Kanban e TRIZ – Abordagem multidisciplinar para o desenvolvimento de um software para solução em IoT

Rafael Augusto Pissardo[1] Josué Marcos de Moura Cardoso[2] Valéria Cristina dos Santos Silva[3] Gabriel Oliveira Gomes[4] Yuzo Iano[5]

[1] Instituto de Pesquisas e Educação Continuada em Economia e Gestão de Empresas – Escola Superior de Agricultura "Luiz de Queiroz", Piracicaba – SP, Brasil
[2, 4, 5] Universidade Estadual de Campinas, Campinas - SP, Brasil
[3] Pontifícia Universidade Católica de Campinas, Campinas - SP, Brasil
card.jmm@gmail.com

Resumo: as soluções em Internet das Coisas (IoT) se baseiam pela conexão entre dispositivos diversos com elementos microcontroladores e módulos "*gateway*" com o intuito de automatizar processos e promovendo monitoramentos diversos. Em sistemas de irrigação automatizados as tecnologias de IoT consolidaram-se principalmente pelo custo acessível dessas soluções digitais. Para gerar um sistema robusto e confiável é necessário um "*software*" que atenda ao projeto integralmente, com possibilidade de incrementar e alterar ao longo do desenvolvido. Os métodos ágeis consolidaram-se no segmento da indústria das empresas de Tecnologia da Informação (TI), onde as rápidas mudanças do mercado impactam diretamente essas organizações. Dessarte, o objetivo principal deste trabalho foi descrever e exemplificar por meio de um estudo de caso a aplicação do Quadro Kanban e da TRIZ enquanto ferramentas auxiliares para a pessoa desenvolvedora de "*software*". Este artigo se baseou em uma solução em IoT denominada Sistema Hidropônico Automatizado (SHA), onde as ferramentas da TRIZ utilizadas foram: "*Ideality Audit*"; "*Ideal System*"; "*Ideal Outcome*"; "*Smart Little People*"; "*Size Time Cost*"; "*Ideality Plot*". Por meio dessas ferramentas o trabalho buscou uma solução para problemas reais encontrados em projetos de "*software*". Como resultado foi observado que a pessoa desenvolvedora do "*software*" chegou em duas possíveis soluções: desenvolver a comunicação via mensageria ou via "*Application Programming Interface*".

Palavras-chave: Gestão de projetos de "*softwares*"; Teoria da Resolução de Problemas Inventivos (TRIZ); Kanban; IoT; Sistema Hidropônico Automatizado (SHA).

I. INTRODUÇÃO

No mundo globalizado atual, as tecnologias digitais que emergiram com o advento da indústria de semicondutores permitiram que novas soluções fossem criadas para a otimização de processos, redução de custos e recursos, e principalmente transformando a sociedade. A Internet das Coisas, do inglês "*Internet of Things*" (IoT) depontou nesse cenário ampliando a utilização de mecanismos digitais e sistemas autônomos, tanto no contexto das residências, quanto em setores mais robustos [1-4]. Novos modelos de mercado e de negócios surgiram nesse contexto amplo de digitalização, onde empresas de Tecnologia da Informação (TI) se disseminaram nas respectivas indústrias trazendo novas possibildaddes de protudos e serviços considerando sobretudo o uso da comunicação sem fio para monitoramento e acionamento de dispositivos e sistema à distância. Especificamente no que se refere ao desenvolvimento de "*softwares*", metodologias estruturadas na organização das equips se popularizam nesse contexto, uma vez possuem ferramentas e métodos consolidados para que os prazos e a qualidade do sistema sejam prioridade ao longo do projeto [5-8].

Os métodos ágeis se caracterizam pela interação entre as equipes, a empresa e o próprio cliente, portanto, a comunicação objetiva é um fator fundamental para o sucesso do projeto. Uma vez compreendido que o desenvolvimento de "*softwares*" contempla etapas gerenciáveis dentro de um projeto, os métodos ágeis se consolidaram nas empresas de IT por promover adaptações rápidas em um contexto corporativo mutável e competitivo [5, 7-10]. Essa agilidade se tornou fundamental para os padrões do mercado, logo as equipes trabalham continuamente para melhorar o produto final. Dados do "*13th State of Agile Report*" apontaram que 97% das empresas de tecnologia já incorporaram os métodos ágeis em suas rotinas de desenvolvimento de "*software*" [11].

O Guia *"Project Management Body of Knowledge PMBOK*" do "*Project Management Institue*" (PMI) oferece técnicas, métodos e ferramentas genéricos que podem ser aplicadas na gerência de projetos. Os métodos ágeis promovem mais hibridez na condição do projeto, com estruturas conceituais e Abordagens interativas e adaptativas [8]. Entre os métodos consolidados para a confecção de algoritmos o Kanban e o Scrum são utilizados em larga escala nas empresas de TI [7, 11]. Eles priorizam as interações e a equipe, mais que o processo e as ferramentas, a colaboração com os clientes é mais importante que a negociação do contrato, e ser responsivo às mudanças é mais importante do que seguir o plano. O "*framework*" Scrum é uma parte das metodologias ágeis que se espera aumentar a velocidade e flexibilidade no projeto de desenvolvimento de "*software*". O Kanban corrobora com a implantação de mudanças que não precedem a prescrição de papéis ou práticas específicas no início do projeto, com o foco na redução do desperdício de tempo e esforços, excluindo assim atividades que não agregam valor ao projeto. Os métodos ágeis apesar de consolidados nas práticas de desenvolvimento de "*software*" possuem métodos, práticas e ferramentas ágeis são uteis em diferentes tipos de projeto [12-17].

A resolução de problemas ao longo do projeto pode partir de aspectos inovadores, principalmente no segmento de desenvolvimento de "*softwares*". Essas soluções podem ser baseadas no aprendizado entre as pessoas integrantes da equipe e também pelos problemas solucionados anteriormente. Quando é descoberta, estruturou-se em documentações e compartilhou-se para evitar um ciclo de redescoberta. Ou seja, a inovação tornou-se fluida e contínua [7-10, 17]. A Teoria da Resolução de Problemas Inventivos (do russo "*Teoriya Resheniya Izobretatelskikh Zadatch*") (TRIZ) busca exatamente o oposto dessa organização. A TRIZ se refere ao método de cruzar diferentes disciplinas, com encapsulamento da lógica fundamental para a solução de problemas. A TRIZ foi desenvolvida em 1946 pelo soviético Genrich Altshuller. Inicialmente, foi pensada para os setores, mais tradicionais das engenharias no geral. Além disso, a TRIZ possui um aspecto fundamental no que tange as metodologias ágeis, sendo o destaque a interdisciplinaridade que esse método promove no desenvolvimento de "*softwares*". Um de seus pilares é que alguns problemas, ou subproblemas, são comuns nas diversas áreas do conhecimento. E, constantemente, pessoas reinventam soluções, outrora estabelecidas, tentando solucionar seus problemas a partir de um papel em branco [18-24].

Desenvolver um "*software*" com suas funcionalidades se tornaram mais robustos, sendo que pode haver uma certa complexidade de elaborar esses algoritmos. Portanto, para elaborar um código exige do profissional capacitado o reconhecimento de uma possível crise no sistema. Para tal, precisa-se de ferramentas que apoie o time na decisão e no controle da crise. Dessa maneira, se houver o entendimento de quão similar os problemas são, pode-se aproveitar soluções já documentadas e solucionar o problema alvo de forma mais rápida e eficaz [9, 10, 25, 26]. O uso da TRIZ na engenharia de "*software*" considera parâmetros, elementos físicos e não elementos abstratos, como métodos computacionais ou processos de desenvolvimento. Dessa forma, pesquisas e a aplicações da TRIZ são vastas e possibilitam a junção desses dois elementos, com o mensuramento do quão eficaz seria, em contextos incertos e sem respostas claras [18, 20, 21, 24, 26]. O gerenciamento de projeto, na Engenharia de "*Software*", envolve uma personagem fundamental: a pessoa arquiteta de "*software*". Ela é um dos pilares para uma boa execução e planejamento; as decisões desta personagem vão definir como o sistema será desenvolvido. Não importa se foram utilizados métodos empíricos ou preditivos [17, 22, 26]. Desta forma, o trabalho da pessoa arquiteta de "*software*" é entender, analisar e conciliar elementos em detrimento de outros elementos considerando soluções e tecnologias disponíveis. E é nesse aspecto que a TRIZ, a engenharia de "*Software*" e o gerenciamento de projetos se encontram.

Para tanto, por meio de um estudo de caso o objetivo principal deste trabalho foi descrever e exemplificar como a TRIZ pode auxiliar a pessoa desenvolvedora de "*software*", na amplitude do Gerenciamento do Projeto de "*Software*" em que ela estava inserida. Em outras palavras, foi

apresentado como as ferramentas disponibilizadas pela TRIZ podem auxiliar a pessoa arquiteta do "*software*" a gerenciar e desenvolver o projeto de forma mais eficaz e consistente. Para atingir tal objetivo, foi feito um estudo de caso de um Sistema Hidropônico Automatizado (SHA) baseado em IoT desenvolvido para produtores rurais do Rio Grande do Sul, onde o "*software*" foi desenvolvido a partir de projetos compartilhados na plataforma "*open-source*" GitHub [27]. Este sistema teve como finalidade automatizar a produção hidropônica e permitir controle remoto da pessoa responsável por aquele sistema produtivo. Além disso, com os dados obtidos em tempo real da área de plantio, a pessoa responsável pode prever colheita e plantio de novas mudas, excluindo a necessidade de realizar tais tarefas presencialmente. A metodologia, discussão e conclusão foram feitas baseadas em uma abordagem descritiva e qualitativa, aplicando o Quadro Kanban associado a TRIZ para descrever as etapas de elaboração desse projeto.

II. REFERENCIAL TEÓRICO

As soluções em IoT contemplam os sistemas, serviços, dispositivos e equipamentos que empregam e utilizam algum grau de automação, processamento e acurácia de dados, além de redes de sensores conectados em interfaces e "*gateways*", com a adoção de um protocolo de comunicação característica, que tornam atividades e ações mais otimizadas e com maior rendimento [1, 3, 4]. Para tanto, algoritmos devem ser desenvolvidos para atender a essas soluções, onde a estruturação do "*software*" pode ser estabelecida com ferramentas de gestão de projetos. No geral, o gerenciamento de projetos pode ser conduzido com o emprego de metodologias ágeis e de controle como o Quadro Kanban adicionado à abordagem de desenvolvimento adaptativo. Assim, pode-se aproximar conceitos ágeis ao desenvolvimento do projeto [5, 8, 10, 12, 14]. A TRIZ pode ser aplicada em projetos onde as soluções são baseadas em problemas inventivos, cuja as abordagens contribuem com o aparecimento de problemas recorrentemente. Os princípios básicos da TRIZ consideram a técinica, a evolução, a contradição, os recursos e a idealidade [17-21, 26]. Ao combinar os métodos ágeis e a TRIZ para o desenvolvimento de "*softwares*" voltados às soluções IoT, alguns benefícios e desafios podem ser observados e analisados, além das desvantagens também comum no segmento de desenvolvimento de algoritmos.

O Sistema Hidropônico Automatizado (SHA) utilizado como estudo de caso deste trabalho adotou ao longo do desenvolvimento do "*software*" a aplicação da metodologia ágil Quadro Kanban combinado com a ferrramenta TRIZ. Essas metodologias consolidaram-se na indústria de tecnologia, que ao longo dos últimos anos cresceu consideravelmente oferecendo soluções e serviços digitais importantes para o mundo contemporâneo [6-8]. Portanto, para elaborar um "*software*", ou algoritmo, robusto e que atenda aos usuários de forma íntegra e segura, as redes (ou sistemas) de monitoramento contendo soluções em IoT devem assumir premissas de organização e comunicação indispensável para que os processos ocorram dentro da conformidade [3, 4, 9, 12, 25].

Sistemas automatizados aplicados à agricultura colaboram com o gerenciamento efetivo dos recursos naturais, como a água por exemplo, sendo que as soluções de IoT fornecem mecanismos de baixo custo e com grande aplicabilidade inclusive nesse segmento. Ao combinar tecnologia com as atividades rurais e também cumprindo com a sustentabilidade do planeta, a automatização dos processos de irrigação com o uso de tecnologias IoT, com acionamento e monitoramento remoto sendo possível por um servidor em nuvem ("*cloud*") e interfaces gráficas de fácil acesso aos usuários, trazendo versatilidade e independência. No contexto recente da pandemia do COVID-19 as soluções em IoT contribuiram para com o monitoramento da proliferação do vírus da doença, adotando também mecanismos precisos de rastreabilidade [28-32]. Contudo, o desenvolvimento de "*softwares*", nos mais diferentes segmentos, deve seguir métodos efetivos de gestão e organização, para que as entregas sejam realizadas dentro do prazo.

Os métodos ágeis aplicados ao desenvolvido de "*softwares*" possuem configurações estruturadas conceitualmente, com abordagens iterativas e adaptativas, sendo que em contextos de engenharia de "*software*" elas também contribuem

positivamente para as entregas em cada etapa do projeto. A partir de relações e princípios, esses métodos consideram valores fundamentiais, sendo eles: usuários e interações são mais relevantes que processos e ferramentas; "*software*" funcionando, documentação despriorizada; colaboração com o cliente, negociação de contratos despriorizada; responder rapidamente às mudanças. Esses métodos surgiram na década de 1990, sendo continuamente aprimorados e revisados, colaboram com o aumento da eficiência, produtividade e também contribuindo com respostas mais efetvias aos clientes (que podem ser equipes internas da empresa, ou outros personagens). Nesse interim, as metodologias ágeis atendem a projetos de desenvolvimento de "*software*" com alterações tanto em requisitos, quanto nas habilidades dos indivíduos envolvidos e, também, nas tecnologias utilizadas. Um ponto a ser ressaltado refere-se à qualidade do "*software*" que pode estar relacionada com a especificação e implementação dos objetivos funcionais e o desempenho técnico das equipes [5, 8-10, 14].

Os investimentos estratégicos no segmento da indústria das empresas de TI para o desenvolvimento de "*softwares*" refletem parcelas importantes do orçamento corporativo, uma vez que existem processos relativos à tomada de decisão para minimizer prejuízos [7, 9]. Entre os métodos ágeis mais utilizados nesse sentido, o Kanban e o Scrum são predominante empregados para organização das equipes e das entregas, com estruturas consolidadas e personalizadas para cada empresa. Ainda, outros métodos ágeis disponibilizados na literatura são o Extreme Programming, o Lean Startup, o 5S, entre outros [10, 11]. Tais métodos geram competitivadade para as organizações por meio de melhorias na eficiência de seus processos e redução de seus desperdícios operacionais [12, 14], sendo que na engenharia de "*softwares*" o desenvolvimento particionado em etapas de forma enxuta, "*lean*", demonstrou resultados interessantes na indústria. Outro ponto relevante é o compartilhamento de informações para atingir os objetivos de entregas determinados no início do projeto [12, 21, 25].

O método ágil Kanban foi preconizado pela Toyota para promover maior organização nos estoques da empresa, onde o principal intuito se dá pelo mapeamento de todos os processos de abastecimento e fluxo, onde a produção obedece o princípio "*just in time*", muito importante na indústria de montage. Apesar do contexto de fábrica onde o Kanban se preconizou, atualmente ele é utilizado em diversos segmentos da indústria com finalidades plurais. Desde seu surgimento, o método ágil Kanban foi incorporado com três ferramentas, sendo elas: cartão – que representa uma ação, ou atividade, com coloração característica para Identificação dos processos; colunas – que representam o "*status*" dos cartões, ou seja, quais são os posicionamentos dos mesmos e também os fluxos que eles podem seguir; quadro – que representa a associação dos cartões alocados em cada coluna delineada para o projeto. À medida que o desenvolvimento vai sendo efetuado, os cartões avançam dentre os estágios pré-estabelecidos. Por outro lado, o Quadro Kanban também pode ser compreendido e estruturado como uma planilha de controle com referências visuais e organizacionais atreladas aos cartões que podem representar "*tickets*", ou demandas, sendo que as equipes são informados simultaneamente a cada mudanças ocorrida no projeto do "*software*". O Kanban contribui com as especificações atreladas ao sistema, que corresponde à descrição das funcionalidades do "*software*", de modo a incrementar mais solução ao problema, com o cumprimento e o atendimento das necessidades, pré-estabelecidas. Sendo assim, inicialmente devem ser detalhadas todas as funções do sistema almejado para permitir o desenho do mesmo, com sua construção e respectivos testes necessários [5, 7, 11, 12]. Outro método consolidado para projetos contendo o desenvolvimento de "*softwares*" é o Scrum, que permite a criação de "*frameworks*" e o estabelecimento de posições e funções bem definidas na equipe, onde os eventos são separados em "*sprints*" e artefatos divididos para dinamizar o projeto de "*software*" [13, 14, 16].

Em linhas gerais, uma teoria de resolução de problemas (TRIZ) deve conter: processos sistemáticos com etapas delimitadas; guias provenientes de soluções conhecidas; devem prevalecer métodos que permitam repetição, confiabilidade e independência; permitir adicionar à base de conhecimento das soluções inventivas; os

inventores, devem estar familiarizados com o projeto de inovação. A TRIZ se baseia na técnica, nas contratições, na idealide e nos recursos de campo e substancial [19; 23; 24]. A TRIZ foi desenvolvida inicialmente para projetos de engenharia mecânica, sendo que os princípios inventivos e os parâmetros de engenharia são amplamente aplicados aos procesos. No que tange o desenvolvimento de soluções ela também também pode ser considerada, onde os produtos podem ser "*softwares*" e "*hardwares*", com um modelo estruturado de funcionalidades. Uma ferramenta da TRIZ interessante para o desenvolvimento de "*software*" é a análise campo-substância para a identificação de falhas, erros ou mesmo da falta de componentes e/ou processos no sistema. Durante a análise de requisitos são distinguidos requisitos funcionais e não funcionais ou de qualidade relativos a um produto de "*software*". Os requisitos de qualidade são propriedades desejadas que ultrapassam a funcionalidade correta, como confiabilidade, integrabilidade, capacidade de manutenção, testabilidade ou modificabilidade. Entre algumas ferramentas disponibilizadas pela TRIZ, cita-se: Matriz de contradição – uma contradição na sentido mais amplo é um problema a ser resolvido; Tendências de evolução – solução de problemas TRIZ visualiza a evolução como um processo que tem um ponto (um ponto além do qual a necessidade de evoluir é não é necessário ou não é possível); Análise de Funções e Atributos (FAA) – técnica para formar uma compreensão do atual estado de um sistema mapeando seus elementos e suas interações; entre outros [18, 20, 21, 33].

A partir dos objetivos traçados para este trabalho, o Sistema Hidropônico Automatizado (SHA) foi idealizado para fornecer soluções de monitoramento e ação remotamente, contemplando ferramentas que se inserem no contexto das IoT. O desenvolvimento do "*software*" dessa solução foi estruturado com base no Kanban e na TRIZ, descritas acima. Os processos decriação se basearam na plataforma de desenvolvedores GitHub foi consultada, além de armazenar também o código escrito. O GitHub é uma plataforma de desenvolvimento colaborativo que permite o versionamento do projeto. Ou seja, com ele é possível desenvolver e submeter o trabalho para agregar à versão oficial do projeto em questão. Tal submissão fica disponível aos membros do projeto que podem comentar, sugerir alterações, aprovar ou recusar a inclusão à versão principal. Além disso, como é focada para o desenvolvimento de "*software*", ela possibilita a integração com diversas ferramentas que ajudarão na execução de testes automatizados e na publicação do código ao servidor oficial do projeto [27, 33].

Para tanto, o SHA se estruturou pela concepção de uma rede de sensores para monitoramento de parâmetros ambientais e climáticos, além de possuir mecanismos de acionamento remoto para ligar os dispositivos de irrigação. Conforme discutido por [29] o emprego de sensores a partir de soluções IoT em setores como a agricultura pode amenizar o desperdício de água, além de promover a automação de processos e atividades manuais. No sistema de irrigação construído pelos autores foi utilizado o microcontrolador *RaspberryPi*, compondo uma rede de sensores sem fio com um "*gateway*" que transmitia os dados pelo padrão "*Message Queuing Telemetry Transport*" (MQQT) para um servidor em nuvem e os dados explícitos na plataforma *Zigbee*. Outros protocolos também podem ser utilizados em redes de monitoramento e acionamento de pequena porte. [30] elaboraram um sistema baseado em IoT para regar automaticamente espaços residenciais, tais como jardins e quintais. O microcontrolador utilizado foi o Arduino MEGA com um módulo "*gateway*" Wi-Fi ESP-8266 e também sensores de parâmetros ambientais e climáticos, tais como umidade do solo, temperatura ambiente e umidade relativa do ar. O protocolo de comunicação foi o ZigBee, onde foi desenvolvido um aplicativo para celular para visualização dos dados coletados pelo nó sensor. A solução em IoT desenvolvida apresentou dinamismo e portabilidade no monitoramento.

Abaixo são apresentados as demais seções deste artigo, com a aplicação do Kanban e TRIZ para elaboração do "*software*" do SHA.

III. METODOLOGIA

Este trabalho por meio de um estudo de caso descreveu e apresentou as etapas envolvidas em um

projeto destinado à automação hidropônica desenvolvido em 2019. O projeto consistiu no desenvolvimento de um "*software*" voltado à aplicação descrita, onde foram utilizados os princípios e práticas contextualizados nas metodologias ágeis e também nos conceitos da TRIZ. Partindo do princípio que a engenharia de "*software*" possui complexidades multifatoriais. Os métodos ágeis foram escolhidos e estruturados conforme o "*framework*" de Cynefin, exemplificado na Figura 1 ("*The Cynefin Co*.") [34]. Recomenda-se esse "*framework*" com a utilização de ferramentas empíricas, ou seja, o processo de definição das funcionalidades do "*software*" passou pelas ações de experimentação, definição de próximos passos e resposta ao problema. Esse raciocínio prático pode ser comparado com o viés filosófico exposto por John Locke onde o mesmo propôs que "a resposta para o problema vem depois que o problema aparecem" [35].

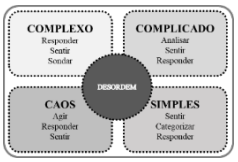

Figura 1. Exemplificação do "*framework*" elaborado para desenvolvimento do "*software*".
Fonte: adaptado de [34].

Inicialmente, o projeto foi estruturado considerando a ferramenta de metodologias ágeis Kanban, sendo que esse método possui uma ferramenta em "*framework*" pré-estabelecida e estruturada para atender demandas no que se refere ao contexto do desenvolvimento dos "*softwares*". O Kanban é uma metodologia de trabalho definida como "puxada", ou seja, as tarefas são realizadas de acordo com a demanda. Além disso, um conceito muito importante no Kanban que corroborou com o foco das entregas e desenvolvimento do "*software*" foi o "*status*" "*work in progress*" (WIP). O WIP limitou o número de tarefas executadas ao mesmo tempo, sendo assim, tratou-se de um importante indicador para o foco das entregas o longo das etapas de desenvolvimento do "*software*". Outro método largamente utilizado no desenvolvimento de "*softwares*" é o Scrum colabora com a estruturação das equipes contempla as experiências, sucessos e falhas ocorridos durante o projeto. Em suma, o Scrum contribuir para que equipes tenham ferramentas consolidadas para guiar as entregas que devem ocorrer dentro dos prazos [13, 15, 16].

Neste trabalho o Kanban [8, 10, 12, 14] foi utilizado porque havia alguns indicadores que possibilitaram utiliza-lo, sendo eles: o tamanho da equipe de desenvolvedores que era pequena; inicialmente apenas uma pessoa era responsável por todos os papéis de desenvolvimento; em alguns momentos o desenvolvimento fora realizado com duas pessoas, no máximo. Não houve um "*backlog*" completo e bem definido. As tarefas foram construídas de acordo com as necessidades do projeto que envolveu o desenvolvimento do "*software*" para o sistema de automação hidropônica.

Para tanto, um quadro foi desenhado contendo quatro colunas: "*backlog*"; "*doing*"; "*quality assurance*"; e "*done*". Na coluna de "*backlog*" as tarefas foram alocadas em ordem de prioridade. Quando alguma tarefa estava sendo executada, o cartão era movido para a coluna ao lado, ou seja, na coluna de "*doing*". Após o desenvolvimento do "*software*" concluído, a tarefa entrava em um período de testes. Posteriormente, a tarefa era enviada à base de código de teste e era disponibilizada em um ambiente para esses testes. Em caso de falha, as correções necessárias eram feitas e uma nova rodada de testes era executada. Em caso de sucesso, o código era enviado à base de código de produção e disponibilizado no ambiente de produção. Apenas quando uma tarefa movida para "*done*" que o time poderia iniciar outra tarefa do "*backlog*" e repetir o ciclo. O Quadro Kanban da Figura 2 ilustra o modelo usado ao longo do desenvolvimento do "*software*".

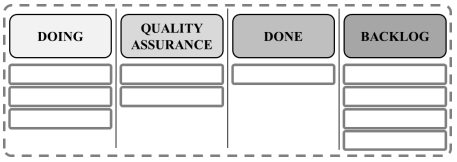

Figura 2. Quadro Kanban elaborado para estruturação do desenvolvimento do "*software*".

Em alguns casos, antes de disponibilizar a tarefa no ambiente de produção, o time de desenvolvimento apresentava a nova funcionalidade aos clientes, ou "*stakeholders*". Com a aprovação deles, o código

era disponibilizado em produção. Caso os clientes não aprovassem, dois caminhos poderiam ser seguidos: criação de novas tarefas no "*backlog*" ou ajuste no código antes de disponibilizar em produção. Nas fases em que o desenvolvimento era realizado por mais de uma pessoa, alguns processos eram realizados, tais como: na "*daily*" cada membro respondia em qual tarefa estava atuando, qual foi o resultado de ontem e qual seria o resultado esperado para o dia. Tal encontro era realizado diariamente, de preferência, nas primeiras horas do dia.

Na etapa denominada "*refining*", normalmente uma vez por semana, o time detalhava as tarefas que estavam em "*backlog*". Nesse momento eram argumentadas as dependências e riscos para uma certa tarefa, além de priorizá-las por ordem de execução, ou seja, as tarefas na parte superior da coluna deveriam ser feitas primeiro. Na retrospectiva, normalmente a cada duas semanas, o time construiu um quadro respondendo quatro perguntas: "o que foi feito de bom?"; "o que pode melhorar?"; "o que parar de fazer?" e também as preocupações eram expostas, assim como trazido pelo "*Project Management Institute*" [8]. Tais processos foram fundamentais para manter a qualidade do time e para realizar os alinhamentos quanto ao andamento e entregas do projeto. Após as pessoas envolvidas responderem às perguntas, o time votava em quais julgava mais importante e as três mais votadas recebiam uma ação. Essa ação deveria ser executada imediatamente. Assim, o time conseguiu corrigir de forma rápida e precisa qualquer problema encontrado no caminho. Nesse processo é fundamental a segurança psicológica das pessoas envolvidas, afinal elas precisam sentir segurança em criticar ou expressar seus sentimentos independente de quem seja atingido por eles.

O projeto detalhado neste trabalho foi desenvolvido para pequenos produtores do Rio Grande do Sul. Suas versões foram colocadas e descritas na plataforma de colaboração e "*open source*" GitHub e encontra-se hospedado na Plataforma Heroku e conta com 145 versões, até o momento. Com base nele foi aplicado o conceito da TRIZ e o resultado obtido também foi analisado [27]. O "*software*" foi desenvolvido a partir de 2019 e foi implantado em Sobradinho, no Rio Grande do Sul. A proposta consistia em criar uma plataforma de usabilidade acessível, capaz de controlar e ajudar o produtor rural no plantio, colheita e manutenção por meio de um sistema automatizado. Portanto, o projeto em questão contou com três pilares: pessoa usuária; aplicação em nuvem; SHA como solução em IoT. O diagrama exemplificativo da topologia da rede com seus dispositivos e interações proposto pelo SHA ser observado pela Figura 3.

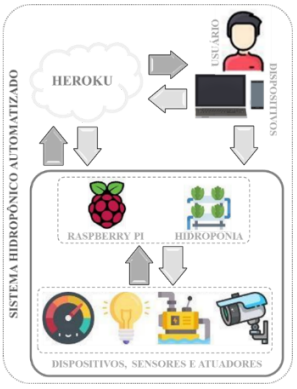

Figura 3. Esquema de interação proposto para o SHA baseado em IoT.

Ainda, o "*software*" contou com algumas funcionalidades básicas para o mínimo funcionamento. Sendo elas: autenticação de pessoa usuária; contagem de atuadores, sensores, câmeras e plantas; exibição de gráficos gerados a partir das medições de sensores; detalhamento de cada planta no SHA; definição da planta como colhida ou morta; bandeira com a definição de cada estado da planta; visualização da câmera para verificação remota. O desenvolvimento de todas as funcionalidades foi apoiado nos conceitos do Kanban, ou seja, cada funcionalidade foi entendida e priorizada em um quadro e o desenvolvimento foi realizado conforme a demanda, chegando a um Mínimo Produto Viável (MVP).

Sendo assim, algumas ferramentas da TRIZ foram utilizadas para aprimorar o MVP. Para compreender e dimensionar o problema a ferramenta escolhida foi a "*Ideality Audit*"; para descobrir todas as necessidades do sistema as ferramentas escolhidas foram "*Ideal System*" e "*Ideal Outcome*"; para definir o problema a ferramenta escolhida foi a "*Smart Little People*"

(SLP); para gerar soluções para o problema a ferramenta escolhida foi a SLP em conjunto com a "*Size Time Cost*" (STC); e por fim, para ordenar as soluções e suas implementações a ferramenta escolhida foi a "*Ideality Plot*". O fluxograma da Figura 4 exemplifica essa estrutura realizada para as entregas associadas ao MVP.

O conceito de idealidade foi fundamental para o entendimento e aplicação da TRIZ. Ele balanceia o desejado e o indesejado em qualquer tipo de sistema. Todas as ferramentas expostas pela TRIZ buscam o aumento da idealidade [19]. Sendo assim, pode-se definir matematicamente tal conceito e torná-lo mais tangível às necessidades do projeto que envolvem o desenvolvimento de "*softwares*". Pela Equação 1 pode-se obter indicadores numéricos para a idealidade considerando a divisão dos benefícios pelo custos somados aos danos. Onde: benefícios representam todos os resultados que se quer na saída do sistema; custos são todas as entradas necessárias para criar o sistema; danos se referem a todos os resultados que não se quer na saída do sistema.

$$Idealidade = \frac{benefícios}{custos+danos} \quad (1)$$

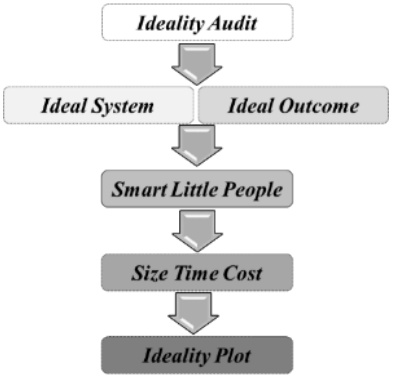

Figura 4. Fluxograma proposto para aplicação da TRIZ.

Vale ressaltar que um sistema na TRIZ é um termo bastante genérico [18-20]. Ele significa qualquer tipo de produto ou processo creado, ou usado, para encontrar necessidade. Ainda, como os custos podem ser considerados dinheiro, tempo, material, conhecimento entre outros. Da mesma forma pode-se idealizar os danos, não sendo necessariamente algo efetivamente prejudicial. O conceito de idealidade não tem apenas importância na TRIZ, mas ele encontra sua importância significativa no desenvolvimento de "*software*" e na aplicação de metodologias ágeis quando, uma vez que ele concebe ações para a solução do problema e seu aperfeiçoamento constante, de uma forma iterativa. A "*Ideality Audit*" necessita de dois elementos principais: o sistema que se tem hoje e o sistema que se quer amanhã. Para resolver o problema do sistema, é necessário, mais que, romper o "*status quo*" do sistema, é necessário rompê-lo em direção ao sistema que se quer amanhã, assim como explanado por [21, 23, 24].

Consequentemente, a ferramenta "*Ideal Outcome*" foi utilizada após a aplicação da "*Ideality Audit*". Com ela foi possível a definição do estado da arte para o sistema em determinado momento do projeto do "*software*". Uma vez defininda, pôde-se delinear quais foram as maiores necessidades e quais os requisitos para o sistema que se quer no futuro. Para a saída ideal houve três tipos de benefícios: o objetivo principal, o benefício primário e os benefícios ordinários. Ou seja, essa característica permitirá a compreensão e dimensionamento do problema sem detalhes irrelevantes e sem propostas de solução inapropriadas. Como o próprio nome sugere, a SLP, consistiu em transformar as necessidades e requisitos levantados com o "*Ideality Audit*" e o "*Ideal Outcome*" em personagens pequenos. Ela foi baseada na empatia e na criação de uma analogia do problema como se fosse uma pessoa. Com a empatia esperava-se que a arquiteta mudasse o prisma em que ela estava inserida e passasse a entender o problema do ponto de vista daquela pessoa (ou daquele problema), conforme explicado por [23, 24].

Neste contexto, novas observações podem ocorrer por meio de novas perspectivas, onde tais constatações podem ser úteis e/ou prejudiciais assim como discutido por [36]. Pode ser prejudicial quando a pessoa arquiteta resistir a resolver um problema se a solução significar que seria dissolvido em ácido, amassado, dissecado etc. Esta amarra com caráter mais psicológica pode ser superada utilizando várias "*Smart Little People*" (SLPs) descartáveis, pela qual não se responsabiliza, com base na premissa de que uma

única apenas morte é uma tragédia. Esses conceitos foram discutidos previamente por [19].

A SLP opera modelando os diferentes aspectos do problema (causas e soluções) com diferentes grupos de SLPs rivais ou complementares. Ainda, equipes rivais de SLP podem ser criadas, onde algumas podem causar problemas, outras podem resolvê-los. Elas fazem o que for necessário, mesmo que isso signifique que serão destruídas. Para gerar soluções para o problema pode-se associar a SLP em conjunto com a STC. Essa é uma ferramenta muito poderosa para estimular novas soluções e superar a inércia psicológica [36].

Afinal, afasta a pessoa arquiteta de soluções atuais e da imagem estreita do problema, e a reconstrói de uma maneira extrema e diferente. A *Size Time Cost*" (STC) considera os resultados que se obteria se o tamanho, tempo e custo fossem extrapolados, para mais e para menos. Esses são seis lugares extremos para procurar soluções; eles efetivamente e instantaneamente ampliam a imaginação para visualizar novas soluções. Por fim, mas não menos importante, a ferramenta "*Ideality Plot*" é uma maneira de categorizar as ideias de acordo com o que elas têm de bom (seus benefícios) e o que há de ruim neles (seus custos e danos). Identificando o nível de benefícios, e então seus custos e prejuízos, colaborando assim com a priorização e a implementação de ideias.

IV. RESULTADOS

O "*software*" foi desenvolvido a partir do protocolo de comunicação entre as unidades controladoras em campo e o servidor em nuvem via "*Hypertext Transfer Protocol*" (HTTP) [1, 2, 28-32]. Ou seja, as unidades controladoras deveriam, constantemente, consultar uma "*Application Programming Interface*" (API) específica para cada funcionalidade, conforme Figura 5. Por essa figura tem-se alguns problemas clássicos conhecidos ao utilizar APIs. O primeiro ponto é que no caso do projeto do SHA analisado, essas APIs devem realizar consultas periódicas, buscando por possíveis alterações para seus atuadores (bombas e luzes, por exemplo) onde, se houvesse alguma atualização respondida pelo servidor em nuvem, ações eram tomadas automaticamente. Ou seja, quando a pessoa usuária desligava uma bomba de água, por exemplo, o servidor em nuvem não informava ativamente o SHA sobre a alteração. Quem realizava essa verificação era a controladora do sistema, sendo que nesse cenário havia desperdício de recursos. A controladora consultava constantemente as APIs, todavia nem sempre recebia uma mensagem para alterar o estado de algum atuador do sistema automatizado.

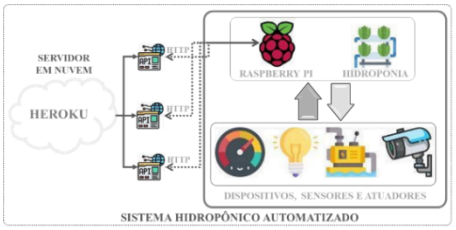

Figura 5. Topologia da rede de comunicação e protocolos utilizados entre o servidor em nuvem e o SHA.

Ainda, foi analisado uma hipótese de não existir um sistema de (re) tentative (s) caso o processamento de uma mensagem falhasse. Ou seja, se a controladora recebesse uma resposta positiva para acionamento de uma bomba, por exemplo, e por algum motivo esse acionamento falhasse, a bomba não seria ativada corretamente. O ponto positivo desse tipo de comunicação foi a simplicidade e a facilidade de implementação. Comunicação via HTTP não depende de aplicações terceiras ou configurações complexas de infraestrutura. Ou seja, esse foi o primeiro elemento necessário para a utilização da ferramenta de "*Ideality Audit*". O próximo elemento definido foi como era o sistema que se desejava no futuro. O sistema que se quer para amanhã pode ser definido como um sistema em que o Servidor em Nuvem e o Sistema Hidropônico atuem de forma independente, de forma assíncrona e que permita a (re) tentativa em caso de falhas.

O próximo passo, visto que a "*Ideality Audit*" foi definida nos parágrafos acima, foi definir o "*Ideal Outcome*" e com eles seus três pilares: o objetivo principal; o benefício primário; e os benefícios ordinários. Como objetivo principal escolheu-se a construção de um sistema resiliente a erros e falhas. Ou seja, em caso de falhas, ambos sistemas deveriam ser capazes de executar uma rotina de (re) tentativa, por exemplo. Como benefício primário

pode-se desejar um sistema assíncrono, ou seja, uma interação entre o servidor em nuvem e o SHA que permita, no limite, a individualidade entre as aplicações. Um benefício ordinário desejado foi que não existisse desperdício de banda na comunicação sem fio construída, ou seja, um sistema que não faça consultas desnecessárias de forma constante. Outro benefício ordinário é a consistência dos dados, ou seja, precisa-se garantir que os dados coletados sejam os mais consistentes e verídicos possíveis. Sendo assim, o diagrama de "Ideal Outcome" é apresentado na Figura 6.

Figura 6. Diagrama de "*Ideal Outcome*" para o SHA.

Posteriormente, o terceiro passo foi seguir com a ferramenta SLP. Nela deve-se abstrair os elementos dos problemas como pessoas. Foram delimitados quatro tipos de indivíduos nos problemas, sendo eles: Processadoras do Servidor (PS); Mensagens (M); Processadoras do Sistema de Controle (PSC); Armazenadoras de Mensagens (AM). Para o problema de consistência de dados o cenário permitia que indivíduos M se percam caso as pessoas PSCs não estejam disponíveis e vice-versa. A Figura 7 demonstra esse tipo problema. Por exemplo, caso os indivíduos PSs enviassem indivíduos Ms para os indivíduos PSCs e estes estivessem dormindo, ou indisponíveis, os indivíduos Ms seriam perdidos e não chegariam aos indivíduos PSCs, a menos que fossem enviados novamente pelas pessoas PSs.

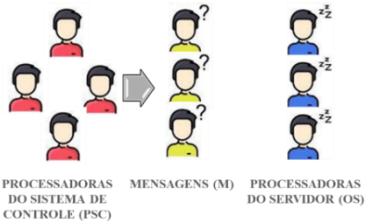

Figura 7. Exemplificação do problema de consistência de dados utilizando "SLP".

Outra situação que pode ocorrer refere-se pode ser observadas pela Figura 8. Essa hipótese consistiu nos indivíduos AMs que possam entrar em ação e fornecer "abrigo" a essas pessoas. Assim, as pessoas PSs não enviaram as pessoas Ms diretamente às pessoas PSCs, mas sim às pessoas AMs, que abrigariam essas pessoas até a disponibilidade das pessoas PSCs. A mesma situação ocorreria quando as pessoas PSCs precisarem informar às pessoas PSs alguma informação que seria exibida à pessoa usuária do sistema: as pessoas MS enviadas pelas pessoas PSCs, seriam abrigadas até que as pessoas PSs pudessem recebê-las.

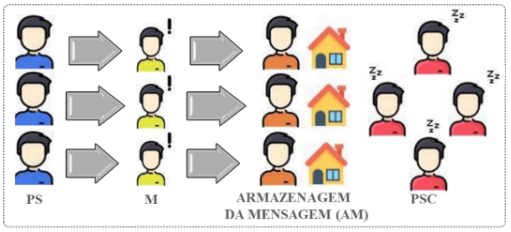

Figura 8. Indivíduos "AMs" abrigando indivíduos "M" enquanto indivíduos "PSC" estavam indisponíveis.

Uma observação a ser ressaltada foi que as pessoas AMs ficaram configuradas e hospedadas em plataformas de alta confiabilidade garantiam ao projeto 99,9% de disponibilidade, assim como proposto por [37]. Com a ferramenta SLP estruturada e organizada, foi feita a implementação da STC. Houve alguns limites práticos impostos pela formulação de ideias e propostas que foram naturais e individuais. Esses limites foram impostos pelo ambiente de aplicação do SHA. Logo, foi importante tentar ir além deles, forçando o pensamento além desse limite inconsciente.

A STC expandiu os conceitos de tamanho, tempo e custo para o problema em questão. Pode-se, nesse caso, extrapolar os conceitos de tamanho e tempo juntos, mas de forma inversa, ou seja, quando se tem alto número de mensagens em um baixo período de tempo. Ou seja, no primeiro cenário (quando indivíduos PSs e indivíduos PSCs enviaram indivíduos Ms diretamente, uns aos outros, no caso do indivíduo receptor daquele indivíduo M estivesse ocupado, ele não poderia receber outro indivíduo (M). Portanto, existiu um gargalo limitante no envio de indivíduos Ms definido pela capacidade de processamento dos

receptores, sejam eles indivíduos PSs ou indivíduos PSCs.

No segundo cenário (quando indivíduos PSs e indivíduos PSCs enviavam indivíduos Ms às pessoas AMs, um problema identificado estava, somente, na capacidade de hospedagem dos indivíduos Ms. Assim, se o indivíduo AM for considerado robusto o suficiente para receber muitos Ms em um baixo intervalo de tempo, a capacidade de processamento de indivíduos PSCs e PSs não seria um problema e poderia ser adequada de acordo com a capacidade financeira do projeto.

Na última etapa da análise a TRIZ houve a proposição da distribuição das soluções discutidas em quatro quadrantes: implementáveis; resolutivas; aprimoráveis; e estagnadas. Essa distribuição foi proposta por [19]. As soluções implementáveis foram aquelas que demonstraram alto benefício e baixos custos e danos. Ou seja, de acordo com a Equação (1), as soluções implementáveis apresentaram maior grau de idealidade. As soluções resolutivas se referem àquelas com alto grau de benefício mas, também, alto grau de custos e danos. Tais soluções tinham perfil de alcance a longo prazo, mas que ainda assim havia necessidade de um esforço mais elaborado. As soluções aprimoráveis foram aquelas com baixo grau de custo e danos, mas baixo grau de benefício. Normalmente, elas eram de fácil implementação, mas sem valor agregado alto. As soluções estagnadas foram aquelas que, pela Equação (1), têm o menor grau de idealidade. Ou seja, elas têm baixo grau de benefício e alto grau de custo e dano. Foram soluções que dificilmente podem ser implementadas. Tais quadrantes estão apresentados na Figura 9.

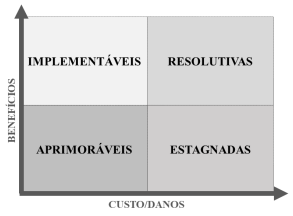

Figura 9. Diagrama da ferramenta "*Ideality Plot*" aplicado ao projeto estudo de caso (SHA).

A solução via mensageria foi mais complexa para o desenvolvimento, afinal, como citado anteriormente, essa solução envolveu a configuração de sistemas externos e, normalmente, em nuvem. Além disso, o desenvolvimento de APIs foi uma tecnologia mais difundida e comum para pessoas que desenvolvem "software". Porém, de acordo com nossa análise de idealidade, os custos e danos que a solução via APIs retornaria seria grande, visto que ela requer um consumo de banda larga maior, por exemplo. Consequentemente, a solução via APIs foi caracterizada como resolutiva, em outras palavras, ela traz benefícios, é de simples desenvolvimento, mas trazia custos e danos importantes ao sistema. Dessarte, considerando as duas soluções discutidas até o momento (via mensageria ou via APIs), tem-se a seguinte disposição no diagrama em quadrantes da Figura 10.

Figura 10. Diagrama da ferramenta "*Ideality Plot*" com as soluções discutidas.

V. DISCUSSÃO

Apesar da TRIZ ter sido inicialmente para projetos de engenharia mecânica [18, 23, 24], os princípios inventivos dela podem ser aplicados ao desenvolvimento do "*software*" [19-22], como o estudo de caso do SHA deste trabalho. Entretanto, salienta-se que é necessário que a pessoa que irá aplicar a ferramenta tenha sólido conhecimento na área trabalhada, afinal, a TRIZ oferece conceitos e ideias caracterizadas para que as pessoas envolvidas possam escolher a que mais faz sentido para o projeto naquele momento. Ou seja, a TRIZ não será factível se troca de mensagens for feita por mensageiras [38], sendo mais recomendada, para o presente estudo, a comunicação via API. Sendo assim, o conceito de uma comunicação assíncrona é mais aplicável do que a de comunicação síncrona para o desenvolvimento do "*software*" aqui estudado. Ainda, cabe às pessoas observadoras saber e escolher quais ferramentas podem ser utilizadas ao longo do projeto.

O Quadro Kanban oferece recursos visuais e organizacionais para que o projeto do "*software*" seja particionado devidamente para que as entregas sejam efetuadas de forma competente e ágil. A interação entre as equipes também é valorizado e a orquestração das reuniões é fundamental para que o projeto tenha sucesso. No contexto da engenharia de "*software*", no processo de desenvolvimento ágil é importante pensar em entregas contínuas e incrementais que garantem alto valor ao cliente. Portanto, desenvolver o "*software*" em questão evoluindo-o para utilizar mensageria não significa que a escolha do passado é errada, mas sim que a escolha do passado não é mais suficiente para o presente [10-14].

No processo de desenvolvimento de "*software*", as demandas do usuário são refletidas primeiramente na qualidade do "*software*". A qualidade pode escolher o padrão de uso apropriado, a fim de tornar o processo, desde os requisitos do cliente até os requisitos de design, as características técnicas das peças e, em seguida, a operação técnica aos requisitos de produção. Nesse processo haverá fenômenos ruins, como conflitos e problemas, que não podem ser resolvidos, e neste momento é necessário combinar o desenvolvimento de software ou processo de qualidade de software com organicamente [33].

No que tange ao projeto do SHA, quanto ao problema de uso da banda houve o seguinte cenário: as pessoas PSCs, a cada dez segundos enviam indivíduos Ms às pessoas PSs, solicitando informações sobre um periférico específico. Ao mesmo tempo, as pessoas PSs enviam indivíduos Ms para as pessoas PSCs solicitando informações. Porém, na grande maioria das vezes, o envio é desnecessário, ou porque o indivíduo M terá o mesmo conteúdo de dez segundos atrás ou porque o indivíduo M não terá informação alguma disponível. Sendo assim, se os papéis se inverterem e as mensagens forem enviadas por quem tem a informação para ser disponibilizada e não por quem quer saber a informação, pode-se mitigar o envio de indivíduos Ms desnecessários.

Quanto ao problema de consistência de dados e (re) tentativas é comum bibliotecas de mensageiras que conseguem realizar o controle de recebimento e disponibilização dessas mensagens. Ou seja, se um indivíduo M está hospedado em um indivíduo AM, o hospedeiro tem a capacidade de avaliar se aquele indivíduo M foi consumido corretamente por um indivíduo consumidor e hospedá-lo por mais tempo caso o consumo apresentar algum erro. Algumas ferramentas podem ser utilizadas como indivíduos AMs, entre elas RabbitMQ e Kafka. O Kafka é uma plataforma de controle de eventos distribuídos, desenvolvido em *Java* e Scala, que permite a publicação e assinatura desses eventos. Esses eventos permanecem até que o consumidor as leia ou atinja seu limite de retenção. Ele, também, utiliza uma abordagem baseada em "*pull*", permitindo que as pessoas consumidoras solicitem lotes de mensagens. O RabbitMQ é um agente de mensagens distribuído que facilita a entrega eficiente de mensagens em cenários de roteamento complexos. Ele utiliza o modelo "*push*". A analogia com uma agência postal, que recebe, armazena e entrega correspondências é válida uma vez que o RabbitMQ aceita, armazena e transmite mensagens de dados binários [38, 39].

VI. CONCLUSÃO

O Kanban foi utilizado como recurso para alocação das equipes a partir de "*tickets*", ou requisições corriqueiras, que comumente ocorrem ao longo do desenvolvimento de "*softwares*". Com os passos da TRIZ trabalhados e analisando o resultado obtido ao longo do projeto do SHA percebe-se que ambas as soluções para o problema funcionaram e foram viáveis (vide Figura 9) e aplicáveis ao desenvolvimento do "*software*". Porém, uma foi caracterizada com aspectos implementáveis e a outra com perspectivas resolutivas. A solução implementada inicialmente foi caracterizada como resolutiva e pôde-se considerar como principal fator para a escolha da TRIZ o objetivo da construção do SHA estruturando o projeto direcionado a um MVP. Portanto, a tecnologia de mais fácil implementação foi escolhida, sem considerar que ela teria um custo alto no futuro devido às funcionalidades que seriam necessárias e determinadas no início do projeto. No contexto das soluções IoT, os "*softwares*" tendem a se tornar cada vez mais robustos e resolutivos, uma vez que plataformas "*open-source*" de compartilhamento de algoritmos, comandos lógicos e conhecimentos

computacionais continuam se popularizando em uma indústria em crescente ascenção. Técnicas, ferramentas, metodologias, entre outros recursos de gerenciamento são fundamentais para o segmento das tecnologias contemporâneas corroborando com a melhor estruturação das equipes de desenvolvimento e também atingindo as entregas dentro do prazo.

Referências

[1] LEE, I. The Internet of Things for enterprises: An ecosystem, architecture, and IoT service business model. Internet of Things, v. 7, n. 100078, 2019.

[2] OLIVEIRA, José Miguel Rocha Valente. Desenvolvimento de um sistema IoT com comunicação via App/Cloud para monitorização de uma cama médica. 87 p. 2020. Dissertação de Mestrado. Mestrado Integrado em Engenharia Eletrotécnica e de Computadore. Faculdade de Engenhari. Universidade do Porto. 2020.

[3] SANTOS, B. P.; SILVA, L. A. M.; CELES, C. S. F. S.; NETO, J. B. B.; PERES, B. S.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; GOUSSEVSKAIA, O. N.; LOUREIRO, A. A. F. Internet das Coisas: da Teoria à Prática. Disponível em <https://homepages.dcc.ufmg.br/~mmvieira/cc/papers/internet-das-coisas.pdf>. Acesso em 20 jul. 2022.

[4] NAGAJAYANTHI, B. Decades of Internet of Things towards twenty‑first century: a research‑based introspective. Wireless Personal Communications, Springer Publication, v. 123, p. 3661-3697.

[5] OLIVEIRA, R. L. F.; PEDRON, C. D. Métodos Ágeis: Uma revisão sistemática sobre benefícios e limitações. Brazilian Journal of Development, v.7, n.1, p. 4520-4535, 2021.

[6] INSTITUTO BRASILEIRO DE GEOGRAGIA E ESTATÍSTICA. Pesquisa de Serviços de Tecnologia da Informação. Ministério do Planejamento, Orçamento e Gestão. 2009. Disponível em <https://biblioteca.ibge.gov.br/visualizacao/livros/liv49099.pdf> . Acesso em 21 jul. 2022.

[7] PETRY, Larissa Todt. Fatores de disseminação de métodos ágeis em empresas de TI. 73 p. 2018. Trabalho de Conclusão de Curso. Departamento de Ciências Administrativas. Universidade Federal do Rio Grande do Sul. 2018.

[8] Project Management Institute. 2021. Padrão de gerenciamento de projetos e Guia do conhecimento em gerenciamento de projetos (Guia PMBOK). 7ª ed. Estados Unidos. RabbitMQ. 2022. Disponível em <https://www.rabbitmq.com/>. Acesso em 17 jul. 2022.

[9] SCATOLINO, A. R.; CAMILO, R. D. Influência da aplicação de métodos ágeis e da gestão do conhecimento na qualidade de software: uma análise multivariada. Revista de Gestão e Projetos, v. 10, n. 3, p. 65-80, 2019.

[10] COUTO, J. M. C. Métodos ágeis e PMBOK: uma revisão sistemática da literatura sobre o uso de abordagens híbridas no gerenciamento de projetos de software. Conference: Estacio, MBA Gestão de Projetos. 2016. Disponível em < https://www.researchgate.net/publication/308557793_Metodos_Ageis_PMBoK_Uma_Revisao_Sistematica_da_Literatura_sobre_o_uso_de_Abordagens_Hibridas_no_Gerenciamento_de_Projetos_de_Software>. Acesso em 28 jul. 2022.

[11] COLLABNET VERSION ONE. 13th Annual State of Agile Report. Disponivel em <https://www.duxdiligens.com/wp-content/uploads/2019/09/13th-annual-state-of-agile-report_7_May_2019.pdf>. Acesso em 25 jul. 2022.

[12] AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: 39th Software Engineering and Advanced Applications (SEAA/EUROMICRO), p. 9-16, 2013.

[13] PERMANA, P. A. G. Scrum method implementation in a software development project management. International Journal of Advanced Computer Science and Applications, v. 6, n. 9, p. 198-204, 2015.

[14] SANTOS, P. S. M.; BELTRÃO, A. C.; SOUZA, B. P.; TRAVASSOS, G. H. On the benefits and challenges of using Kanban in software engineering: a structured synthesis study. . Journal of Software Engineering Research and Development, v. 6, n. 13, 2018.

[15] HEMA, V.; THOTA, S.; KUMAR, S. N.; PADMAJA, C.; KRISHNA, R.; MAHENDER, K. Scrum: an effective software development agile tool. IOP Conf. Series: Materials Science and Engineering, v. 981, n. 22060, 2020.

[16] BAXTER, D.; TURNER, N. Why Scrum works in new product development: the role of social capital in managing complexity. Production Planning & Control. DOI: 10.1080/09537287.2021.1997291.

[17] PIRES, J. G. C. Metodologia TRIZ uma opção para solução de problemas orientada ao ser humano e estruturada para inovação. In: XI Simpósio de Excelência em Gestão e Tecnologia. Evento online. Disponível em <https://www.aedb.br/seget/arquivos/artigos14/2720384.pdf>. Acesso em 11 jul. 2022.

[18] PIMENTEL, A. R. Considerações sobre TRIZ e sua aplicação no desenvolvimento de software. Disponível em <https://www.inf.ufpr.br/andrey/publicacoes/trizartigo.pdf>. Acesso em 11 jul. 2022.

[19] HAINES-GADD, L. TRIZ for dummies. Wiley, United States. 2016.

[20] KLUENDER, D. TRIZ for software architecture. Procedia Engineering, v. 9, p. 708–713, 2011.

[21] GOVINDARAJAN, U. H.; SHEU, D. D.; MANN, D. Review of Systematic Software Innovation Using TRIZ. International Journal of Systematic Innovation, v. 3, n. 5, p. 72-90, 2019.

[22] KRASNOSLOBODTSEV, V.; LANGEVIN, R. Applied TRIZ in high-tech industry. Technical Innovation Center Inc., 100 Barber Avenue, Worcester, MA 01606 USA. Disponível em <https://new.aitriz.org/articles/TRIZFeatures/6B7261736E6F736C6F626F64747365762D31323033.pdf>. Acesso em 08 jul. 2022.

[23] MANN. D. Application of TRIZ tools in a non-technical problem context. Systematic Innovation, 5A Yeo-Bank Business Park Kenn Road, Clevedon BS21 6UW, UK. 2000. Disponível em < http://systematic-innovation.com/assets/200008-applicationoftriztoolsinanon-technicalproblemcontext.pdf>. Acesso em 10 jul. 2022.

[24] MANN, Darrel. 2004. TRIZ for software? TRIZ Journal. Disponível em http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.125.1809&rep=rep1&type=pdf. Acesso em 29 jul. 2022.

[25] RICHARDS, M.; Ford, N. 2020. Fundamentals of Software Architecture: An Engineering Approach. 1ed. O'Reilly. United States of America.
[26] NAKAGAWA, T. 2005. Software engineering and TRIZ (1) structured programming reviewed with triz. Osaka Gakuin University, Japão. Disponível em <https://www.osakagu.ac.jp/php/nakagawa/TRIZ/eTRIZ/epapers/e2005Papers/eNakaTRIZCONSE1/eTRIZCON2005-SE-050604.pdf>. Acesso em 29 jul. 2022.
[27] PISSARDO, R. A. 2019. Pissardo Saúde e Tecnologia - Plataforma de Controle Hidropônico. Disponível em https://github.com/rafael-pissardo/pissardo-saude-tecnologia. Acesso em 02 jul. 2022.
[28] ROCCIA, C.; TERUEL, B.; ALVES, E.C. S.; ARNOLD, F.; BRAVO-ROGER, L.; MORETTI, A.; GONÇALVES, M.S. Experimental evaluation of the performance of a wireless sensor network in agricultural environments. Engenharia Agrícola, Jaboticabal, v. 32, n. 6, p. 1165-1175, 2012.
[29] WORLD ECONOMIC FORUM. State of the Connected World 2020 Edition – In collaboration with the Global Internet of Things Council and PwC. Disponível em < https://www3.weforum.org/docs/WEF_The_State_of_the_Connected_World_2020.pdf>. Acesso em 29 jul. 2022.
[30] SONI, H.; KAHAR, S. Design IoT based Smart Irrigation System using Arduino. International Journal of Advanced Research in Computer and Communication Engineering, v. 9, n. 3, 89-94, 2020.
[31] BAVKAR, S.; PATIL, N.; BIRJE, Y. IoT Enabled Smart Irrigation System Using Arduino. In: II International Conference on Communication and Information Processing. 2020. Disponível em <https://ssrn.com/abstract=3648829 or http://dx.doi.org/10.2139/ssrn.3648829>. Acesso em 28 jul. 2022.
[32] VENDEMIATTI, Caíque. Sistema remoto para monitoramento do consumo de água em tempo real. 111 p. 2020. Dissertação de Mestrado. Instituto de Ciência e Tecnologia, Universidade Estadual Paulista. Sorocaba. 2020.
[33] WANG, S.; SAMADHIYA, D.; CHEN, D. Software Development and Quality Problems and Solutions by TRIZ. Procedia Computer Science, v. 5, p. 730-735. 2011. Disponível em <https://www.sciencedirect.com/science/article/pii/S1877050911004212>. Acesso em 29 set. 2021.
[34] About – Cynefin Framework. 2022. The Cynefin Co. Disponível em < https://thecynefin.co/about-us/about-cynefin-framework>. Acesso em 13 jul. 2022.
[35] LOCKE, John. Ensaio Sobre o Entendimento Humano. 1ª ed. Martins Fontes. Brasil. 2012.
[36] GADD, K.; GODDARD, C. TRIZ for Engineers: Enabling Inventive Problem Solving. 1ª ed, John Wiley & Sons, United Kingdom. 2011.
[37] AMAZON. 2022. Amazon MKS Faqs. Disponível em <https://www.amazonaws.cn/en/msk/faqs/>. Acesso em 13 jul. 2022.
[38] KAFKA. 2022. Disponível em <https://kafka.apache.org/>. Acesso em 15 jul. 2022.
[39] RABBITMQ. 2022. Disponível em <https://www.rabbitmq.com/>. Acesso em 16 jul. 2022.