DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Técnicas de teste e simulação automatizada de software voltadas a sistemas embarcados

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Rafael Figueiró Berto

Florianópolis, julho de 2019

Técnicas de teste e simulação automatizada de software voltadas a sistemas embarcados

Rafael Figueiró Berto

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Rômulo Silva de Oliveira

Banca Examinadora:

Luiz Felipe Raupp
Orientador na Empresa

Prof. Rômulo Silva de Oliveira
Orientador no Curso

Prof. Hector Bessa Silveira
Responsável pela disciplina

Prof. Julio Elias Normey Rico
Responsável pela disciplina

Prof. Ricardo José Rabelo
Responsável pela disciplina

Rodrigo Tacla Saad, Avaliador

Bruno Marcon Bez Batti, Debatedor

Vanessa Souza Vilela, Debatedora

Agradecimentos

Agradeço aos meus pais e ao meu irmão pelo auxílio e encorajamento durante todas as etapas de minha formação.

Ao Vinicius por estar sempre apoiando, auxiliando e me aturando durante esta fase e, espero, em muitas que ainda virão.

À *Hexagon Agriculture* por abrir suas portas para o desenvolvimento deste trabalho e para aquisição de conhecimentos. Em especial agradeço ao meu orientador, Raupp, e ao restante do time D, sempre dispostos a discutir e auxiliar na solução dos problemas.

A UFSC e ao curso de Engenharia de Controle e Automação por propiciarem minha formação acadêmica e me auxiliarem a chegar neste ponto. Aos muitos professores que tive durante minha vida, muito obrigado por seus ensinamentos.

A minha tia Ani por me auxiliar e incentivar durante a escrita deste trabalho e em diversos momentos da vida acadêmica.

E aos demais amigos e familiares que sempre estiveram presentes e ficam alegres por minhas conquistas.

Muito obrigado.

Resumo

A *Hexagon Agriculture* é uma empresa que desenvolve soluções para digitalização da agricultura, combinando tecnologias em *hardware*, *software* e *know-how* para converter dados em informações inteligentes e acionáveis e trazendo soluções para cultivo, colheita e *Original Equipment Manufacturer* (OEM). No cotidiano do desenvolvimento é necessária a validação de cada funcionalidade antes dela ser entregue ao cliente, assim, evitando danos a imagem da empresa e do produto, para isso são realizados diversos testes do *software* e do *hardware* produzidos. A realização destes testes ocorre em duas etapas: testes unitários, realizados pelos desenvolvedores, e testes de usabilidade, realizados pela equipe de garantia de qualidade (QA). Os testes realizados pelo QA possibilitam uma visão do produto funcionando como um todo, porém exigem uma elevada quantidade de maquinário, espaço físico, mão-de-obra e tempo. Buscando localizar problemas relacionados ao desenvolvimento, de forma mais rápida e sem a necessidade da elevada quantidade de recursos, este trabalho apresenta o desenvolvimento de técnicas de simulação e teste de *software* automatizada, tendo como objetivo realizar testes e simulações diariamente, ainda na fase de desenvolvimento. Com o auxílio destas ferramentas se espera a melhora na qualidade dos produtos, redução do tempo gasto em testes e na solução de problemas.

Palavras-chave: agricultura, testes de *software*, testes automatizados, simulação.

Abstract

Hexagon Agriculture is a company that develops solutions for the agriculture digitization, combining hardware, software and know-how technologies to convert data into intelligent, actionable information and bringing cultivation, harvesting and Original Equipment Manufacturer (OEM) solutions. In the daily development of new features is necessary to validate each functionality before it is delivered to the client, thus avoiding damage to the company and product image, for this are performed several tests of the software and hardware produced. These tests are performed in two stages: unit tests performed by the developers and usability tests performed by the quality assurance team (QA). QA testing enables a view of the product as a whole, but requires a large amount of machinery, space, labor and time. In order to find problems related to development, in a faster way and without the need of a large amount of resources, this project presents the development of automated software simulation and testing techniques, with the objective of performing tests and simulations daily, still in the development phase. These tools are expected to improve product quality, reduce time spent on testing and problem solving.

Keywords: agriculture, software testing, automated testing, simulation.

Lista de ilustrações

Figura 1 – Processo sistêmico - P&D. 23
Figura 2 – Principio de simulações *hardware in the loop*. 29
Figura 3 – Modelo-Visão-Controlador (MVC). 35
Figura 4 – Esquema de variáveis do piloto automático. 38
Figura 5 – Diagrama da geração do pacote de atualização. 40
Figura 6 – Simulação do piloto automático - caminho 1. 42
Figura 7 – Simulação do piloto automático - caminho 2. 42
Figura 8 – Simulação do piloto automático - caminho 3. 43
Figura 9 – Diagrama da simulação do piloto automático. 45
Figura 10 – Diagrama dos testes de comportamento. 48
Figura 11 – Diagrama da primeira versão da ferramenta. 49
Figura 12 – Diagrama idealizado para a ferramenta. 50
Figura 13 – Diagrama da ferramenta implementada. 51
Figura 14 – Ferramenta para execução da simulação do piloto automático. 53
Figura 15 – Ferramenta para execução dos testes de comportamento. 54
Figura 16 – Relação de tipo de tarefa por versão. 55

Lista de abreviaturas e siglas

BDD	Desenvolvimento Orientado a Comportamento (do inglês *Behavior-Driven Development*)
CAN	do inglês *Controller Area Network*
D-Bus	do inglês *Desktop Bus*
GNSS	Sistemas Globais de Navegação por Satélite (do inglês *Global Navigation Satellite System*)
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
MVC	Modelo-Visão-Controlador (do inglês *Model-View-Controller*)
OEM	Fabricantes de equipamentos (do inglês *Original Equipment Manufacturer*)
P&D	Pesquisa e Desenvolvimento
PPM	Gerenciamento de Portfólio de Projetos (do inglês *Project Portfolio Management*)
QA	Garantia de Qualidade (do inglês *Quality Assurance*)

Sumário

1 INTRODUÇÃO 15
1.1 Plano de atividades e cronograma 16
1.2 Relação do trabalho com o curso de Engenharia de Controle e Automação 17
1.3 Estrutura do relatório 18

2 A HEXAGON 19
2.1 A Hexagon Agriculture 20
2.1.1 O setor de desenvolvimento embarcado 21

3 O PROJETO 23
3.1 Objetivos 25
3.2 Metodologia 25
3.3 Fases do projeto 26

4 TESTES E SIMULAÇÕES 27
4.1 Teste de *software* 27
4.1.1 Testes de unidade 27
4.1.2 Testes de integração 28
4.1.3 Testes de sistema 28
4.1.4 Testes de aceitação 28
4.2 Simulações *hardware in the loop* 29
4.3 Ferramentas utilizadas 30
4.3.1 *Cucumber* 30
4.3.2 V-Rep 30
4.3.3 GoCD 31

5 TÉCNICAS DESENVOLVIDAS 33
5.1 Atualização das ferramentas de simulação existentes 33
5.1.1 *Simulation Gateway* - Gerenciador de dados da simulação 33
5.1.2 V-Rep *Scene Manager* - gerenciador de cenas 34
5.1.3 *Kilgrave* - reprodutor de *logs* de campo 35
5.1.4 *Morpheus* - gerenciador de caminhos 36
5.1.5 *Logparser* - *scripts* de averiguação do resultado da simulação 37
5.2 Novo pacote de atualização do sistema 39
5.3 Instalação do pacote de atualização 40

5.4 Simulação automatizada do piloto automático 41
5.4.1 Resultados de simulação 44
5.5 Testes de comportamento automatizados 47
5.6 Ferramenta obtida 48
5.6.1 Solução implementada 51

6 ANÁLISE DOS RESULTADOS 53
6.1 Resultados esperados 54
6.2 Trabalhos futuros 54

7 CONSIDERAÇÕES FINAIS 57

REFERÊNCIAS 59

ANEXOS 63

ANEXO A – TITANIUM. 65

ANEXO B – EXEMPLO DE RESULTADO DOS TESTES DE COMPORTAMENTO. 67

1 Introdução

No cotidiano de um empresa de tecnologia são diariamente produzidas novas funcionalidades que visam a melhoria ou a criação de produtos. Estes desenvolvimentos podem ser fruto de pedidos da clientela, melhorias no produto ou correção de problemas. À medida que são realizadas alterações se faz necessário um grande cuidado para que não se adicionem problemas ou se façam alterações indesejadas nas funcionalidades já existentes. Para isso podem ser empregadas técnicas que trazem maior qualidade, auxiliando também no dia a dia do desenvolvimento de produtos. Estas técnicas, em grande parte, envolvem a realização de diversos tipos de testes.

A qualidade do produto é essencial para a visibilidade da empresa perante a seus clientes, um produto que apresenta grande estabilidade e funcionalidade terá maior confiança no mercado. Outro problema que falhas em produtos pode causar é o elevado custo para sua solução, visto que pode envolver diversas etapas e mão de obra especializada. O tempo decorrido até um problema ser encontrado também é uma característica relevante, pois logo que é detectado o desenvolvedor responsável tem informações do processo, de maneira que viabilizam a correção.

Para melhorar esse cenário, podem ser utilizadas técnicas para teste e simulação do *software* desenvolvido. No caso dos testes existem diversas formas de se testar um sistema que vão desde testes de unidade a testes de aceitação. No que diz respeito a simulação, ela pode servir de forma a tentar reproduzir, da forma mais fiel possível, o comportamento real, ou seja, o comportamento que o produto terá nas mãos do cliente.

Na *Hexagon Agriculture*, empresa onde este trabalho foi desenvolvido, se desenvolvem soluções tecnológicas voltadas à digitalização da agricultura. Os produtos produzidos pela empresa se encontram em diversos segmentos da agricultura, com clientes espalhados por diversos países. Dentre as soluções propostas pela empresa estão produtos como piloto automático para máquinas agrícolas, controle de implementos, rastreamento de matéria prima, entre outras [1].

Na produção destas soluções, existe uma grande dificuldade na realização de testes e validações das mudanças e melhorias realizadas. Isso decorre da natureza do produto, que requer um elevado espaço físico para realização de testes, uma elevada quantidade de maquinário agrícola e, ainda, um elevado consumo de tempo por parte dos desenvolvedores e da equipe de garantia de qualidade (QA, do inglês *quality assurance*).

Atualmente os padrões de desenvolvimento da empresa, tem como prática a realização de testes unitários em todos os produtos desenvolvidos. Porém, os testes unitários apenas garantem o comportamento de cada parte isolada do produto e não a integração

entre os diversos sistemas presentes e da sua interação com o ambiente. Isso gera uma grande limitação devido ao fato de que as diversas ferramentas e softwares envolvidos comunicam-se entre si, e, pequenas alterações podem quebrar serviços integrados, sendo eles próprios ou de terceiros.

Estas condições contribuem para que muitos problemas sejam apenas detectados pelo setor de QA, ou, até mesmo, pelos clientes finais. Os problemas detectados, tanto na empresa quanto pelo cliente, geram atraso no desenvolvimento de novas funcionalidades e pressão por parte de clientes, acarretando efeitos sobre os objetivos da empresa e em seu compromisso com a clientela.

Com o objetivo de facilitar os testes e validações dos produtos e atenuar estes problemas são utilizados algumas técnicas de testes integrados e simulação dos produtos. Este trabalho traz um estudo sobre a atualização e adaptação destas técnicas, através da automação do processo de testes a partir de ferramentas que permitam executá-los de forma simples por um usuário e que também sejam executadas diariamente sem a necessidade de interação.

É esperado que os problemas possam ser identificados mais cedo, fazendo com que sejam resolvidos de forma mais eficaz, rápida e com menores custos. Acredita-se que haja melhora na qualidade dos produtos, auxiliando os desenvolvedores, diminuindo o tempo de testes em campo e o uso de maquinário, facilitando a preparação de cenários adversos, como terrenos acidentados, mal funcionamento de sensores, entre outros.

1.1 Plano de atividades e cronograma

Para o desenvolvimento completo deste trabalho, diversas etapas foram necessárias, as atividades que fizeram parte do plano de atividades foram:

1. Atualização de ferramentas proprietárias.
    a) Gerenciador de dados da simulação, conhecido como *Simulation Gateway*;
    b) Gerenciador de cenas de simulação conhecido como *V-Rep Scene Manager*;
    c) Reprodutor de *logs* de campo, conhecido como *Kilgrave*;
    d) Gerenciador de caminhos, conhecido como *Morpheus*.
2. Atualização de *scripts* de averiguação de resultados de simulação.
3. Estudo do *framework* de testes *Cucumber*.
4. Estudo da ferramenta de integração *GoCD*.
5. Automação da simulação do piloto automático.

6. Automação dos testes de comportamento.
7. Automação da criação do pacote de atualização dos produtos.
8. Expansão das soluções para os demais produtos da empresa (trabalhos futuros).

Cada um dos itens será abordado detalhadamente ao decorrer do devolvimento deste documento. As tarefas foram dispostas de acordo com o calendário da empresa, apresentado no quadro 1.1, durante o período do projeto, resultando no cronograma do quadro 1.2.

Sprint	Data de início	Data de fim
01	08/01/2019	18/01/2019
02	21/01/2019	08/02/2019
03	11/02/2019	22/02/2019
04	25/02/2019	08/03/2019
05	11/03/2019	22/03/2019
06	25/03/2019	05/04/2019
07	08/04/2019	19/04/2019
08	22/04/2019	03/05/2019
09	06/05/2019	17/05/2019
10	20/05/2019	31/05/2019
11	03/06/2019	14/06/2019
12	17/06/2019	28/06/2019

Quadro 1.1: Calendário de *sprints*.

Sprint / Tarefa	01	02	03	04	05	06	07	08	09	10	11	12
1a	■	■										
1b		■	■									
1c				■	■							
1d						■						
2							■					
3					■	■						
4					■	■						
5						■	■			■		■
6									■	■	■	■
7								■	■	■	■	■

Quadro 1.2: Cronograma de atividades.

1.2 Relação do trabalho com o curso de Engenharia de Controle e Automação

Durante o desenvolvimento do trabalho diversas áreas do conhecimento foram necessárias para atingir o objetivo final, em sua maioria tendo relação com o curso de

Engenharia de Controle e Automação. De maneira geral, o sistema que a empresa desenvolve necessita de conhecimento nas áreas de sistemas de controle e instrumentação, por envolver o uso de diversos sensores e diferentes formas de atuação que devem ser englobadas nos algoritmos de controle presentes e, por consequência, devem fazer parte dos testes e simulações desenvolvidas. Durante o processo de desenvolvimento do trabalho também foram necessários conhecimentos nas áreas de modelagem de *software*, testes unitários e de integração e um vasto conhecimento nas áreas de desenvolvimento de *software*, abordadas em diferentes disciplinas no decorrer do curso. Outro ponto que necessita atenção é a necessidade de comunicação entre diversos sistemas, sendo eles proprietários ou de terceiros, assim sendo de suma importância conhecimentos na área de integração de sistemas.

1.3 Estrutura do relatório

O restante deste documento está organizado em seis capítulos. No capítulo 2 é apresentado a empresa onde o projeto foi desenvolvido, abordando sua área de atuação e como e onde o projeto se encaixa dentro dela. No capítulo 3 são discutidos os problemas e as propostas de solução, bem como os requisitos do projeto. Em sequência, no capítulo 4, são discutidos as bases teóricas do projeto, que envolvem os principais conceitos utilizados para testes e simulações de *software*. No capítulo 5 é apresentado o desenvolvimento do projeto, descrevendo cada uma das etapas realizadas. Por fim, nos capítulos 6 e 7, são apresentados os resultados e os trabalhos futuros e as considerações finais, respectivamente.

2 A Hexagon

A *Hexagon* foi fundada na Suécia em 1992 e atualmente possui mais de 20.000 funcionários espalhados por 50 países [2]. A empresa é líder em soluções de sensores, *software* e sistemas autônomos, tendo como objetivo utilizar dados para fortalecer um futuro autônomo, trabalhando para que seus clientes possuam meios para prosperar e crescer de forma eficiente, produtiva e com qualidade [3].

As principais soluções desenvolvidas são sensores de posicionamento, sensores de captura de realidade, recursos de design, simulação e recursos de mapeamento de inteligência de localização e sistemas autônomos voltadas a diversas áreas, tais como carros, veículos industriais, trens e outros [3].

Esses produtos estão separados em oito divisões, são estas: Geoespacial, Geossistemas, Inteligência de manufatura, Mineração, Inteligência de posicionamento, Gerenciamento de Portfólio de Projetos (PPM, do inglês *Project Portfolio Management*), Segurança e infraestrutura e Agricultura [4].

A divisão Geoespacial trabalha no desenvolvimento de tecnologias que visam exibir e interpretar dados geográficos baseado na localização de informações de negócios de forma dinâmica e utilizável [5]. Já a divisão de Geossistemas traz um portfólio de soluções geoespaciais que auxiliam em diversos setores do mercado na captura da realidade, gerando informações digitais que trazem benefícios no entendimento, planejamento e na execução de tarefas [6].

A divisão de Inteligência de manufatura trabalha com soluções de metrologia e de manufatura, através da expertise em detecção, inteligência e atuação na coleta, análise e uso ativo de dados de medição [7]. Na divisão de Mineração são desenvolvidas soluções para integrar, automatizar e otimizar fluxos de trabalho de forma que se obtenham vantagens competitivas e com redução de custos visando aumentar a precisão, exatidão e segurança dos processos de mineração [8]. Por sua vez, a divisão de Inteligência de posicionamento desenvolve soluções de Sistemas Globais de Navegação por Satélite (GNSS, do inglês *Global Navigation Satellite System*) que permitem identificar o posicionamento de objetos, na terra, ar ou água, com uma maior exatidão [9].

Soluções de *softwares* empresariais são encontradas na divisão de PPM, com estes sistemas se busca transformar dados desorganizados em dados inteligentes e úteis que permitam um *design* e operação mais inteligentes de plantas, navios e instalações *offshore* [10]. A divisão de Segurança e infraestrutura busca ajudar as organizações de segurança a possuir os dados certos na hora certa, de forma a gerenciar as mudanças dos dados de missões e negócios de maneira mais inteligente e eficaz [11].

Por fim, a divisão de agricultura, onde o trabalho foi desenvolvido, busca fornecer tecnologias inovadoras através de soluções inteligentes com o objetivo de aumentar a eficiência e produtividade no campo [12]. No restante do capítulo será abordado um pouco mais sobre esta divisão.

2.1 A Hexagon Agriculture

A Hexagon Agriculture foi fundada em 2014 pela junção de três empresas com portfólios voltados ao mercado agrícola: Arvus Tecnologia, ILab sistemas e a vertical de agricultura da *Hexagon Geosystems*. Buscando uma transformação digital do campo, foram exploradas as tecnologias desenvolvidas por essas empresas para automatizar operações e aumentar a produtividade agrícola em todo o mundo [13].

A empresa combina tecnologias em *hardware*, *software* e *know-how* para converter dados em informações inteligentes e acionáveis que permitam um planejamento inteligente, uma execução eficiente no campo e o controle preciso de máquinas e fluxos de trabalho [13].

As soluções apresentadas pela empresa visam permitir a gestão, monitoramento e otimização no cultivo, colheita e transporte. Para isso as soluções são divididas em três portfólios: *AgrOn Cultivation*, *AgrOn Harvesting* e *AgrOn OEM* [1].

O *AgrOn Cultivation* traz soluções que permitem a gestão de cultivos de forma a otimizar a eficiência de recursos para um aumento da produção com redução dos insumos e aumento da qualidade. Baseado em soluções modulares os produtos se dividem nas áreas de Planejamento & Otimização, Monitoramento & Gestão de Operações e Automação de Máquinas [14].

Para otimização de operações colheita e transporte são oferecidas as soluções do *AgrOn Harvesting*, onde da mesma forma que o *AgrOn Cultivation* os produtos se dividem nas áreas de Planejamento & Otimização, Monitoramento & Gestão de Operações e Automação de Máquinas. Com soluções modulares para empresas agrícolas e florestais, este portfólio apresenta produtos para otimização da eficiência dos recursos, redução de perdas de matéria prima, sincronização das atividades, garantia qualidade e lucratividade [15].

Também são apresentadas soluções customizadas para fabricantes de equipamentos (OEM, do inglês *Original Equipment Manufacturer*) de acordo com suas necessidades específicas. As soluções para OEM estão divididas nas áreas: Automação de Máquinas, Eletrônica Embarcada, Intervenção Remota de Máquinas, Monitoramento & Gestão de Operações e Tecnologia para Veículos Autônomos [16].

Dentro de cada uma das áreas estão presentes diversos produtos. Entre eles se encontra o piloto automático (*HxGN AgrOn* Piloto Automático), que consiste em um sistema para navegação automatizada e precisa de tratores, maquinas e implementos [17].

Este é um dos diversos produtos produzidos e atualizados pelo setor de desenvolvimento embarcado.

2.1.1 O setor de desenvolvimento embarcado

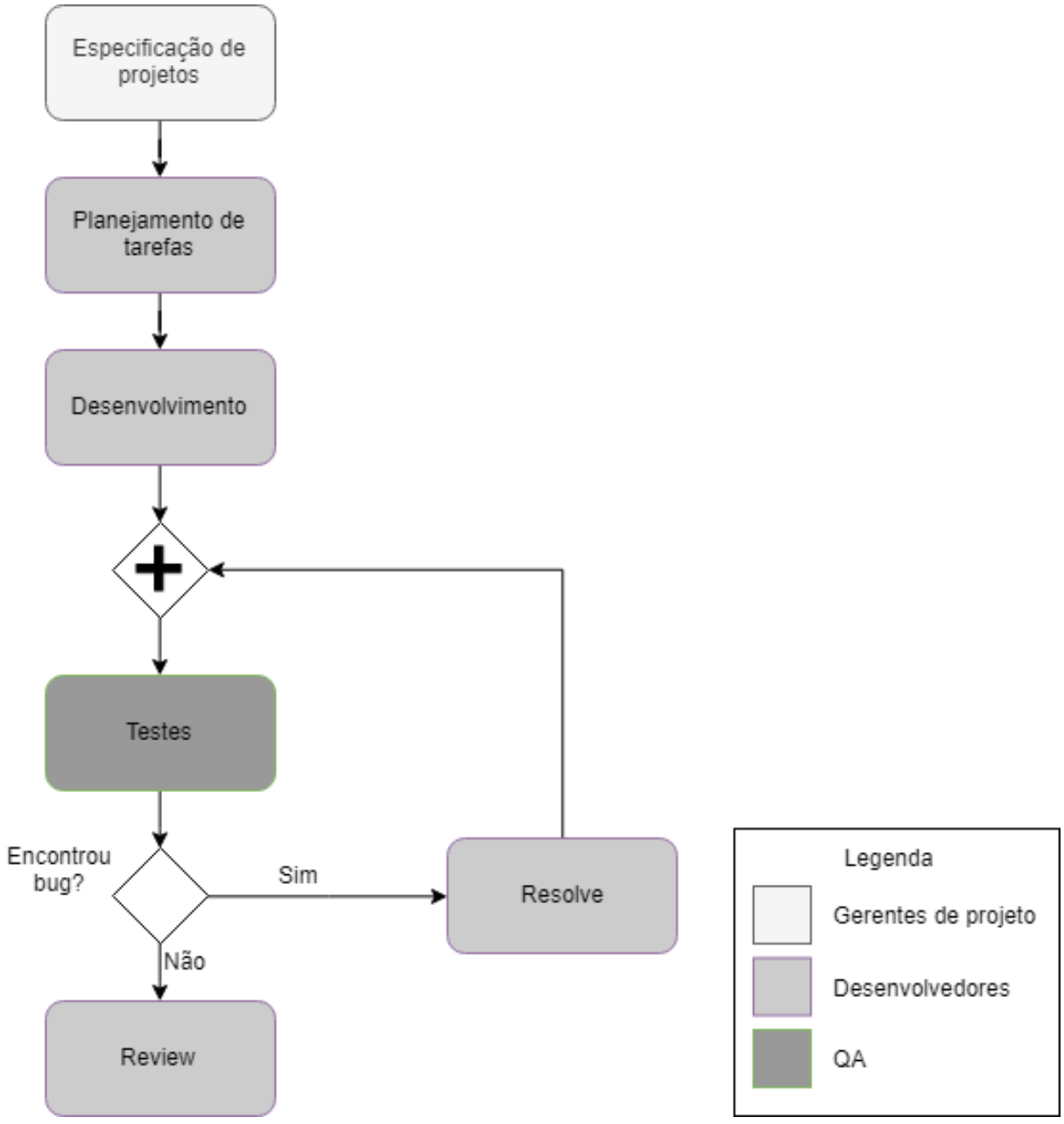
A empresa pode ser dividida em diversos setores, entre eles o setor de Pesquisa e Desenvolvimento (P&D), este, por sua vez, também está dividido em áreas, são essas: desenvolvimento de *hardware*, desenvolvimento de *software web* e desenvolvimento de *software* embarcado. Produtos como piloto automático e controle de implementos fazem parte das demandas do setor de desenvolvimento embarcado e, dentro deste contexto, foi desenvolvido o projeto de fim de curso apresentado neste documento.

Demandas de clientes, variações e melhorias no produto são aspectos constantes no processo de desenvolvimento. Para isso os desenvolvedores passam grande parte do tempo implementando e validando soluções. O projeto desenvolvido busca facilitar o processo de validação e permitir que ela seja feita de forma constante e eficiente, reduzindo seu tempo e custo. As propostas e os objetivos deste trabalho serão apresentados ao decorrer do capítulo 3.

3 O projeto

O processo de desenvolvimento de produtos como o piloto automático, apresentado no capítulo 2, é realizado em diversas etapas. Estas etapas vão do planejamento até a validação do produto. O processo sistêmico da empresa engloba diversas etapas de desenvolvimento, estas são apresentadas na figura 1.

Figura 1 – Processo sistêmico - P&D.



Fonte: Adaptado de [18, 19].

Na especificação dos projetos são realizadas as etapas de descrição e a divisão destes entre as equipes responsáveis. Durante a etapa de especificação são definidos requisitos, custos, valores e a estimativa de tempo de desenvolvimento. Uma vez especificados, os projetos são distribuídos entre as equipes de desenvolvimento que realizarão o planejamento para sua realização. Este planejamento ocorre a cada quinze dias, período chamado de *sprint*, e envolve a definição da abordagem de desenvolvimento, ou seja, como será feito, e a criação e especificação de tarefas para sua realização.

Durante a *sprint* diversas tarefas são realizadas, o que consiste no bloco de desenvolvimento apresentado na figura 1. Esta etapa é realizada por cada desenvolvedor e envolve a execução da tarefa, a criação de testes unitários e a revisão de código por outros membros da equipe e de outras equipes.

Após estas etapas as funcionalidades passam por testes, realizados pela equipe de QA. Caso sejam encontrados problemas, também chamados de *bugs*, será criada uma nova tarefa para correção. A tarefa para correção será adicionada na *sprint* de uma das equipes de desenvolvimento e uma vez corrigida os testes serão refeitos até nenhum *bug* ser encontrado.

Ao final de cada *sprint* é realizada uma reunião de *review*, onde todas as equipes apresentam o que foi realizado. Dessa forma todas as equipes tem conhecimento do que foi produzido durante o período. Este processo é conhecido como o ciclo *scrum*.

No processo de desenvolvimento o retorno de *bugs* encontrados pela equipe de QA ou, até mesmo, por clientes faz com que sejam necessárias alterações frequentes no escopo de desenvolvimento. Estas alterações fazem com que o planejamento previamente realizado sofra grandes mudanças durante a *sprint*, e que os desenvolvedores parem de trabalhar novas funcionalidades para realizar as correções dos problemas encontrados. Isso traz atrasos e custos extras para a empresa e, no caso de um problema encontrado por um cliente, um dano à imagem do produto e da empresa.

Outro grande problema no processo, neste caso na etapa de testes pela equipe de QA, é a necessidade de um grande espaço físico, elevada quantidade de maquinário agrícola e elevado tempo para realização de testes de produtos e funcionalidades. Para isso, vem sendo criadas ferramentas para realização de testes e simulações pelos próprios desenvolvedores. Estas ferramentas fazem com que se possa simular o caso real sem a necessidade de máquinas e espaço físico, porém ainda utilizando o *software* e *hardware* de produção. Outro benefício é a possibilidade de realizar testes de casos adversos, ou seja, casos fora do comportamento normal que não são passíveis de serem realizados normalmente devido às condições geográficas e riscos aos funcionários.

Porém existe a grande dificuldade na inserção do simulador no dia a dia do desenvolvimento devido a necessidade da montagem dos cenários e do aprendizado sobre

o simulador. Com o objetivo de facilitar o uso e diminuir os *bugs* encontrados após o desenvolvimento, este trabalho propõe o desenvolvimento de ferramentas para teste e simulação automatizadas dos produtos, para que possam ser executadas diariamente de forma automática ou de acordo com a necessidade dos desenvolvedores, facilitando o acesso aos testes e garantindo que estes serão executados antes do lançamento de novas versões.

Inicialmente serão construídas as ferramentas para execução da simulação do piloto automático, apresentado no capítulo 2, e testes de sistema, que são apresentados no capítulo 4. Com isso existirá uma base que facilitará a criação de novas simulações e novos tipos de teste. O resultado esperado é que se reduzam os problemas encontrados após o desenvolvimento e que, quando encontrados, seja de forma rápida.

No restante do capítulo serão apresentados os objetivos, a metodologia e as fases do projeto.

3.1 Objetivos

Este trabalho traz como objetivo aumentar, facilitar e reduzir os custos dos testes realizados no produto durante o desenvolvimento, buscando a redução de *bugs* e/ou a redução do tempo até encontrá-los. Para isso são elencados os objetivos específicos:

- Atualizar as ferramentas de simulação existentes;
- Desenvolver ferramentas que auxiliem o processo de simulação de funcionalidades dos produtos da empresa;
- Desenvolver ferramentas que auxiliem a execução dos testes de comportamento;
- Desenvolver uma ferramenta que permita a execução das simulações e testes diariamente de forma automática;
- Criar uma base que facilite a ampliação dos testes e simulações para os demais produtos da empresa e também para produtos desenvolvidos no futuro.

3.2 Metodologia

O desenvolvimento do trabalho foi realizado através do estudo e utilização de técnicas e ferramentas de simulação, fazendo com que trouxessem o melhor beneficio possível aos processos da empresa. A escolha das técnicas e ferramentas e a execução do projeto foram realizadas respeitando o que já estava em vigor na empresa, buscando assim manter compatibilidade e facilitando a inserção do projeto no processo de desenvolvimento.

3.3 Fases do projeto

A execução do projeto foi realizada em diversas etapas, cada uma contribuindo para a solução final. Estas etapas são:

1. Atualização das ferramentas proprietárias para que se adequassem ao projeto;
2. Desenvolvimento da simulação automatizada do piloto automático;
3. Desenvolvimento da ferramenta automatizada para testes de comportamento;
4. Automação da criação do pacote de atualização dos produtos;
5. União das ferramentas e do novo pacote de atualização para que a execução dos testes e simulações seja realizada diariamente.

A fase 1 consistiu em atualizar diversas ferramentas já existentes, de forma que mantivesse seu funcionamento anterior, porém com a possibilidade de serem executadas na forma necessária para cada caso. Cada uma destas ferramentas, sua importância e o que foi alterado será apresentado ao decorrer do capítulo 5.

Os itens 2 e 3 apresentam a criação efetiva das ferramentas propostas e são descritas em detalhes no decorrer do relatório. Nestas fases foram necessárias a criação de diversos estágios, cada um representando uma ação de preparação ou execução.

Já a etapa 4 foi necessária para automatizar o processo de geração de novas versões para os produtos, antes feita manualmente. Com o esforço de automatizar este processo será possível que, diariamente, se tenha uma versão contendo aquilo que foi realizado até o momento de sua criação e também será possível a criação dos pacotes oficiais de atualização, estes lançados para clientes.

Por fim a fase 5 será a união de tudo o que foi proposto, trazendo uma ferramenta que gera uma nova versão diariamente e realiza os testes e simulações existentes. Com isso será possível detectar possíveis problemas com o que foi desenvolvido desde a ultima execução dos testes, sem a necessidade de esperar um teste pela equipe de QA.

Uma vez executadas, estas fases atendem aos objetivos do projeto, trazendo assim os benefícios apresentados. No decorrer do capítulo 4 serão apresentados conceitos relevantes para a compreensão e entendimento do projeto.

4 Testes e simulações

Para o desenvolvimento do projeto, foram utilizados diversos conceitos sobre testes e simulações e diversas ferramentas. Neste capítulo serão apresentados estes conceitos e ferramentas, a sua importância em um ambiente de desenvolvimento e como se encaixam na realização das tarefas propostas.

Inicialmente são abordados testes de *software*, sua motivação e justificativa, seguida de algumas abordagens utilizadas. Na sequência são apresentadas as motivações de se realizar simulações de *hardware* de maneira geral e o como elas auxiliam no processo de desenvolvimento da empresa. Por fim serão apresentadas as ferramentas utilizadas no projeto, seu objetivo e como foram utilizadas para atingir os objetivos.

4.1 Teste de *software*

A construção dos testes de *software* é uma prática que busca determinar se o produto foi capaz de atender as especificações e se está apresentando o funcionamento correto [20]. Tendo como principal objetivo detectar falhas e suas causas os testes fazem parte do ambiente de desenvolvimento.

As falhas em um produto são comportamentos diferentes do esperado pelo usuário, podem ser causados por diversos fatores, entre eles erros (desvios da especificação) e defeitos (falha humana no processo de desenvolvimento). A execução dos testes se torna um último recurso para os desenvolvedores avaliarem o produto antes de sua entrega [20].

O plano de qualidade de *software* da *Hexagon Agriculture* prevê a realização de testes e revisões do que foi desenvolvido. A etapa de revisão é realizada durante todo o ciclo de vida do desenvolvimento de um projeto e trata da inspeção dos requisitos, da arquitetura, da documentação e dos códigos produzidos [21].

Já para os testes o plano de qualidade prevê a realização de testes em quatro níveis: testes de unidade, de integração, de sistema e de aceitação [21]. Estes níveis de testes são previstos pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) [22].

A seguir são apresentados os quatro níveis e sua importância no desenvolvimento de produtos.

4.1.1 Testes de unidade

Os testes de unidade, ou testes unitários como podem ser chamados, tem por objetivo testar a menor unidade do projeto. Nesta fase são desenvolvidos casos de teste

que buscam detectar falhas de lógica e de implementação em métodos e funções ou, até mesmo, em pequenos trechos de código [22].

Esta etapa é realizada pelos desenvolvedores na etapa de codificação. Para toda nova funcionalidade ou modificação no código de produção devem ser executados e/ou implementados testes de unidade [21].

Nesta etapa já exite um certo grau de integração, pois, para qualquer conteúdo desenvolvido ser integrado ao *software* final são executados todos os testes unitários.

4.1.2 Testes de integração

Uma vez que os testes de unidade buscam provocar falhas na menor unidade do projeto, os testes de integração trazem a ideia de procurar falhas na comunicação entre os módulos do produto. Estes testes buscam garantir que a união das diversas partes da própria aplicação e sua comunicação com outras aplicações do produto está ocorrendo da forma planejada na fase de projeto [21, 22].

Nos produtos desenvolvidos pela *Hexagon Agriculture* existem diversos softwares que em conjunto permitem a funcionalidade do sistema. Os testes de integração buscam falhas na integração entre estes sistemas.

4.1.3 Testes de sistema

Os testes de sistema buscam, por sua vez, avaliar o comportamento do *software*. Esta avaliação é realizada no sistema como um todo, buscando utiliza-lo da mesma forma que o usuário final [21, 22].

Estes testes procuram falhas no cenário onde o produto será utilizado, ou seja, utilizando condições e dados iguais ou próximos a realidade. Este tipo de teste é realizado utilizando o conceito de teste caixa-preta. O conceito de testes caixa-preta é o de que o teste apenas fornecerá um conjunto de entradas e verificará se ao final o resultado é o esperado.

Durante este trabalho, os testes de sistema foram chamados de testes de comportamento, um vez que avaliam o comportamento do produto como um todo.

4.1.4 Testes de aceitação

Por fim, os testes de aceitação são realizados por um grupo seleto de pessoas, geralmente gerentes de projeto ou, até mesmo, clientes. Estes testes são realizados com o sistema já pronto e tem por objetivo verificar se atende todos os requisitos necessários e, desta forma, não encontrando erros, o produto está pronto para ser lançado aos demais usuários.

4.2 Simulações *hardware in the loop*

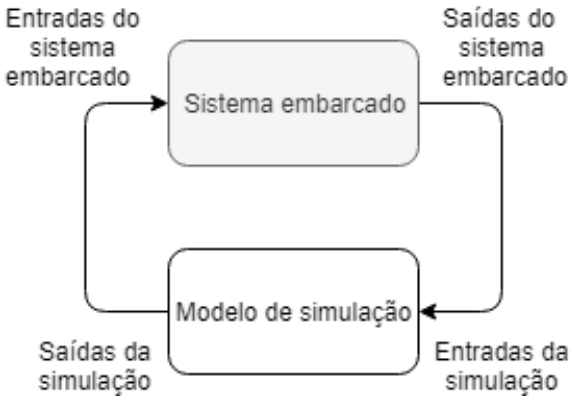
Outro aspecto de grande importância no desenvolvimento do projeto é a realização de simulações do produto. Com o uso de simulação podem sem realizados testes em condições similares às encontradas em campo. Normalmente, para o processo de testes é necessária utilização de um grande número de recursos de maquinário, espaço físico e mão-de-obra, fatores que encarecem o processo de teste.

Através do uso de simulações é possível a realização de testes em situações muito similares às apresentadas em testes reais sem a necessidade do elevado número de recursos. Outro fator importante a respeito de simulações é a possibilidade de criação de cenários de simulações em condições adversas, ou seja, em condições que exponham risco aos usuários ou situações que não podem ser reproduzidas com os equipamentos e/ou espaço físico disponível.

Os produtos da *Hexagon Agriculture*, em grande parte, consistem em sistemas embarcados onde tanto o *hardware* quanto o *software* foram por ela desenvolvidos. Para simular o sistema embarcado como um todo, podem ser utilizados conceitos de simulação *hardware in the loop*.

Na figura 2 é apresentado o princípio de simulação com *hardware in the loop*. Como é apresentado na imagem, as saídas do sistema embarcado alimentam a simulação, da mesma forma que as saídas da simulação alimentam o sistema embarcado. Com isso se torna possível simular o meio físico com o uso do produto em sua forma completa [23].

Figura 2 – Principio de simulações *hardware in the loop*.



Fonte: Adaptado de [23].

4.3 Ferramentas utilizadas

Para o desenvolvimento do trabalho foram utilizadas diversas ferramentas e *frameworks*. Estas ferramentas foram utilizadas para execução dos testes e simulações e também no desenvolvimento das tarefas propostas.

Para isso foram necessários estudos de como dada ferramenta poderia ser utilizada para trazer os melhores benefícios ao projeto. Como na metodologia foi determinado o uso de ferramentas já utilizadas pela empresa, a escolha das ferramentas foi feita não só em base na sua adequação ao proposto, mas também em relação ao seu benefício em relação ao já utilizado.

Ao decorrer do restante do capítulo serão apresentadas as ferramentas utilizadas no processo de desenvolvimento, bem como a forma que cada uma contribui para o desenvolvimento do projeto.

4.3.1 *Cucumber*

O *Cucumber* é um *framework* desenvolvido para Desenvolvimento Orientado a Comportamento (BDD do inglês *Behavior-Driven Development*) [24]. BDD é um conjunto de práticas que realiza todo o desenvolvimento orientado aos testes, ou seja, todo o desenvolvimento ocorre após a criação dos testes [25].

Com o uso desta técnica a codificação do produto ou funcionalidade é realizada de forma a fazer o teste passar [25]. A criação dos testes, neste caso, é realizada tendo como base os requisitos e o planejamento do projeto. Porém seu uso na empresa é realizado na construção dos testes de sistema, ou seja, como testes de comportamento realizados após o desenvolvimento da funcionalidade.

A escrita de testes com *Cucumber* é realizado utilizando *Gherkin*, que corresponde a um conjunto de palavras que dão estrutura a instruções legíveis [26]. Os testes produzindo com essa ferramentas devem ser legíveis para o usuário e devem representar as especificações do projeto.

4.3.2 V-Rep

O V-Rep é um simulador criado pela *Coppelia Robotics* e foi desenvolvido para a simulação de robôs e veículos. Através de um ambiente de desenvolvimento que permite a criação de modelos e objetos que podem ser controlados de forma individual, a ferramenta se torna muito versátil e poderosa [27].

Antes do desenvolvimento do projeto, a empresa já fazia o uso do simulador para validação de algumas funcionalidades. A escolha do simulador foi realizada por sua versatilidade e possibilidade de comunicação com ferramentas já existentes.

No desenvolvimento do projeto foram utilizadas cenas anteriormente desenvolvidas, logo, não foram realizados novos desenvolvimentos para o simulador.

4.3.3 GoCD

O GoCD é uma ferramenta utilizada para organização e automação dos processos de desenvolvimento contínuo de *software*. Para isso a ferramenta é capaz de apresentar para os usuários diversas etapas de visualização do processo e permitir a modelagem de complexos fluxos de trabalho. Trazendo como benefício o auxílio na automação de processos, a ferramenta também auxilia na rastreabilidade e recursos e problemas [28, 29].

Na Hexagon, a ferramenta já era utilizada no processo de integração contínua de novas funcionalidades, realizando a compilação e execução de testes unitários antes de integrar novos conteúdos na linha de desenvolvimento. No desenvolvimento deste projeto o GoCD foi utilizado para desenvolver as ferramentas de automação de testes propostas.

Este desenvolvimento ocorreu através da criação de fluxos de trabalho, compostos por diversas etapas, para cada etapa principal foi desenvolvida uma *pipeline*, que é a abstração de um conjunto de tarefas. Em cada *pipeline* foram desenvolvidos estágios que executam as tarefas propostas.

Ao decorrer do capítulo 5 é apresentado o desenvolvimento do projeto e, por consequência, um pouco mais sobre a utilização das técnicas e ferramentas apresentadas ao decorrer do capítulo.

5 Técnicas desenvolvidas

O trabalho foi desenvolvido em diversas etapas, estas já apresentadas no capítulo 3, o desenvolvimento das tarefas lá apresentadas é descrito ao decorrer do capítulo. São apresentados os objetivos, justificativas e a solução encontrada nas alterações realizadas em soluções já existentes e para as ferramentas desenvolvidas. O conteúdo apresentado neste capítulo foi desenvolvido pelo autor.

Todas as etapas envolvem a utilização de equipamentos de *hardware* produzidos pela empresa, que devem ser testados e simulados juntamente com o *software* de produção (*hardware in the loop*). O *hardware* é composto por um computador de bordo [30], chamado de Titanium (anexo A), e um *driver* no caso do piloto automático.

O Titanium é onde o usuário irá interagir com o sistema e onde são executadas maior parte das funcionalidades do produto, assim como parte da interação com o meio físico, como o recebimento de dados de GNSS. Já o *driver* é responsável por se comunicar com outros dados do meio físico, como atuadores e sensores, e também realizar os cálculos dos sistemas de controle. Durante este capítulo o sistema composto pelo Titanium e *driver* será chamado apenas de sistema.

5.1 Atualização das ferramentas de simulação existentes

Antes do desenvolvimento do projeto, já existiam ferramentas que auxiliavam ou permitiam a realização de simulações. Estas ferramentas, de modo geral, tinham como propósito auxiliar na configuração ou na execução de etapas do processo.

Com o objetivo de automatizar a execução dos testes foi necessária a adaptação e atualização dessas ferramentas. A descrição da função e das alterações de cada ferramenta são descritas a seguir.

5.1.1 *Simulation Gateway* - Gerenciador de dados da simulação

O *Simulation Gateway* é responsável pela troca de mensagens entre o simulador (V-Rep) e o *hardware* que será utilizado na simulação. Para o correto funcionamento da simulação, a ferramenta deve capaz de:

- Receber mensagens do meio físico, simuladas pelo V-Rep;
- Receber mensagens de controle do sistema;
- Enviar mensagens ao simulador;

- Enviar mensagens ao sistema.

São necessários três tipos básicos de mensagens: mensagens CAN (do inglês *Controller Area Network*), sinais enviados pelo simulador e mensagens D-Bus (do inglês *Desktop Bus*). Quais mensagens devem ser recebidas ou enviadas e qual ação deve ser executada quando determinada mensagem é recebida é configurável através de um arquivo de configuração.

De acordo com cada simulação são necessárias diferentes configurações. No arquivo de configuração podem ser definidas configurações para cada um dos tipos de mensagens. A configuração consiste em definir quais mensagens serão recebidas ou enviadas, para cada mensagem recebida pode ser definido uma ação, que, de modo geral, consiste em tratar e encaminhar um dado recebido através de uma mensagem para outra mensagem, podendo ser do mesmo tipo ou não.

O tratamento dos dados é realizado através de *codecs*, que são classes que encapsulam o tratamento do dado recebido e a construção de uma nova mensagem. Os *codecs* podem ser definidos na configuração para tratarem dados recebidos e enviados.

Para a simulação do piloto automático esta é uma ferramenta essencial, uma vez que permite a comunicação entre os sistemas existentes. Para seu uso no contexto do projeto não foram necessárias alterações, porém, foram implementadas melhorias no tratamento de dados e o envio e recebimento de dados via *D-Bus*.

5.1.2 V-Rep *Scene Manager* - gerenciador de cenas

O V-Rep *Scene Manager* é responsável por gerenciar as cenas de simulação, ou seja, configurar o simulador e o Titanium para determinada simulação. Esta configuração permite que o usuário defina veículos, tanto no simulador quanto no Titanium, instale um pacote de configuração no computador de bordo, execute *scritps* com ações pré- e pós-simulação, entre outras possibilidades.

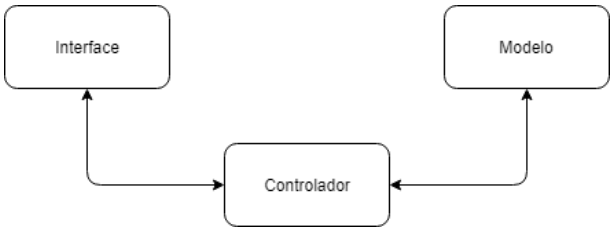
Em sua forma original o uso do gerenciador de cenas era possível apenas através do uso de uma interface gráfica. Mesmo existindo a possibilidade de importar configurações de simulação através de um arquivo *json*, ainda eram necessárias ações do usuário para sua execução. Estas ações consistiam essencialmente em realizar a conexão com o Titanium e com o simulador e dar o início à simulação.

Com o objetivo de automatizar seu funcionamento, foi adicionada a opção de passar o arquivo de configuração e realizar a conexão e dar início diretamente na execução da aplicação. Para isso foi necessária uma reestruturação na arquitetura do projeto, com a alteração das classes responsáveis por apresentar e controlar a interface do usuário.

Com o uso desta opção o V-Rep *Scene Manager* é capaz apenas de executar ações pré-simulação, para que não seja necessário que o usuário ou outra aplicação sinalize seu fim. Sendo assim, após as configurações terem sido realizadas o gerenciador de cenas irá encerrar, indicando sucesso ou falha na configuração.

A reestruturação foi realizada segundo a arquitetura Modelo-Visão-Controlador (MVC, do inglês *Model-View-Controller*). Essa arquitetura separa a aplicação da sua representação ao usuário, permitindo que as funcionalidades possam ser executadas sem dependências com a interface. Para isso o *software* é separado em três partes: modelo, controlador e apresentação. O modelo é onde está presente a parte lógica da aplicação, responsável por toda a execução do processo. Já interface é responsável por apresentar os dados ao usuário, porém, nesta arquitetura, não interfere diretamente no funcionamento do modelo. Por fim, o controlador é o responsável pela troca de dados entre o modelo e a interface [31]. Na figura 3 é apresentado um diagrama que representa a arquitetura.

Figura 3 – Modelo-Visão-Controlador (MVC).



Fonte: Autor.

Com o uso desta arquitetura, foi possível remover a figura da interface sem afetar o funcionamento do restante da aplicação. Dessa forma, quando executado na opção sem interface, a figura do controlador será responsável por realizar as definições necessárias para a execução do modelo, e, para isso, a aplicação receberá como argumento todos os dados necessários para a execução.

5.1.3 *Kilgrave* - reprodutor de *logs* de campo

O *Kilgrave* é um *software* responsável por reproduzir arquivos de dados coletados em campo, chamados de *logs*, no computador de bordo com o objetivo de reproduzir as mesmas circunstancias durante o teste. Os *logs* que são reproduzidos são de dois tipos: mensagens GNSS e mensagens CAN, que devem ser executados de acordo com o tempo em que ocorreram quando foram gerados, este tempo é informado nos arquivos e é conhecido como *timestamp*.

Seu uso para o projeto é o de permitir a execução de simulações baseadas em situações existentes em campo ou reproduzir *bugs* encontrados. Durante os testes da

ferramenta foi verificada a necessidade de algumas alterações e correções, principalmente no que diz respeito a leitura dos arquivos e correção de *bugs* encontrados.

Inicialmente a leitura dos dados de GNSS era realizada apenas uma vez, no início na execução, fazendo que os dados ficassem armazenados em memória. Este comportamento teve que ser alterado, pois, os arquivo podem possuir dados de muitas horas de execução, o que poderia fazer com que possuam um tamanho elevado, ocupando muita memória do sistema.

Uma vez que o arquivo com os dados estava em memória, a execução era iniciada e os dados informados ao Titanium linha a linha, levando em conta seu *timestamp* em relação à última informação enviada. O envio ao computador de bordo era, e ainda é, realizado com o uso do protocolo serial.

Para solucionar o problema do tamanho dos arquivos, a aplicação foi alterada de forma que a aquisição da informação seja realizada apenas quando necessário. Para isso, o arquivo é aberto, porém apenas uma linha, ou seja, um dado válido, é armazenado. Ao realizar o envio da informação ao computador de bordo, o *Kilgrave* fará a aquisição de um novo dado, e fará seu envio respeitando o tempo de execução, seguindo assim até o fim do arquivo e, por consequência, o fim da simulação.

Já os dados CAN eram reproduzidos através do *canplayer* [32], porém foram detectados problemas na sincronização dos dados, para solucionar o problema foi planejado utilizar a mesma solução utilizada para o *log* GNSS juntamente com o *cansend* [33]. Porém, devido a inexistência de *logs* de campo que permitissem a utilização desta ferramenta e ao surgimento de outras demandas a finalização desta etapa da refatoração foi colocada para trabalhos futuros.

5.1.4 *Morpheus* - gerenciador de caminhos

O *Morpheus* é utilizado para gerenciar os caminhos, ou guias de referência, nas simulações do piloto automático. Seu funcionamento é, de forma geral: dado um arquivo de configuração dos caminhos, para cada caminho presente na configuração posicionar o veículo no local e direção correta para o início da simulação, definir uma velocidade para o veículo e alimentar a guia de referência para o computador de bordo até que o caminho seja completado. Caso durante a execução o piloto automático seja desarmado, o que pode acontecer devido a falhas, a execução irá finalizar indicando que ocorreu um erro.

A configuração dos caminhos podem ser realizadas de diversas formas, entre elas:

- Dados três pontos gerar um caminho circular;
- Dados dois pontos gerar um caminho em linha reta;

- Dada uma direção gerar um caminho em linha reta;
- Dado um arquivo de pontos, gerar o caminho representado por eles.

Para cada caminho podem ser configurados também o número de repetições, a velocidade do veículo e se se trata de um circuito. Os circuitos são caminhos que tem o início e o fim no mesmo ponto, neste caso, quando definidos como circuitos na configuração o gerenciador de caminhos não irá reposicionar o veículo e sim continuar até que o número de voltas seja igual ao número de repetições fornecido.

Essa ferramenta apresenta um papel essencial na simulação do piloto automático sendo responsável pelo início efetivo da simulação e também por fornecer os dados de referência ao veículo. O *Morpheus* é executado durante toda a simulação, indicando seu início e fim.

Da mesma forma que o gerenciador de cenas, foi necessário adicionar um modo onde não é apresentada uma interface ao usuário. No caso do gerenciador de caminhos toda a estrutura já permitia a configuração sem a necessidade da interação do usuário, porém ainda apresentando a interface. Dessa forma foi apenas adicionada uma nova opção na execução indicando que a interface não deve ser apresentada.

Também foram necessárias correções de *bugs* encontrados no cálculo da posição inicial onde o veículo é posicionado em cada caminho.

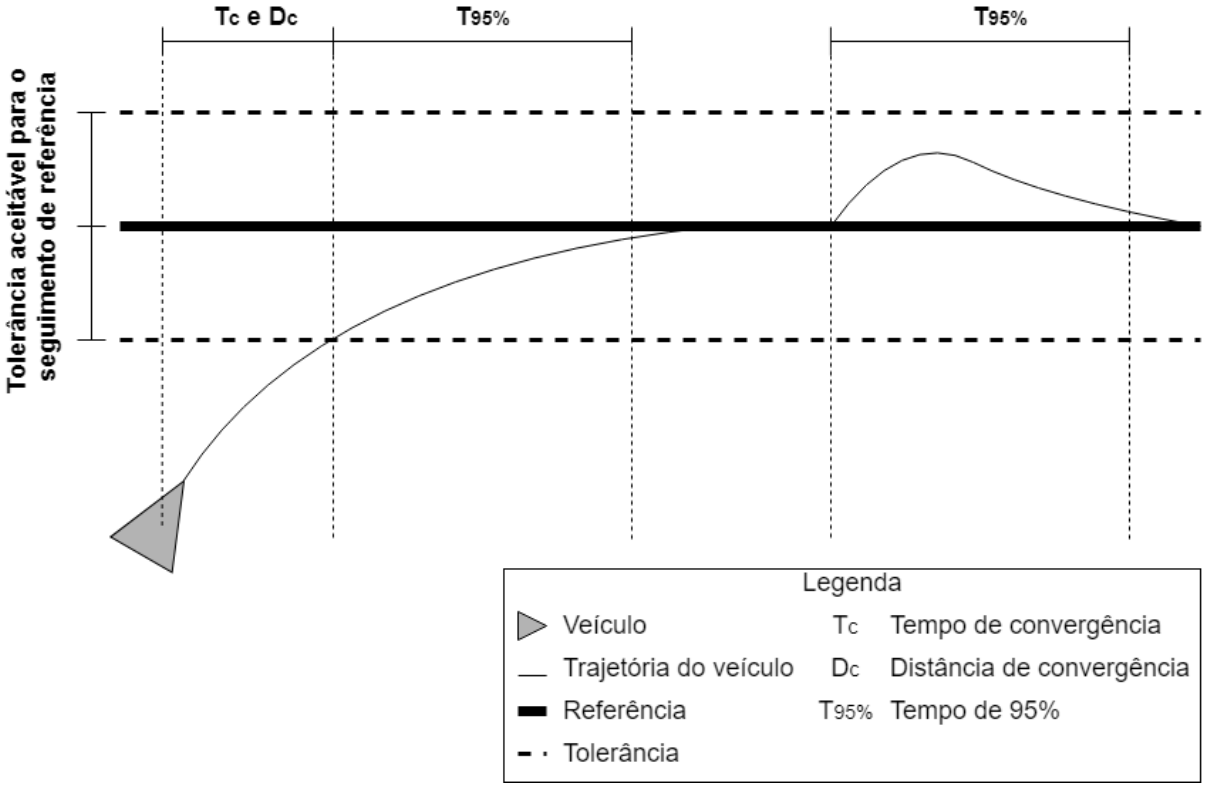
5.1.5 *Logparser* - *scripts* de averiguação do resultado da simulação

Por fim, o *Logparser* é responsável por averiguar o resultado da simulação, o *software* irá se comunicar com o Titanium e coletar os dados do piloto. As informações que podem ser obtidas são relacionadas à performance do piloto, conforme esquema da figura 4, dentre as informações que podem ser obtidas, as que apresentam maior grau de relevância na averiguação das informações da simulação são:

- Distância percorrida até a convergência (D_c): representa a distância percorrida até o veículo entrar na zona de tolerância;
- Tempo decorrido até a convergência (T_c): representa o tempo decorrido até o veículo entrar na zona de tolerância;
- Tempo de 95% ($T_{95\%}$): é o tempo decorrido para a trajetória atingir 95% da referência a partir da tolerância;
- Erro RMS (E_{RMS}): representa o valor RMS do erro durante o percurso;
- Erro médio ($E_{médio}$): representa o valor médio do erro durante o percurso, indicando a presença de *offset* no seguimento de referência;

- Valores médio e absoluto para os ângulos *roll* e *pith*;
- Duração e distância percorridas durante a execução, que quando verificados para um mesmo percurso podem indicar problemas.

Figura 4 – Esquema de variáveis do piloto automático.



Fonte: Autor.

Em todos os itens podem ser obtidos os valores mínimo, máximo e médio.

Para que se possa apresentar um resultado se a simulação falhou ou foi bem sucedida, foi implementada a funcionalidade de comparar os valores obtidos com valores referência, informados através de um arquivo de configuração. Esta comparação será realizada apenas para as variáveis que estiverem presentes no arquivo, dessa forma, apenas as informações requeridas serão utilizadas na averiguação. Um exemplo do arquivo de configuração é apresentado abaixo.

```
{
    "acquisition": {
        "max": 15,
        "mean": 10
    },
    "p95": {
```

```
        "max": 16,
        "min": 10,
        "mean": 13
    },
    "mean_roll": {
        "max": 20,
    },
    "engage_time": {
        "max": 15,
        "mean": 10
    },
    "mean_pitch": {
        "max": 30
    }
}
```

Caso um ou mais valores obtidos excedam o valor de referência a execução indicará falha e serão apresentadas mensagens informando quais critérios não foram atendidos. As mensagens são no formato:

```
Values beyond expectation in acquisition max, value: 19.12
    expectation: 15
```

em português: "Valor acima do esperado em *acquisition max*, valor: 19,12 expectativa: 15".

5.2 Novo pacote de atualização do sistema

Até a versão atual do sistema o pacote de atualização era realizado de forma manual pelos líderes de equipe. O pacote contém arquivos do sistema operacional do computador de bordo e as aplicações, desenvolvidas pela empresa, que são executadas no dispositivo. Com o objetivo de facilitar a geração do pacote e de permitir que os testes e simulações sejam executados em um dispositivo que contemple tudo o que foi desenvolvido até o momento de sua execução, foi realizado um esforço para automatizar a geração da atualização.

Para isso foram necessárias atualizações e modificações em diversos projetos e repositórios e a criação de *scripts* para a compilação e geração do pacote de instalação, chamado de uti, tarefa realizada pelo autor em conjunto com membros de todas as equipes de desenvolvimento embarcado. Porém, para o desenvolvimento deste trabalho o aspecto de maior relevância foi o desenvolvimento de uma *pipeline* no GoCD para a geração e armazenamento do pacote e símbolos de *debug*, esta, desenvolvida pelo autor.

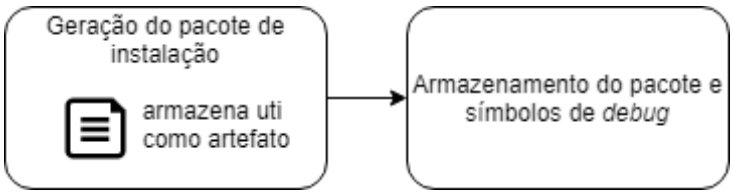
A *pipeline* desenvolvida possui dois propósitos: a geração dos pacotes oficiais para cada versão e como estágio inicial para a execução dos testes e simulações automatizadas. Este estágio é de grande importância, pois garante que tudo será executado com a versão desejada de todos os componentes do sistema.

Sendo o estágio inicial, ou seja, que dará início às demais etapas, este será executado de forma automática uma vez ao dia de segunda a sexta, buscando testar e simular o conteúdo que foi integrado às aplicações durante o dia. Quando executado automaticamente irá gerar uma versão com o nome *build* seguido de um número identificador, quando executada por um usuário, este poderá definir um nome para a versão, permitindo assim a geração de versões oficiais.

A geração do uti é realizada de forma simples e sequencial, dessa forma, apresenta apenas dois estágios. O primeiro estágio é responsável pela geração do pacote, executando atualizações do repositório local e executando o *script* de geração do pacote informando a versão e o nome do pacote que será gerado. Já o segundo estágio é responsável por armazenar o uti e os símbolos de *debug* gerados em um servidor da empresa.

Ao fim do primeiro estágio, o pacote de instalação é adicionado como um artefato da *pipeline*, permitindo assim, que as *pipelines* que serão executadas posteriormente o utilizem. Na figura 5 é apresentado o diagrama desta etapa.

Figura 5 – Diagrama da geração do pacote de atualização.



Fonte: Autor.

Para a execução do projeto, a etapa mais essencial é a geração do pacote, dessa forma, o estágio de armazenamento dos arquivos no servidor será realizado em trabalhos futuros.

5.3 Instalação do pacote de atualização

Uma vez gerado o pacote de atualização, este deverá ser instalado no Titanium de simulação. Para esse fim, foi desenvolvida uma nova *pipeline* no GoCD que executará o processo de instalação.

Esta *pipeline* será executada logo após a geração do pacote e fará o uso do pacote gerado como artefato na etapa anterior. Dessa forma esta etapa consiste na execução das

tarefas:

1. Obter o pacote de atualização do servidor;
2. Encerrar a execução da aplicação do computador de bordo;
3. Copiar o pacote para o Titanium;
4. Executar o *software* de instalação;
5. Editar *scripts* modificados pela atualização, de forma que permitam a execução dos testes e simulações;
6. Reiniciar o computador de bordo.

Na etapa 5 são editados *scripts* que tem como função limitar o número de conexões remotas e gerar arquivos de *debug* em caso de falha crítica no sistema, chamados de *coredumps*. As limitações de acesso devem ser removidas para que os comandos executados durante a simulação e os testes não sejam bloqueados, já a geração de *coredumps* foi removida por se tratar de um processo lento, o que traria falhas na execução dos testes de comportamento. As falhas críticas ainda serão informadas ao usuário para que este possa verificar sua origem.

Todas estas etapas devem ser realizadas de forma sequencial e não apresentam grande complexidade, por isso serão realizadas em apenas um estágio. Uma vez que a *pipeline* é executada com sucesso, o computador de bordo estará atualizado com a versão gerada pela *pipeline* apresentada no item 5.2.

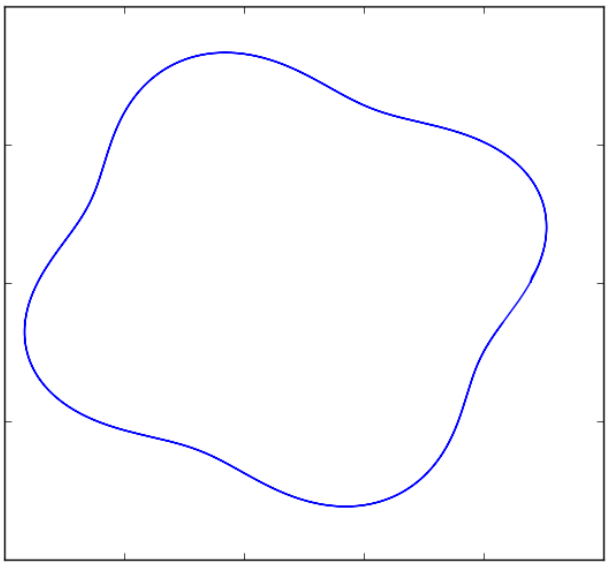
5.4 Simulação automatizada do piloto automático

A simulação do piloto automático é responsável por validar se o funcionamento do produto está dentro do esperado. Para isso, a simulação realiza a execução de diversos caminhos, ou rotas, buscando verificar o comportamento básico do sistema e checar se os requisitos de funcionamento estão sendo atendidos.

Para a validação do funcionamento foram definidos três caminhos em duas diferentes configurações. Na figura 6 é apresentada a primeira rota, seu objetivo é verificar o funcionamento correto do piloto automático em um percurso que apresenta curvas suaves em ambas as direções, representando o funcionamento mais usual. Sua execução é realizada em duas formas: circuito fechado ou aberto. A execução como circuito fechado irá completar o número de voltas definido sem reposicionar o veículo na posição inicial, mantendo o funcionamento constante até o fim das voltas. Já como circuito aberto, ao fim de cada volta o veículo será reposicionado no local de início e o piloto automático

será desativado e posteriormente reativado, permitindo, assim, a avaliação do tempo de convergência do sistema.

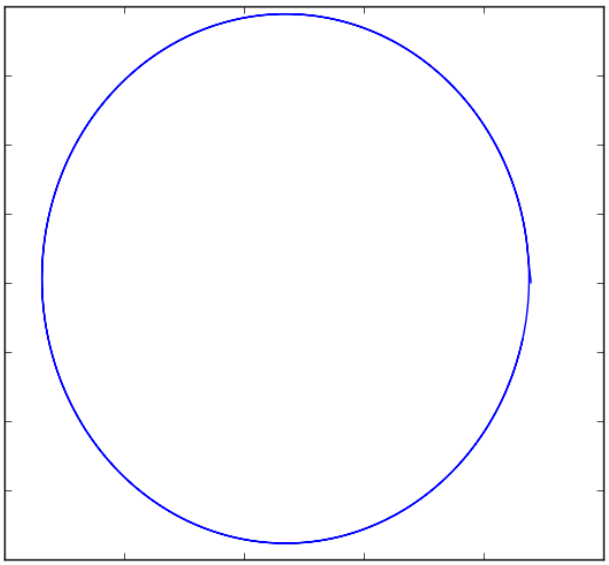
Figura 6 – Simulação do piloto automático - caminho 1.



Fonte: Autor.

O segundo caminho utilizado, apresentado na figura 7, busca verificar o sistema em um percurso sem variação de direção. Este caminho, da mesma forma que o anterior, é executado como um circuito aberto e como um circuito fechado.

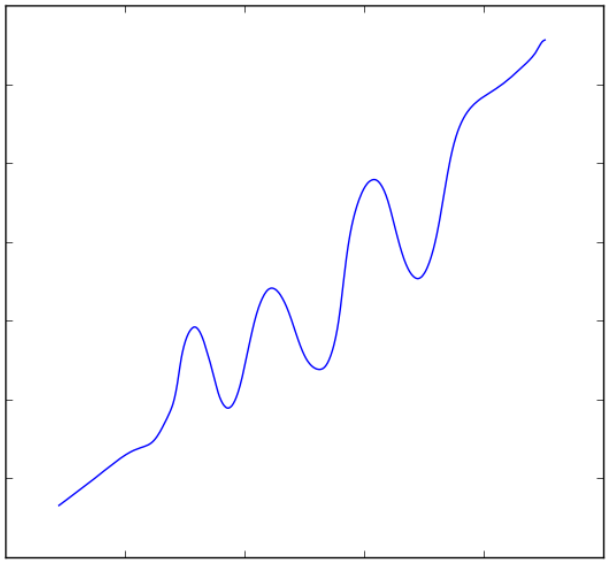
Figura 7 – Simulação do piloto automático - caminho 2.



Fonte: Autor.

Por fim, o caminho três apresenta curvas mais sinuosas, validando a performance do sistema em curvas, a rota é apresentada na figura 8. Este caminho não finaliza no mesmo ponto onde começa, logo, não é possível ser executado como um circuito fechado, devido a isso, será executado mais vezes durante a simulação.

Figura 8 – Simulação do piloto automático - caminho 3.



Fonte: Autor.

Para a execução da simulação são necessárias diversas etapas de compilação, configuração, execução e checagem. As ferramentas apresentadas no item 5.1 são utilizadas com o propósito de facilitar este processo. Buscando realizar as diversas etapas de forma automática foi desenvolvida uma *pipeline*, no GoCD, que executa todas as ações necessárias para a simulação.

De forma a cumprir com as diversas etapas, a *pipeline* foi separada em sete estágios sequenciais, são eles:

1. Atualizar do repositório de simulação;
2. Reiniciar o simulador (V-REP);
3. Compilar as ferramentas de simulação;
4. Executar o V-Rep *Scene Manager*;
5. Iniciar a aplicações do computador de bordo;
6. Executar o *Morpheus*;
7. Verificar os resultados da simulação.

O estágio 1 consiste de apenas uma tarefa e é responsável por garantir que todos os arquivos necessários para o projeto estejam atualizados em sua versão mais recente. Estes arquivos consistem nos códigos fonte das ferramentas e nos recursos de simulação, que são os arquivos e pacotes de configuração utilizados pelas ferramentas. Para isso, nesta etapa, são realizadas operações de clonagem e atualização de submódulos do repositório online, com o uso da ferramenta de controle de versão Git [34].

Já o estágio 2 é responsável pela preparação do simulador. Para o correto funcionamento das ferramentas de simulação, que interagem com o simulador, é necessário que este esteja aberto, porém não executando simulações. Dessa forma o V-REP é encerrado no agente, computador onde as ferramentas são executadas, e posteriormente reiniciado, com isso garantindo que se encontra nas condições necessárias.

O estágio 3 é responsável pela compilação das ferramentas de simulação e é composto por três tarefas, cada uma responsável por uma das ferramentas. As tarefas de compilação são realizadas de forma paralela, uma vez que não possuem dependências entre si.

No estágio 4 é realizada a execução do V-Rep *Scene Manager*. Com isso o computador de bordo e o simulador são preparados para a execução da simulação. No computador de bordo é realizada a instalação de um pacote de configuração, existente no repositório do projeto de simulação. Já no V-Rep é iniciada a simulação adicionando o veículo correspondente ao Titanium de simulação.

A última etapa de configuração acontece no estágio 5 onde serão iniciadas as aplicações do computador de bordo. A primeira aplicação a ser iniciada é o *Simulation Gateway*, para isso é copiado o arquivo de configuração, que contém as informações sobre as mensagens que o *software* deve receber e mandar, e a ferramenta é executada. Posteriormente é iniciada a aplicação principal do computador de bordo.

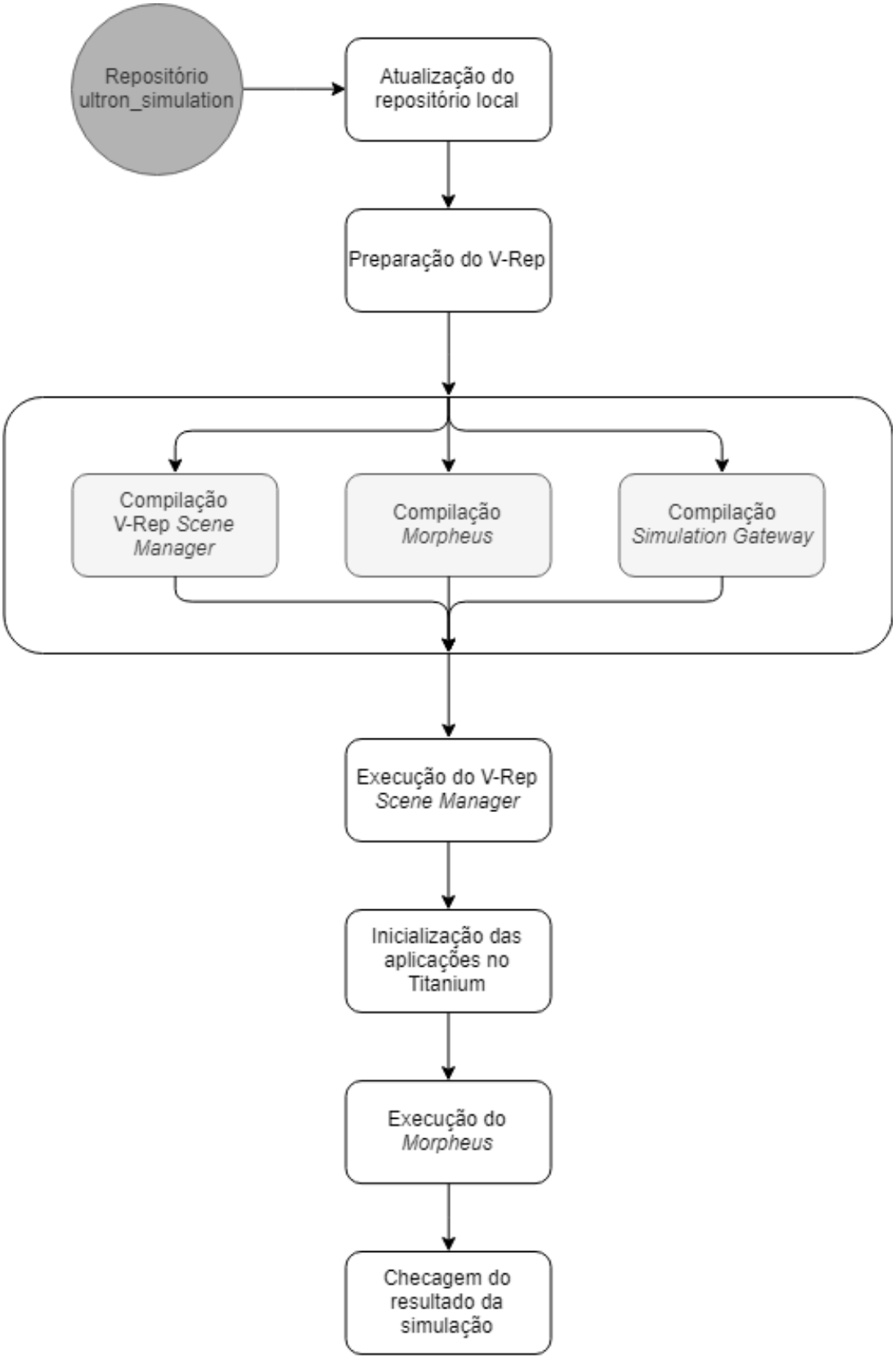
No estágio 6 é executado o *Morpheus*, responsável por gerenciar a simulação. Durante a execução desta etapa o computador de bordo estará se comportando da mesma forma que aconteceria para o funcionamento normal do produto, assim gerando dados confiáveis para análise. Por fim, no estágio 7 é realizada a coleta e análise dos dados gerados, permitindo informar se a simulação apresentou um resultado satisfatório.

A falha na execução de qualquer estágio resulta na falha da simulação, sendo indicado o estágio e a tarefa onde a falha ocorreu. Dessa forma, a simulação segue o fluxo do diagrama apresentado na figura 9.

5.4.1 Resultados de simulação

A verificação dos resultados é realizada no estágio 7. Nele são checadas as métricas definidas no arquivo de configuração da simulação. Para a definição de quais variáveis e quais valores de referência devem ser utilizados, foram executadas simulações com elevado

Figura 9 – Diagrama da simulação do piloto automático.



Fonte: Autor.

número de voltas em cada circuito.

A simulação nesta condição foi executada duas vezes e os resultados são apresentados no quadro 5.1. Cada execução levou, aproximadamente dezoito horas para ser executada, fator que impossibilita que estas condições sejam repetidas diariamente.

Informação		Execução 1	Execução 2
D_c [m]	Máximo	12,47	11,45
	Mínimo	3,611	4,244
	Média	9,767	9,722
T_c [s]	Máximo	6,167	5,681
	Mínimo	1,869	2,294
	Média	4,863	4,837
$T_{95\%}$ [s]	Máximo	19,0	15,0
	Mínimo	1,0	1,0
	Média	8,475	6,541
E_{RMS} [cm]	Máximo	38,47	7,28
	Mínimo	0,022	0,022
	Média	4,134	3,140
$E_{\text{médio}}$ [cm]	Máximo	0,0756	0,0199
	Mínimo	0,0	0,0
	Média	0,01419	0,01032
Roll máximo absoluto [°]	Máximo	1,72	1,67
	Mínimo	0,14	0,13
	Média	1,288	1,240
Roll médio [°]	Máximo	0,3382	0,3278
	Mínimo	0,0401	0,0395
	Média	0,2654	0,2639
Pitch máximo absoluto [°]	Máximo	0,72	0,7
	Mínimo	0,19	0,2
	Média	0,5715	0,5699
Pitch médio [°]	Máximo	0,1652	0,1618
	Mínimo	0,1102	0,1098
	Média	0,1460	0,1464
Duração [s]	Máximo	24951,7	24944,7
	Mínimo	94,72	94,71
	Média	519,6	519,4
Distância percorrida [m]	Máximo	48261,5	48195,1
	Mínimo	189,8	190,7
	Média	1029,3	1029,1

Quadro 5.1: Valores obtidos em simulação aumentada.

A partir dos dados encontrados e das especificações do piloto automático foram levantadas as métricas utilizadas para checagem do resultado. Estas métricas, são apresentadas no quadro 5.2.

Os valores escolhidos para D_c, T_c, $T_{95\%}$, E_{RMS} e $E_{\text{médio}}$ foram realizada levando em consideração as especificações do produto, ou seja, os valores que são informados ao cliente

Informação		Referência
D_c [m]	Média	10
T_c [s]	Máximo	10
$T_{95\%}$ [s]	Média	10
E_{RMS} [cm]	Média	5
$E_{\text{médio}}$ [cm]	Média	5
Roll máximo absoluto	Máximo	3
Pitch máximo absoluto	Máximo	3

Quadro 5.2: Métricas de checagem do resultado de simulação.

na venda do piloto automático. Os valores definidos para *roll* e *pitch* foram escolhidos para uma checagem de sanidade da resposta, ou seja, valores que, se atingidos, indicam que o resultado da simulação não é confiável.

5.5 Testes de comportamento automatizados

A empresa, em um esforço recente, vem buscando implementar uma grande variedade de testes de comportamento utilizando o *framework Cucumber*. De forma a garantir que os testes sejam executados com frequência e permitam a detecção de problemas no produto, também foi desenvolvida uma *pipeline* no GoCD para sua execução.

Esta *pipeline* é composta de três estágios:

1. Atualização e compilação dos testes;
2. *Setup* do Titanium onde os testes serão executados;
3. Execução dos testes e apresentação do resultado.

No estágio 1 são realizadas as etapas de atualização do repositório local, com todas as alterações realizadas desde a última execução, e compilação dos testes. Esta etapa irá atualizar e compilar a aplicação principal do Titanium, os arquivos necessários para a execução dos testes e os casos de teste. Devido a necessidade da compilação da aplicação executada no Titanium, esta acaba sendo uma etapa lenta do processo, impedindo que os testes sejam executados com maior frequência.

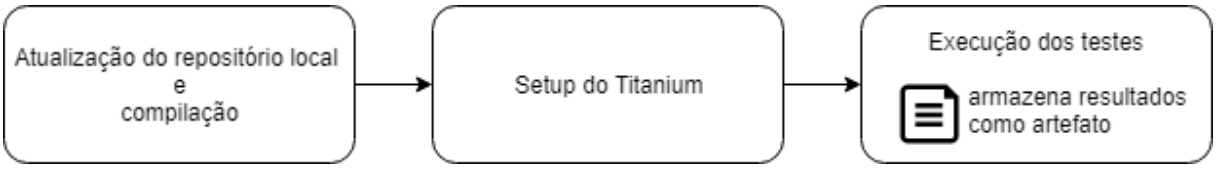
Para a execução dos testes são necessárias configurações no Titanium, estas tarefas são executadas no estágio 2. A primeira tarefa consiste em garantir que a aplicação não esteja sendo executada no computador de bordo, para isso são executados comando para encerrar a execução da aplicação. Já a segunda tarefa consiste em montar a imagem remota da pasta onde os testes foram compilados no Titanium, para isso são realizados comando para criação e montagem da pasta.

Por fim, a execução dos testes é realizada no estágio 3. Para a execução e apresentação do resultado dos testes foi necessária a criação de um *script* que itere sobre cada caso de teste realizando sua execução e armazenando seu resultado. Anteriormente já existia um *script* que facilitava a execução de testes individuais, chamado *run_ behaviour_ tests*, dessa forma o novo *script* faz uso do *script* já existente percorrendo todos os casos de teste.

Para determinar o resultado da execução, é verificado o resultado de cada caso de teste, caso um ou mais testes falhem, a *pipeline ira indicar falha*. Para cada teste é gerado um arquivo de checagem de resultado, indicando falha ou sucesso e, em caso de falha, em qual etapa ocorreu. Os resultados que apresentam falhas serão adicionados como artefato da *pipeline*, permitindo sua coleta para análise, um exemplo é apresentado no anexo B, onde o teste executado falhou no terceiro passo por estar esperando uma determinada tela e esta não aparecer.

Ao final da execução a lista de testes bem sucedidos e a lista de teste que apresentaram falha serão apresentados. Na figura 10 é apresentado o diagrama dos estágio da *pipeline*.

Figura 10 – Diagrama dos testes de comportamento.



Fonte: Autor.

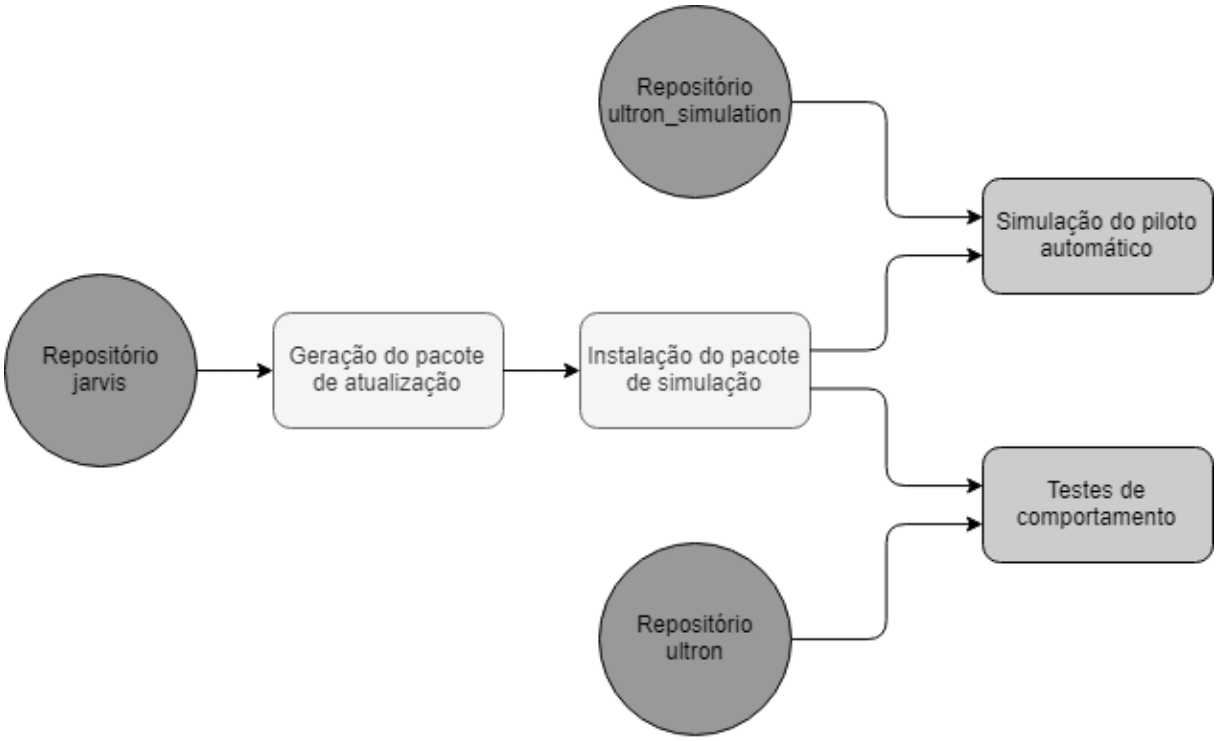
Atualmente, após a execução desta *pipeline* a grande maioria dos testes está falhando. Isso acontece devido aos testes, em sua criação, esperarem uma determinada configuração para seu funcionamento. Como os testes eram criados e executados por desenvolvedores diferentes em equipamentos diferentes, estes estavam dependentes da configuração utilizada pelo desenvolvedor em seu Titanium. Com o uso da ferramenta desenvolvida, os testes serão executados sempre no mesmo equipamento, fazendo assim, que cada teste deva realizar a configuração do equipamento conforme sua necessidade.

5.6 Ferramenta obtida

Com o objetivo de automatizar a execução dos testes e simulações a última etapa consiste na união das quatro *pipelines* desenvolvidas em uma ferramenta capaz de gerar e instalar o pacote de atualização e executar os testes e simulações.

A solução inicial consistia na construção de uma *pipeline* onde a geração e instalação fossem passos únicos, ou seja, os testes de comportamento e a simulação do piloto seriam executadas no mesmo Titanium. Na imagem 11 é apresentado o diagrama das *pipelines* de acordo com esta solução.

Figura 11 – Diagrama da primeira versão da ferramenta.



Fonte: Autor.

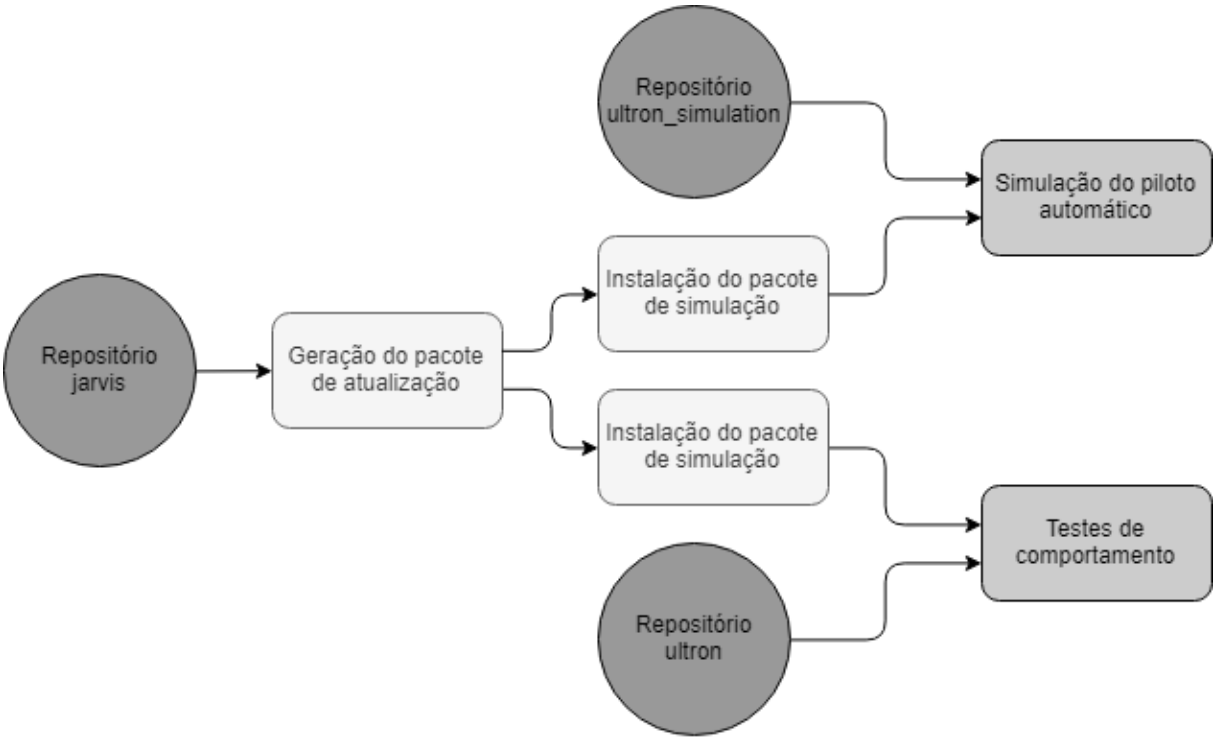
Neste formato, a simulação e os testes seriam executados em paralelo, porém, como deveriam compartilhar o computador de bordo, uma poderia causar falhas na outra. Isso ocorreria, pois, neste formato, as *pipelines* executariam seus estágios de forma alternada, fazendo com que o computador de bordo fosse configurado por ambas.

Uma segunda solução seria de forma sequencial, ou seja, a simulação apenas ser executada após os testes de comportamento ou vice versa. Apesar de solucionar o problema do paralelismo das atividades, esta solução não podê ser implementada devido a restrições do GoCD. A ferramenta de integração não permite que uma *pipeline* dependente de outra, já executada, seja iniciada caso a etapa anterior falhe. Ou seja, neste caso, se a primeira *pipeline* executada apresentasse falha, a seguinte não seria executada.

Buscando solucionar o problema do paralelismo foi proposta a solução da figura 12. Nesta solução são necessários dois computadores de bordo, permitindo sua execução em paralelo sem uma interferir na outra. Neste caso, foi necessária a duplicação da *pipeline* de instalação do pacote de atualização, pois, cada *pipeline* realiza a instalação em apenas

um computador de bordo.

Figura 12 – Diagrama idealizado para a ferramenta.



Fonte: Autor.

Esta solução é a ideal, pois, com a adição de um segundo agente para execução, permite que as *pipelines* em paralelo sejam executas ao mesmo tempo, sem uma interferir na outra. Porém, tem como ponto negativo a necessidade de um segundo computador de bordo.

Outro benefício existente dessa solução é a possibilidade de expansão e melhoria no processo. Para o caso dos testes de comportamento, uma vez que sejam solucionados os problemas nos testes e encontrada uma forma de acelerar o processo de compilação, como por exemplo o uso de ferramentas como o *ccache* [35], existe a possibilidade de executar os testes antes da integração de novos conteúdos nos repositórios.

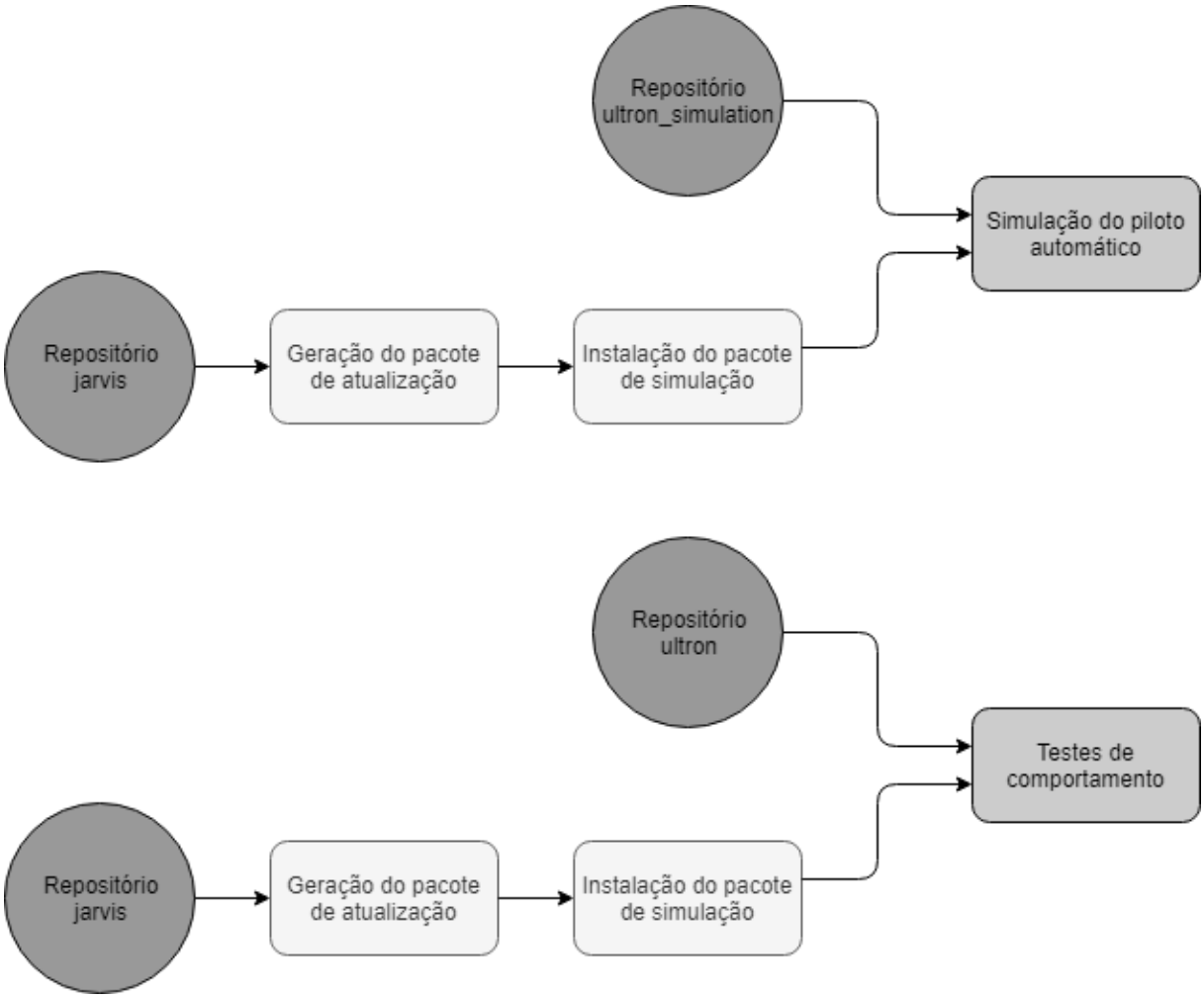
E, para o caso da simulação, a possibilidade de executar novas simulações de forma sequencial. Para isso o início da simulação seguinte deve ser realizado assim que o uso do computador de bordo for finalizado, ou seja, antes da checagem do resultado da simulação, o que permitiria a continuação da execução mesmo em caso de falha da simulação anterior. Porém a implementação dessa solução ainda deve ser trabalhada de forma que possíveis falhas durante as simulações não inviabilizem a execução das subsequentes.

5.6.1 Solução implementada

No momento, as *pipelines* ainda não estão completas, faltando a correção dos testes de comportamento e algumas atividades previstas para trabalhos futuros. A solução apresentada somente poderá ser implementada quando estas etapas forem concluídas, pois o uso do segundo computador de bordo apenas será permitido neste ponto, devido ao aumento dos custos do projeto. Logo, existe apenas um Titanium disponível.

Com isso, a solução foi adaptada de forma provisória para que funcione da forma mais próxima possível da solução prevista. A solução implementada é apresentada na figura 13.

Figura 13 – Diagrama da ferramenta implementada.



Fonte: Autor.

Nesta solução a ferramenta foi separada em duas, uma para as simulações e outra para os testes de comportamento. Para que não ocorram interferências entre elas, a execução diária automática foi definida para turnos opostos do dia. Com isso será possível

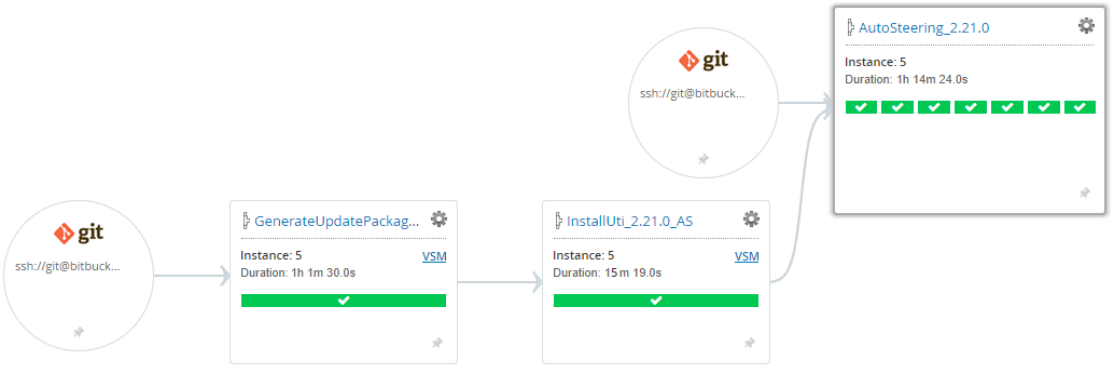
a realização de testes e a validação do comportamento das *pipelines*. Uma vez que a solução esteja estável e finalizada a solução será alterada para o apresentado na figura 12.

6 Análise dos resultados

Durante o desenvolvimento do trabalho, buscou-se atender o objetivo, apresentado no capítulo 3, de buscar a facilitação e ampliação dos testes e simulações no produto, buscando melhorias em sua qualidade. Para isso foram desenvolvidas ferramentas que possibilitam a execução de testes e simulações de forma automatizada. Dessa forma, os objetivos a curto prazo foram alcançados e, em conjunto com melhorias propostas durante o desenvolvimento, também serão alcançados os objetivos a longo prazo, que dizem respeito a melhoria na qualidade do produto.

Ao final do projeto de fim de curso foram desenvolvidas duas ferramentas, uma voltada à simulação do piloto automático e outra aos testes de comportamento. Estas ferramentas serão, futuramente, reduzidas a uma única ferramenta capaz de executar ambas as atividades. As ferramentas foram criadas utilizando o GoCD e são apresentadas nas imagens 14 e 15, respectivamente.

Figura 14 – Ferramenta para execução da simulação do piloto automático.

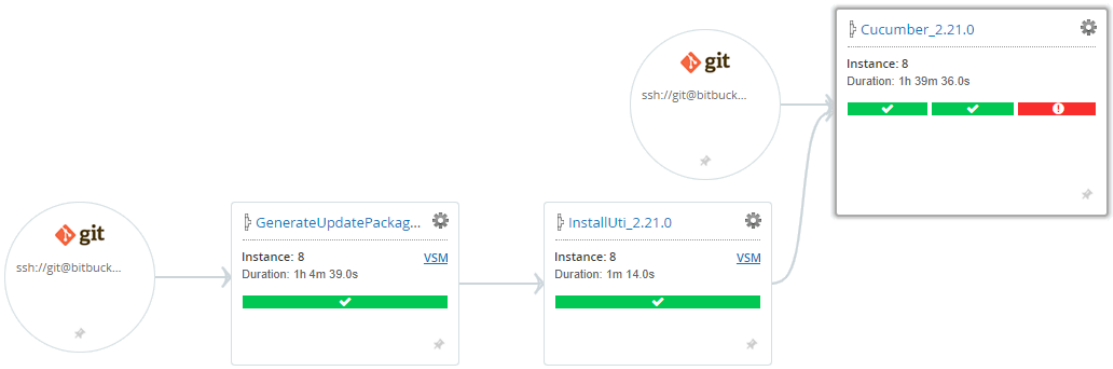


Fonte: Captura de tela do GoCD.

Apesar do projeto ainda não estar em sua fase final, as ferramentas já foram capazes de detectar problemas no produto e nas ferramentas de simulação. Um ponto que chama a atenção foi a capacidade de detectar problemas na geração e instalação do novo pacote de atualização do sistema, apresentado no item 5.2.

Durante todo o processo de desenvolvimento, as *pipelines* de geração e instalação foram capazes de reproduzir o comportamento real e, assim, detectar a existência de *bugs* com maior rapidez. Este fator indica, que uma vez que a ferramenta esteja finaliza e executando automaticamente, ela será capaz de ajudar no processo de desenvolvimento, principalmente garantindo que, pelo menos nas condições testadas e simuladas, as modificações e atualizações do código de produção não trouxeram novos problemas.

Figura 15 – Ferramenta para execução dos testes de comportamento.



Fonte: Captura de tela do GoCD.

6.1 Resultados esperados

Como objetivo a longo prazo, o projeto busca a redução de *bugs* e, por consequência, a melhoria na qualidade do produto. Com a facilitação e automação da execução dos testes e simulações, se espera que novos casos sejam implementados, de forma a aumentar a cobertura de testes e simulações. Estes aumento refletirá em possíveis problemas serem detectados mais facilmente e de forma mais rápida, reduzindo o custo por eles gerados [36].

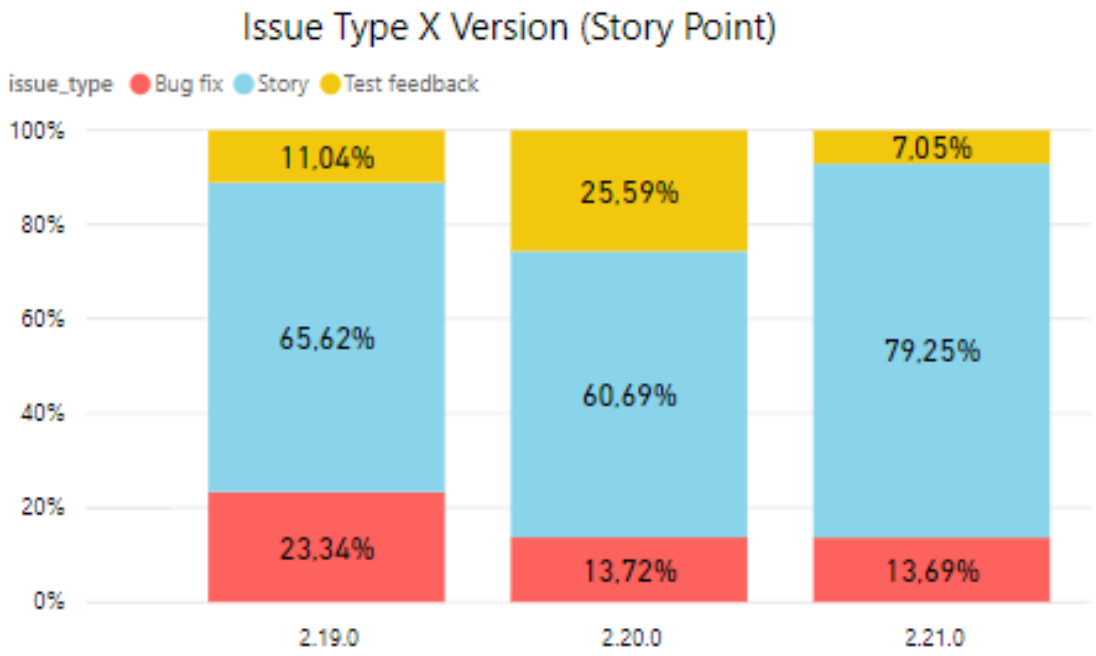
Para o processo de desenvolvimento, se espera um aumento de tarefas relacionadas aos dados obtidos pelos testes, conhecidas como *test feedback*, e uma redução nas tarefas de correção de problemas, conhecidas como *bug fix*. Isso é esperado, pois indica que os testes realizados dentro do setor de desenvolvimento estão sendo capazes de detectar problemas antes que saiam dele. Na figura 16 são apresentados os dados da porcentagem de cada tipo de tarefa presente nas três últimas versões.

Na figura, as tarefas do tipo *story* são as tarefas que contemplam o desenvolvimento de novas funcionalidades. Assim, quanto maior a porcentagem do tipo *story* maior estará sendo o aproveitamento da equipe de desenvolvimento na melhoria do produto. As tarefas do tipo *test feedback* trazem correções de problemas, porém estes, apesar de reduzirem a porcentagem de *story*, possuem um custo menor para execução, pois representam bugs detectados cedo. Já tarefas de *bug fix* são as mais custosas para a empresa, pois representam os problemas detectados pela equipe de QA ou reportados pelos clientes.

6.2 Trabalhos futuros

Com o objetivo de melhorar a ferramenta e tornar possível que ela traga maiores benefícios para o processo de desenvolvimento da *Hexagon Agriculture*, foram elencados trabalhos futuros que envolvem a finalização e melhorias do projeto. São eles:

Figura 16 – Relação de tipo de tarefa por versão.



Fonte: Adaptado de documentos internos da *Hexagon Agriculture*.

- Geração do pacote de atualização:
  - Desenvolver o estágio de armazenamento do pacote de atualização, conforme explicado no item 5.2.
- Simulação do piloto automático:
  - Melhoria na checagem do resultado da simulação: atualmente a checagem realizada leva em consideração todos os caminhos percorridos, porém, se deseja realizar esta checagem de forma individual para cada caminho percorrido, de forma a poder detectar os problemas específicos de cada tipo de rota;
  - Finalização das melhorias no reprodutor de *logs* de campo (*Kilgrave*), como explicado no item 5.1.3;
  - Expansão das simulações para outros produtos da empresa, buscando aumentar a cobertura de testes.
- Testes de comportamento:
  - Correção dos testes de comportamento que estão apresentando falha devido a erros de configuração.
- União das ferramentas de teste e simulação, confirme explicado no item 5.6.

Com a realização desses itens, se espera uma melhora na qualidade da ferramenta e que ela se torne cada vez mais útil no processo de desenvolvimento, buscando auxiliar cada vez mais os desenvolvedores e os gerentes de projeto e também possibilitando a melhora contínua da qualidade do produto.

7 Considerações finais

No decorrer do projeto, diversas dificuldades foram encontradas, dificuldades estas, que geraram a necessidade de estudos da teoria e do sistema e suas características. Estes problemas foram encontrados em praticamente todas as etapas do processo de desenvolvimento, fazendo com que a capacidade de pesquisa e compreensão das informações e do problema fossem um item indispensável para solução das dificuldades e para o desenrolar do projeto. Estes problemas foram de diversas naturezas, compreendendo desde a dificuldade na utilização das ferramentas e *frameworks* até testes corrompendo o sistema operacional do computador de bordo.

Durante todo o processo, os conhecimentos adquiridos durante o curso de engenharia de controle a automação foram utilizados. A estrutura do curso proporciona um vasto conhecimento em diversas áreas, o que garantiu ao autor uma maior facilidade no estudo e compreensão de diversos aspectos do projeto.

Porém, principalmente no que diz respeito a área de desenvolvimento de *software*, existe uma lacuna de conhecimento em aspectos de extrema importância. Durante toda a experiência na área de desenvolvimento as áreas de arquitetura de *software*, métodos de desenvolvimento ágeis, qualidade de *software* e testes de *software* foram essenciais, porém abordadas apenas de sintética durante o curso.

O trabalho na *Hexagon Agriculture* possibilitou um grande crescimento profissional e pessoal. Durante o período o autor teve a possibilidade de trabalhar em diversos projetos e em diversas atividades, o que garantiu a aquisição de uma vasta gama de conhecimento. Outra importante visão adquirida na empresa foi a de trabalho em equipe dentro de um ambiente profissional e não mais educacional.

Um dos principais ganhos no desenvolvimento do trabalho de fim de curso, além de todo o conhecimento adquirido para seu completo desenvolvimento foi o da necessidade de um bom planejamento e da manutenção da qualidade dos sistemas. Estes aspectos trazem benefícios para a empresa, para os clientes e também para os desenvolvedores. Para os desenvolvedores se nota uma melhora na qualidade de vida durante o processo, onde, não é mais necessário o constante chaveamento entre o desenvolvimento de melhorias do produtos para a correção de *bugs*.

Referências

1 HEXAGON AGRICULTURE. *Soluções*. Acessado em 12/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions>. Citado 2 vezes nas páginas 15 e 20.

2 HEXAGON AGRICULTURE. *Trabalhe Conosco*. Acessado em 14/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/careers>. Citado na página 19.

3 HEXAGON. *Our Story*. Acessado em 14/06/2019. Disponível em: <https://hexagon.com/our-story>. Citado na página 19.

4 HEXAGON. *Divisions*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions>. Citado na página 19.

5 HEXAGON. *Geospatial*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/geospatial>. Citado na página 19.

6 HEXAGON. *Geosystems*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/geosystems>. Citado na página 19.

7 HEXAGON. *Manufacturing Intelligence*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/manufacturing-intelligence>. Citado na página 19.

8 HEXAGON. *Mining*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/mining>. Citado na página 19.

9 HEXAGON. *Positioning Intelligence*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/positioning-intelligence>. Citado na página 19.

10 HEXAGON. *PPM*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/ppm>. Citado na página 19.

11 HEXAGON. *Safety & Infrastructure*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/safety-infrastructure>. Citado na página 19.

12 HEXAGON. *Agriculture*. Acessado em 16/06/2019. Disponível em: <https://hexagon.com/about/divisions/agriculture>. Citado na página 20.

13 HEXAGON AGRICULTURE. *Sobre*. Acessado em 16/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/about>. Citado na página 20.

14 HEXAGON AGRICULTURE. *Cultivation*. Acessado em 17/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions/cultivation>. Citado na página 20.

15 HEXAGON AGRICULTURE. *Harvesting*. Acessado em 17/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions/harvesting>. Citado na página 20.

16 HEXAGON AGRICULTURE. *OEM*. Acessado em 17/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions/oem>. Citado na página 20.

17 HEXAGON AGRICULTURE. *HxGN AgrOn Piloto Automático*. Acessado em 17/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions/oem/machine-automation/auto-steering>. Citado na página 20.

18 HEXAGON AGRICULTURE. *PS-PED-01 Projeto e desenvolvimento*. 2018. Citado na página 23.

19 HEXAGON AGRICULTURE. *PS-RAO-02 Projeto e desenvolvimento*. 2019. Citado na página 23.

20 NETO, A. C. D. Introdução a teste de software. *Engenharia de Software Magazine*. Citado na página 27.

21 HEXAGON AGRICULTURE. *Plano de Qualidade SW*. 2016. Citado 2 vezes nas páginas 27 e 28.

22 CRAIG, S. P. J. R. D. *Systematic Software Testing*. [S.l.]: Artech House, 2002. Citado 2 vezes nas páginas 27 e 28.

23 SHORT, M. J. P. M. Hardware in the loop simulation of embedded automotive control systems. *IEEE*, 2005. Citado na página 29.

24 CUCUMBER LTD. *Cucumber*. Acessado em 30/06/2019. Disponível em: <https://cucumber.io/>. Citado na página 30.

25 CUCUMBER LTD. *BDD Overview*. Acessado em 30/06/2019. Disponível em: <https://cucumber.io/docs/bdd/overview/>. Citado na página 30.

26 CUCUMBER LTD. *Gherkin Reference*. Acessado em 30/06/2019. Disponível em: <https://cucumber.io/docs/gherkin/reference/>. Citado na página 30.

27 COPPELIA ROBOTS. *V-Rep*. Acessado em 30/06/2019. Disponível em: <http://www.coppeliarobotics.com/>. Citado na página 30.

28 THOUGHTWORKS INC. *GoCD*. Acessado em 30/06/2019. Disponível em: <https://www.gocd.org/>. Citado na página 31.

29 THOUGHTWORKS INC. *GoCD FEATURES*. Acessado em 30/06/2019. Disponível em: <https://www.gocd.org/why-gocd/>. Citado na página 31.

30 HEXAGON AGRICULTURE. *HxGN AgrOn Ti5 / Ti7*. Acessado em 22/06/2019. Disponível em: <https://hexagonagriculture.com/pt-br/solutions/oem/embedded-electronics/ti5--ti7>. Citado na página 33.

31 CROUCH, S. *Developing scientific applications using a Model-View-Controller approach*. Acessado em 25/06/2019. Disponível em: <https://www.software.ac.uk/developing-scientific-applications-using-model-view-controller-approach>. Citado na página 35.

32 CANONICAL LTD. *Canplayer*. Acessado em 23/06/2019. Disponível em: <http://manpages.ubuntu.com/manpages/bionic/man1/canplayer.1.html>. Citado na página 36.

33 CANONICAL LTD. *Cansend*. Acessado em 23/06/2019. Disponível em: <http://manpages.ubuntu.com/manpages/bionic/en/man1/cansend.1.html>. Citado na página 36.

34 SOFTWARE FREEDOM CONSERVANCY. *Git*. Acessado em 23/06/2019. Disponível em: <https://git-scm.com/>. Citado na página 44.

35 TRIDGELL, A. *Compiler cache*. Acessado em 27/06/2019. Disponível em: <https://ccache.dev/>. Citado na página 50.

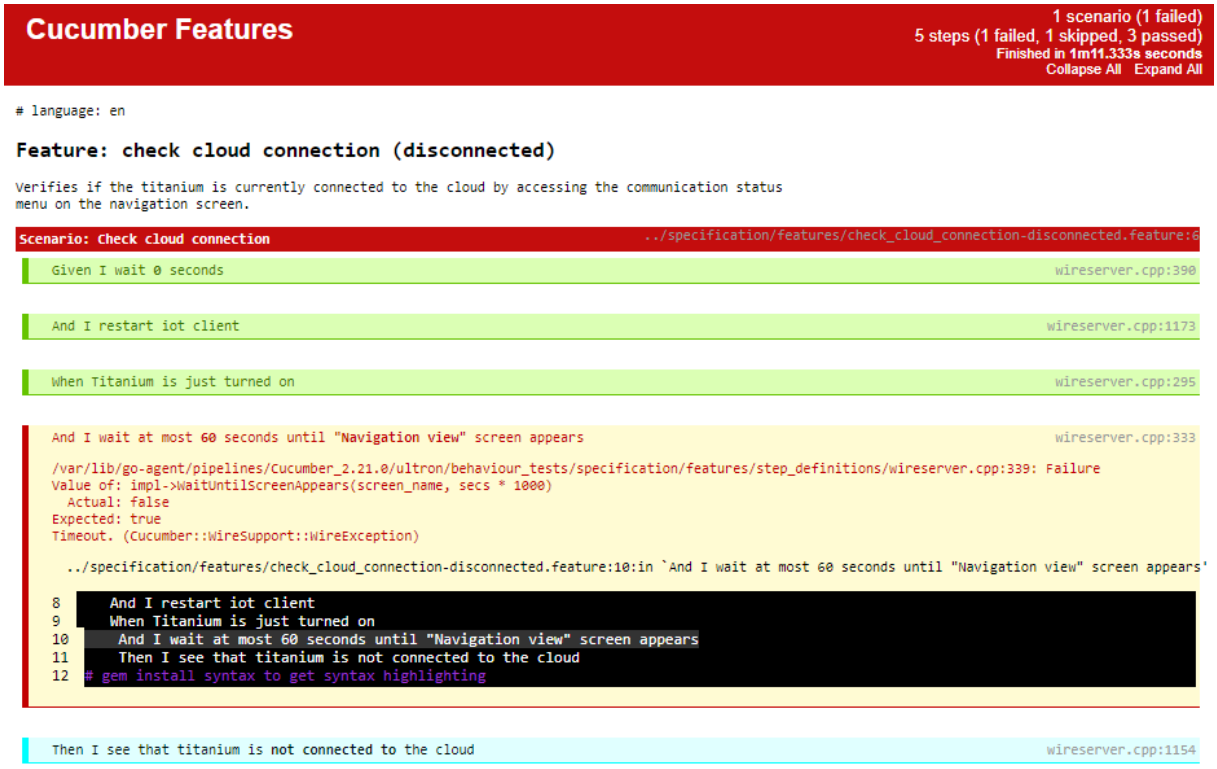
36 PATTON, R. *Software Testing*. [S.l.]: Sams, 2000. Citado na página 54.

Anexos

ANEXO A – Titanium.

Fonte: Manual do usuário - *displays* Ti5 e Ti7.

ANEXO B – Exemplo de resultado dos testes de comportamento.



Fonte: Resultado de teste de sistema desenvolvido pela *Hexagon Agriculture* através do *framework* Cucumber.