos envolvendo empregados terceirizados, a companhia necessita atuar proativamente para reduzir índices de acidentes com a sua Força de Trabalho (Taxa de Frequência – TF e Taxa de Gravidade – TG), garantindo segurança e integridade a todos os empregados.
Devido ao grande número de empregados terceirizados, a elevada "rotatividade", ou seja, alta frequência de substituição de empregados nas empresas, e ao número de inspeções

Figura 2 – Documento inicial.

histórias aguardando aprovação, ou seja, não foram analisadas ainda; histórias de usuário para serem revisadas, são as histórias que precisam de alguma mudança e por último as histórias de usuário que foram aprovadas. A figura 3 mostra o exemplo na prática.

Após as histórias de usuário estarem prontas, o processo seguiu para a segunda etapa.

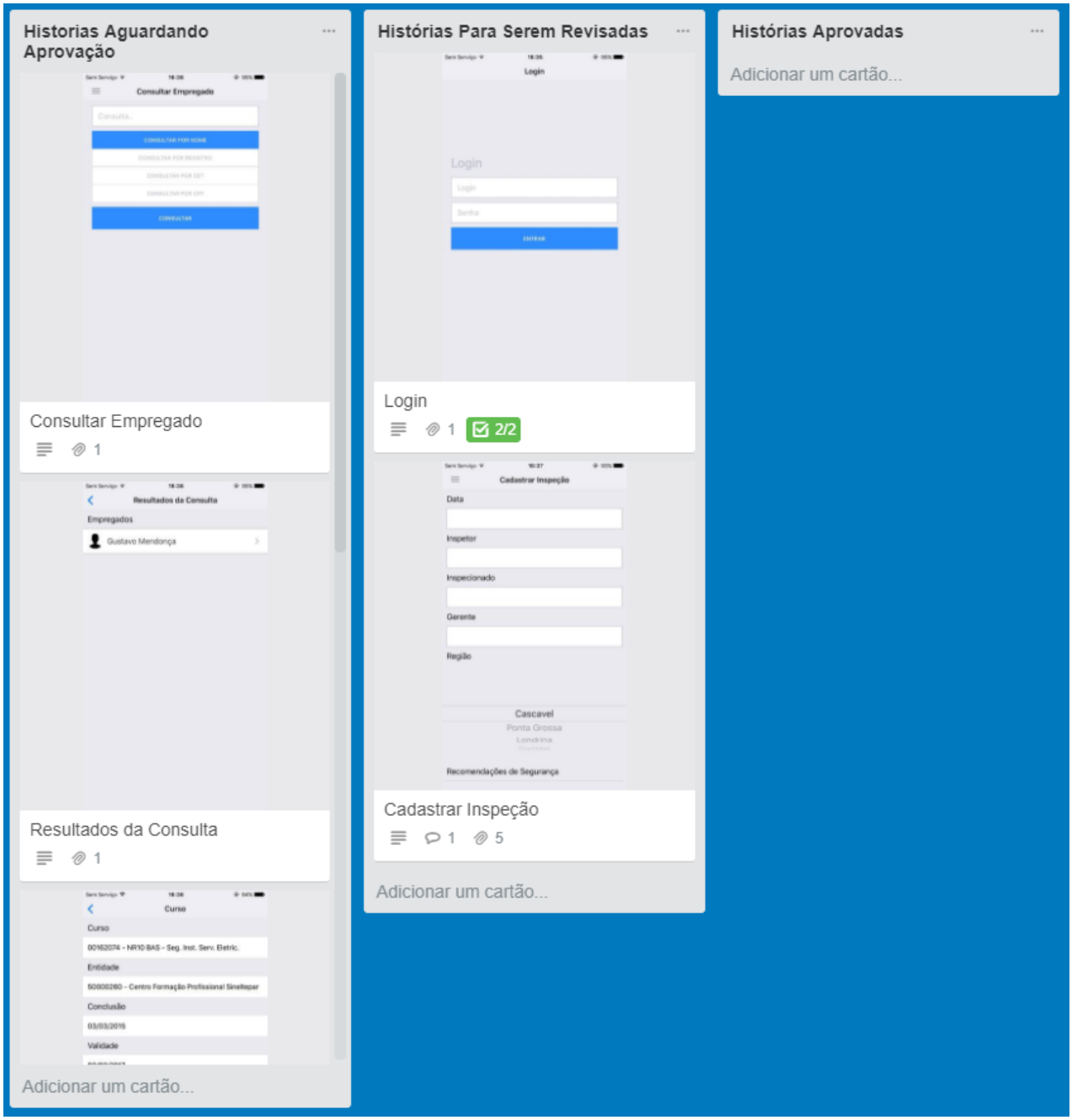


Figura 3 – Imagem exemplificando as histórias de usuário no Trello.

4.2 Etapa 2 - Prototipação

Nesta etapa ocorre a codificação de fato das funcionalidades e consequentemente a criação do protótipo. Então, antes de tudo, é necessário escolher a tecnologia que será utilizada tanto para a codificação quanto para a prototipação.

Então para este trabalho, foi definido que a tecnologia utilizada para a construção do aplicativo móvel em si seria o React Native. O React Native é uma biblioteca feita para a linguagem JavaScript.

A linguagem JavaScript foi criada por Brendan Eich e foi originalmente implementada como parte dos navegadores web para que *scripts* pudessem ser executados do lado do cliente na arquitetura cliente-servidor, realizando comunicações assíncronas e alterando o conteúdo do documento exibido. Hoje em dia a linguagem JavaScript tem um grande destaque na comunidade de desenvolvedores, pois é uma linguagem interpretada, ou seja, não necessita compilação do código e é considerada uma linguagem com um bom desempenho de velocidade. Nas versões atuais, esta linguagem tem suporte para orientação a objetos.

A partir desta linguagem, o engenheiro de software da empresa Facebook, Jordan Walke, criou a biblioteca chamada React no ano de 2011. Esta biblioteca foi criada com o intuito de melhorar a performance com que navegadores web alteram a interface do usuário de forma dinâmica. No ano de 2015, o Facebook anunciou uma biblioteca que utiliza a biblioteca React, porém para desenvolvimento de aplicações móveis, a biblioteca React Native. As principais vantagens desta biblioteca é que além de executar as interfaces nos dispositivos de modo nativo, o código escrito para todas as plataformas é apenas um, em JavaScript, o que facilita no desenvolvimento, pois o programador apenas precisa ter o conhecimento desta linguagem e já é capaz de desenvolver aplicativos *cross-platforms*.

Então vemos que React Native é uma ótima tecnologia para desenvolvimento de aplicativos móveis que necessitam rodar em diversas plataformas. Por este motivo, foi definido que neste trabalho é utilizado esta tecnologia. Na figura 4, vemos um exemplo de código em JavaScript, onde seu produto é um aplicativo *cross-platform*, graças à biblioteca React Native.

Em auxílio do React Native, há um aplicativo chamado Expo, disponível gratuitamente para todas as plataformas, que faz com que rode o código escrito usando React Native em qualquer plataforma, facilitando muito testes e prototipação.

Então para o protótipo em si, foi utilizado o aplicativo Expo, no qual o *stakeholder* da COPEL instalou no seu dispositivo móvel de testes e a partir disto a equipe de desenvolvimento disponibilizava o aplicativo através do Expo.

Então, com tudo definido, começou o desenvolvimento em si do aplicativo móvel. A equipe de desenvolvimento dividiu as funcionalidades que seriam codificadas, a partir das histórias de usuário, em *sprints*, podendo variar o tempo de codificação entre uma a três semanas para cada *sprint*. Para cada *sprint*, era gerado um QR Code (código de barras bidimensional) que ao ser inserido no aplicativo Expo, o *stakeholder* da COPEL tinha acesso as últimas alterações feitas no protótipo, fazendo assim os testes serem muito fáceis, pois o *stakeholder* não precisava de nenhuma outra ferramenta ou instalar o protótipo no seu dispositivo várias vezes. A figura 5 mostra um exemplo de como é um dos códigos bidimensionais utilizados para carregar o aplicativo no Expo.

```
render(){
  return(
    <View style={{flex: 1}}>
      <ScrollView style={{marginBottom: 20}}>
        {/* Data Inspeção */}
        <Text style={styles.title}>Data</Text>
        <InputMask type={'datetime'} options={{format: 'DD/MM/YYYY'}} style={{marginHorizontal: 10}} autoCorrect={fals
        {/* Usuário */}
        <Text style={styles.title}>Usuário</Text>
        <Input style={{marginHorizontal: 10}} keyboardType="numeric" autoCorrect={false} value={this.state.usuario} or
        {/* Inspetor */}
        <Text style={styles.title}>Inspetor</Text>
        <Input style={{marginHorizontal: 10}} keyboardType="numeric" autoCorrect={false} value={this.state.inspetor} o
        {/* Inspecionado */}
        <Text style={styles.title}>Inspecionado</Text>
        <Input style={{marginHorizontal: 10}} keyboardType="numeric" autoCorrect={false} value={this.state.inspecionad
        {/* Região */}
        <Text style={styles.title}>Região</Text>
        <Switcher style={{marginVertical: 5, marginHorizontal: 10}} onChange={(regiao) => this.setState({regiao})} dir
          <TabButton value="CEL" style={{height: 40}} text="CEL" selected={this.state.regiao==="CEL" ? true : false} /
          <TabButton value="CTA" style={{height: 40}} text="CTA" />
          <TabButton value="LNA" style={{height: 40}} text="LNA" />
          <TabButton value="MGA" style={{height: 40}} text="MGA" />
          <TabButton value="PGO" style={{height: 40}} text="PGO" />
        </Switcher>
        {/* Recomendações de Segurança */}
        <Text style={styles.title}>Recomendações de Segurança</Text>
        <Input style={{marginHorizontal: 10}} multiline={true} autoCorrect={true} value={this.state.recomendacoes} onC
```

Figura 4 – Imagem exemplificando o código em JavaScript de uma funcionalidade do aplicativo móvel.

4.3 Etapa 3 - Verificação e Validação

Então após a equipe de desenvolvimento gerar este código em cada *sprint*, o *stakeholder* da COPEL utilizava o código e carregava a última versão do protótipo em seu dispositivo. Com o protótipo em mãos, o *stakeholder* analisava se as funcionalidades que devia ser implementadas naquele *sprint* está de acordo com o esperado. Caso esteja de acordo com o esperado, o desenvolvimento seguia para o próximo *sprint*, caso contrário, o *stakeholder*, através do Trello, informava a equipe de desenvolvimento o que precisava ser alterado. A imagem 6 mostra um exemplo de uma funcionalidade que foi implementada, porém, faltando alguns detalhes.

Então no próximo *sprint*, a equipe de desenvolvimento implementou o que foi solicitado e foi aceito pelo *stakeholder*.

4.4 Etapa 4 - Integração com Servidor Externo

No caso deste desenvolvimento, um ponto muito importante era a integração do aplicativo móvel com o servidor externo da COPEL que já existia. Antes deste desenvolvimento acontecer, a COPEL já utilizava uma aplicação web para realizar as tarefas necessárias, porém surgiu a necessidade de uma aplicação móvel com as mesmas funcio-

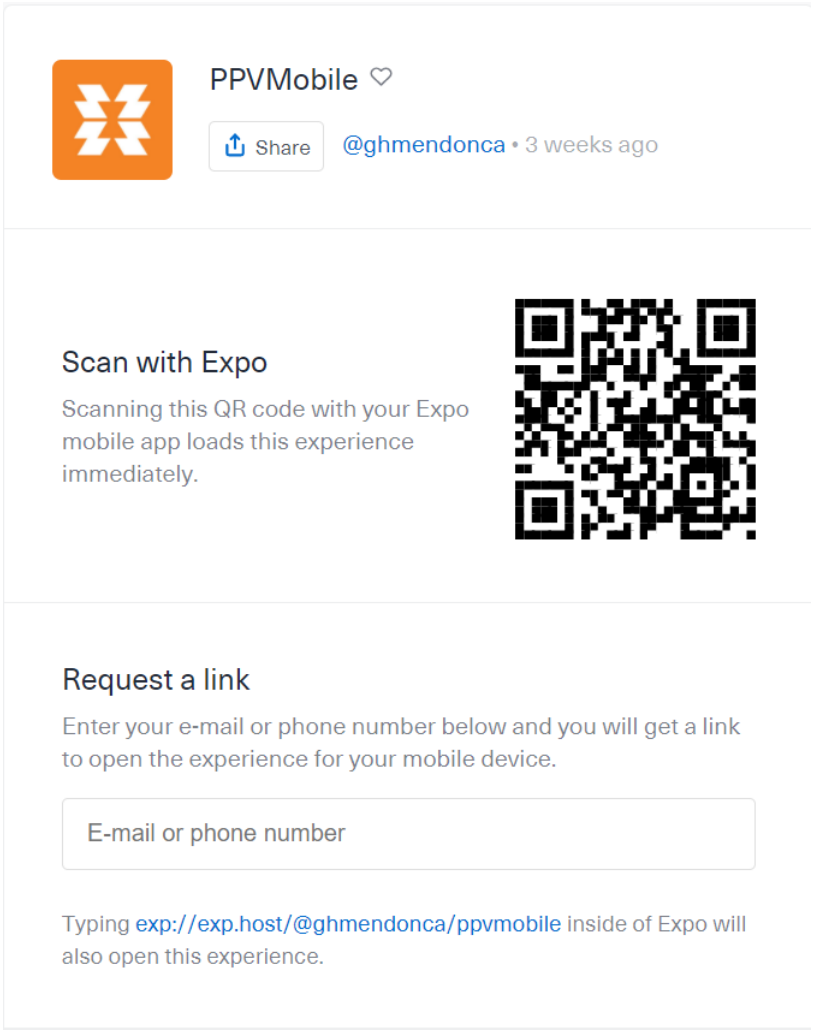


Figura 5 – Imagem exemplificando o código bidimensional gerado para fazer o carregamento do aplicativo em qualquer dispositivo que tenha o Expo instalado.

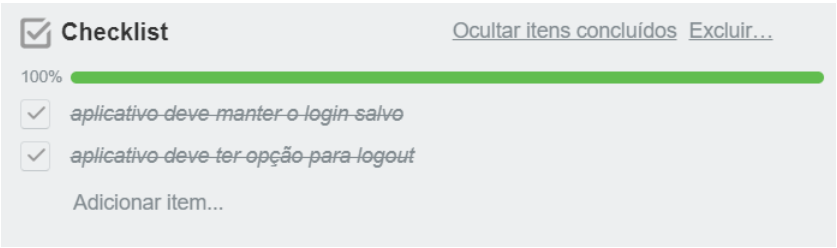


Figura 6 – Imagem exemplificando uma funcionalidade que precisa ser alterada.

nalidades e adição de mais algumas.

Então na etapa de integração com o servidor externo, entramos em contato com o *stakeholder* para ele indicar quem era o especialista técnico da COPEL que podia passar todas as informações necessárias para que a equipe de desenvolvimento conseguisse realizá-la.

Após conversas telefônicas, conseguimos o contato do especialista técnico da CO-

PEL e começou a discussão sobre como seria a integração. Como já existia uma aplicação web, a equipe de desenvolvimento sugeriu que fosse utilizado REST para a comunicação entre o aplicativo móvel e o servidor externo.

REST (*Representational State Transfer*) é uma abstração da arquitetura web. Foi apresentado e definido por Roy Fielding no ano 2000. O REST utiliza como base para a troca e manipulação de dados o protocolo HTTP (*Hypertext Transfer Protocol*), e cada um dos verbos do HTTP significa uma ação no servidor. Por exemplo, uma requisição seguindo o padrão REST utilizando o verbo GET do HTTP, significa que o servidor deve retornar uma informação. O REST tem uma descrição de como deve ser utilizado em boas práticas para agilizar e facilitar o desenvolvimento, por exemplo, se a funcionalidade do sistema é para alterar apenas um campo de um objeto, o ideal seria utilizar o verbo PATCH, porém também é possível utilizar os verbos POST e PUT para isso, mas com um desempenho menor.

O especialista técnico da COPEL aceitou a sugestão de utilizar o REST para a comunicação. O mesmo, então, criou a documentação necessária para que fosse possível a integração pela parte da equipe de desenvolvimento. Este documento consiste de uma série de requisições HTTP, com uma chave de acesso, e uma descrição do que cada uma das requisições realizava no servidor e a resposta que retornava de cada. A figura 7 mostra como é uma parte deste documento.

Após a integração com o servidor externo, o processo continua em iterações com os *sprints* até que todas as funcionalidades estejam devidamente implementadas, testadas e validadas, e só assim, o processo segue para a próxima etapa.

4.5 Etapa 5 - Aplicativo Final

Com o protótipo testado e validado pelo *stakeholder* da COPEL, integrado com o servidor externo de acordo com a documentação do especialista técnico da COPEL, o processo parte para sua parte final.

Nesta etapa, a equipe de desenvolvimento é responsável por entregar em si as distribuições de produção do aplicativo móvel, o que significa que a equipe de desenvolvimento deve mandar os arquivos para serem instalados nos dispositivos dos usuários finais, para todas as plataformas.

No caso deste trabalho, o React Native facilita o trabalho de exportação para todas as plataformas. Há um comando que apenas deve ser utilizado no final do projeto, onde todo o código em JavaScript é comprimido em um arquivo pequeno, é removido todo o ambiente de desenvolvimento do mesmo, para otimizar o desempenho e remover mensagens de erro ou avisos desnecessários para o usuário final e a partir daí é gerado um código nativo das plataformas que com este código, a equipe de desenvolvimento consegue

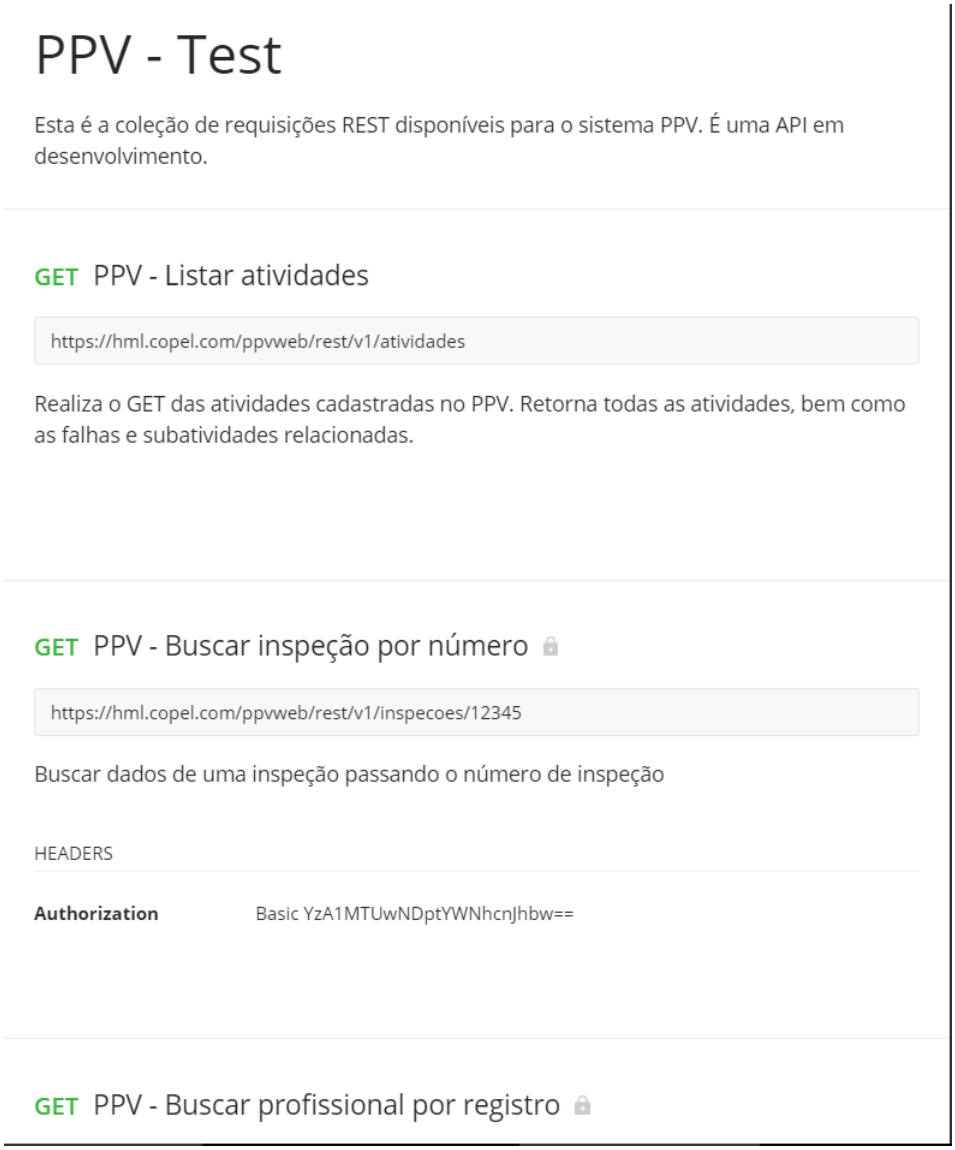


Figura 7 – Imagem mostrando uma parte do documento de integração com servidor externo.

manualmente compilar e instalar no dispositivo com a plataforma em questão. Portanto, exemplificando, é gerado uma pasta com a estrutura de um projeto em Android, onde há classes geradas automaticamente pelo React Native e a equipe de desenvolvimento precisa apenas abrir este projeto em qualquer ferramente que consiga compilar e transformar em um arquivo com a extensão APK (arquivo executável em dispositivos utilizando a plataforma Android) e disponibilzar este arquivo compilado para os usuários finais.

Com isto, o processo se encerra por completo.

5 RESULTADOS OBTIDOS

Após desenvolver e aplicar o processo proposto neste trabalho, foi possível perceber alguns resultados sobre o processo em si.

O primeiro ponto que foi possível notar, é que na etapa 1 do processo, o processo define que a equipe de desenvolvimento deve criar as histórias de usuário de acordo com documentos feitos pelo *stakeholder* para depois começar o desenvolvimento. Porém, com as histórias de usuário sendo criadas pela equipe de desenvolvimento, foi percebido que muitas das vezes não há o entendimento total do que é para ser feito nas funcionalidades do aplicativo, então na etapa de revisão e validação, houve muitas alterações no protótipo. Uma sugestão é realmente ter uma reunião com o *stakeholder* e ambos definirem as histórias de usuário juntos, ou utilizar um outro método de documentação de requisitos.

Na segunda etapa, as tecnologias escolhidas pela equipe de desenvolvimento se tornou um ótimo ponto para melhor aplicar o processo. Isso porquê facilitou a equipe de desenvolvimento a codificar em si o protótipo e facilitou com que a equipe de desenvolvimento disponibilizava o protótipo para o *stakeholder* poder testar na próxima etapa.

Na terceira etapa, o processo se mostrou ótimo, utilizando a prototipação para realizar testes e validação da parte do *stakeholder*. É de conhecimento geral que requisitos não são fáceis de serem atendidos e uma vez que o software final está pronto, se torna custoso para ser arrumado e o processo proposto trata muito bem o aspecto da implementação ser de acordo com o que o *stakeholder* deseja, fazendo com que no final do desenvolvimento, o protótipo vire o aplicativo final sem alterações nas funcionalidades. Percebemos que com o auxílio do aplicativo móvel Expo e com a ferramenta on-line Trello, tivemos um ótimo fluxo de desenvolvimento sem haver dúvidas do que era para ser feito e sempre fornecendo a última versão do protótipo para o *stakeholder*.

6 CONCLUSÃO

O objetivo proposto neste trabalho foi alcançado através do projeto e implantação do processo proposto para uma aplicação de dispositivos móveis. Para este fim, foi realizado uma análise das características do aplicativo que a empresa parceira COPEL precisava, a fim de criar um processo que enquadre dentro das características apresentadas, como equipes fisicamente distribuídas. Essa análise foi relevante para chegar no processo final e aplicá-lo no desenvolvimento do aplicativo móvel.

Após a aplicação do processo proposto, foi analisado quais os principais aspectos do processo, mostrando o que funcionou bem no desenvolvimento e o que poderia ter sido diferente.

REFERÊNCIAS

BASS, L.; CLEMENTS, P.; KAZMAN, R. Architectural tactics and patterns. *Software Architecture in Practice*, p. 214, 2013.

FILHO, W. de P. P. *Engenharia de software*. [S.l.]: LTC, 2003. v. 2.

FORMAN, G. H.; ZAHORJAN, J. The challenges of mobile computing. *Computer*, IEEE, v. 27, n. 4, p. 38–47, 1994.

GRINTER, R. E.; HERBSLEB, J. D.; PERRY, D. E. The geography of coordination: Dealing with distance in r&d work. In: ACM. *Proceedings of the international ACM SIGGROUP conference on Supporting group work*. [S.l.], 1999. p. 306–315.

HERBSLEB, J. D.; GRINTER, R. E. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software*, IEEE, v. 16, n. 5, p. 63–70, 1999.

HERBSLEB, J. D.; MOITRA, D. Global software development. *IEEE software*, IEEE, v. 18, n. 2, p. 16–20, 2001.

HOFSTEDE, G.; HOFSTEDE, G. Cultures and organizations software of the mind: intercultural cooperation and its importance for survival mcgraw hill. *New York*, 1997.

JOHNSON, T. M. *Java para dispositivos móveis: desenvolvendo aplicações com J2ME*. [S.l.]: Novatec Editora, 2007.

O'HARA-DEVEREAUX, M.; JOHANSEN, R. Globalwork: Bridging distance, culture, and time. San Francisco, CA (USA) Jossey-Bass Pub., 1994.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016.

SHAW, M.; GARLAN, D. *Software architecture: perspectives on an emerging discipline*. [S.l.]: Prentice Hall Englewood Cliffs, 1996. v. 1.

SOMMERVILLE, I. et al. *Engenharia de software*. [S.l.]: Addison Wesley São Paulo, 2003. v. 6.

WAZLAWICK, R. *Engenharia de software: conceitos e práticas*. [S.l.]: Elsevier Brasil, 2013. v. 1.