

UNIVERSIDADE DE CAXIAS DO SUL

ROGÉRIO SALES GUIMARÃES

**ESTUDO DE AUTOMAÇÃO DE TESTES FUNCIONAIS E INTEGRAÇÃO
CONTÍNUA PARA UM SISTEMA DE GESTÃO**

CAXIAS DO SUL

2016

ROGÉRIO SALES GUIMARÃES

**ESTUDO DE AUTOMAÇÃO DE TESTES FUNCIONAIS E INTEGRAÇÃO
CONTÍNUA EM UM SISTEMA DE GESTÃO**

Trabalho de Conclusão de Curso do Título de Bacharel em Ciência da Computação pela Universidade de Caxias do Sul, como requisito parcial para obtenção do título de Bacharel.

Orientador Msc. Iraci Cristina da Silveira de Carli

CAXIAS DO SUL

2016

RESUMO

Controlar e garantir a qualidade de um software é uma atividade extremamente complexa devido às variantes e problemas aos quais o processo de desenvolvimento está exposto, e cada vez mais as empresas são demandadas na criação de sistemas de qualidade em um curto espaço de tempo. A automação de testes tem sido uma alternativa para as empresas reduzirem custos e maximizarem a qualidade do produto entregue. Automação permite a execução de uma gama muito grande de testes em curto espaço de tempo, o que seria inviável se fossem executados de forma manual. O objetivo deste trabalho é propor um processo de automação com integração contínua para a empresa alvo do estudo. Na primeira fase deste trabalho foram levantadas as necessidades da empresa, definido um processo que contemple as atividades de automação e realizada a escolha de uma ferramenta para automatizar os testes. Na segunda etapa foi realizada a criação de um ambiente de protótipo para utilização dessa ferramenta, configurado processo de integração contínua e realizada a integração entre as ferramentas Ranorex utilizada para automação e o Jenkins que auxilia nas atividades de integração contínua.

Palavras-chave: Automação de Teste de Software, Teste funcional, Teste de software, Integração Contínua, Ranorex, Jenkins.

ABSTRACT

To control and to ensure the software's quality is an extremely complex activity due to the variants and problems to which the development process is exposed, and more and more companies are required to create quality systems in a short time. Test automation has been an alternative for companies to reduce costs and maximize the quality of delivered products. The automation allows the execution of a very large range of tests in a short time, which would be unfeasible if they were performed manually. The objective of this work is to propose an automation process with continuous integration for the company that was target of this study. In the first phase of this work, the needs of the company and the process were defined, contemplating the automation activities and it was performed the choice of a tool to automate tests. In the second stage, it was created the prototype environment for the use of this tool, configured continuous integration process and realized the integration between the tools Ranorex used for automation and Jenkins that assists in the activities of continuous integration.

Keywords: Software Test Automation, Functional test, Software testing, Continuous Integration, Ranorex, Jenkins.

Lista de Figuras

Figura 1 - Exemplo de testes de validação	17
Figura 2 - Processo de testes.....	18
Figura 3 - Representação teste caixa branca.....	21
Figura 4 - Ilustração teste caixa preta	21
Figura 5 - Exemplo particionamento por equivalência.....	23
Figura 6 - Exemplo análise valor limite.....	23
Figura 7 - Exemplo grafo causa e efeito.....	24
Figura 8 - Aspectos dos testes de unidade	26
Figura 9 - Processo de desenvolvimento da empresa alvo.....	33
Figura 10 - Processo de testes da empresa alvo	33
Figura 11 - Atividades Etapa Planejar Testes	34
Figura 12 - Atividades Preparar Testes.....	35
Figura 13 - Atividades Executar Teste Integrado	36
Figura 14 - Atividades Executar Teste Sistemico	36
Figura 15 - Planejar Testes com atividades de automação.....	38
Figura 16 - Preparar Testes com atividades de automação.....	38
Figura 17 - Executar Teste Integrado com atividades de automação	39
Figura 18 - Executar Teste Sistemico com atividades de automação.....	39
Figura 19 - Processo para automação de testes.....	40
Figura 20 - Ilustração da ferramenta Selenium	46
Figura 21 - Ilustração ferramenta Ranorex.....	47
Figura 22 - Ilustração ferramenta TestComplete	47
Figura 23 - Ilustração ferramenta Sahi	48
Figura 24 - Ilustração ferramenta IBM Rational.....	48
Figura 25 - Infraestrutura do Protótipo	53
Figura 26 - Comunicação com o Banco de Dados	55
Figura 27 - Rollback e Update do Banco de Dados	56
Figura 28 - Estrutura de Diretórios	57
Figura 29 - Processo de compilação do SyncProgress.....	58
Figura 30 - Relatório de Acompanhamento SyncProgress	59
Figura 31 - Principais Ferramentas do Protótipo.....	60
Figura 32 - Ferramentas de Apoio.....	60
Figura 33 - Caso de Teste no Kanoah	62

Figura 34 - Script de Teste	63
Figura 35 - Criação de Projetos Ranorex	65
Figura 36 - Estrutura de Diretórios dos Projetos do Ranorex	66
Figura 37 - Criação de Gravação Ranorex.....	67
Figura 38 - Edição de Records	68
Figura 39 - Seleção do Tipo de Plataforma para Gravação	69
Figura 40 - Criação de Validation na Ranorex.....	70
Figura 41 - Edição de Componente do Repositório.....	71
Figura 42 - Variáveis na Ranorex.....	72
Figura 43 - Alteração de Componente Form no Ranorex.....	73
Figura 44 - Script de automação	74
Figura 45 - Scripts Particionados	75
Figura 46 - Suit Test Ranorex	76
Figura 47 - Caso de Teste Ranorex	76
Figura 48 – Nível de erro do Casos de Teste.....	77
Figura 49 - Importação de Data-Driven Ranorex.....	78
Figura 50 - Linha de Comando para Iniciar o Jenkins	79
Figura 51 - Configuração do Jenkins.....	81
Figura 52 - Divisão dos Jobs no Jenkins	82
Figura 53 - Jobs de Atualização de Fontes	82
Figura 54 - Log de Atualização de Diretórios	83
Figura 55 - Jobs de Compilação de Fontes.....	84
Figura 56 - Jobs Atualização de Servidores	84
Figura 57 - Jobs de Banco de Dados	85
Figura 58 – Casos de Usos Ferramenta Controle de Bugs.....	87
Figura 59 - Características da Ferramenta de Controle de Bugs	88
Figura 60 - Arquivo de Conexão com o Banco de Dados	88
Figura 61 - Tabelas Ferramenta Controle de Bugs	89
Figura 62 - Diagrama de classes Ferramenta de Bugs	91
Figura 63 - Diretórios de Integração.....	92
Figura 64 - Diretórios dos Scripts de integração	93
Figura 65 - Script do Plano de Testes	94
Figura 66 - Script de Integração para Caso de Teste.....	95

Lista de Tabelas

Tabela 1 - Critérios de avaliação das ferramentas de automação	44
Tabela 2: Pontuação da avaliação das ferramentas de automação	49

Sumário

1	INTRODUÇÃO	11
1.1	CONTEXTO	11
1.2	PROBLEMA DE PESQUISA	12
1.3	QUESTÃO DE PESQUISA	14
1.4	OBJETIVOS	14
1.4.1	<i>Objetivos gerais</i>	14
1.4.2	<i>Objetivos Específicos</i>	15
1.5	METODOLOGIA	15
1.6	ESTRUTURA DO TEXTO	16
2	TESTE DE SOFTWARE	17
2.1	PROCESSO DE TESTE	18
2.2	ABORDAGENS DE TESTES	20
2.2.1	<i>Teste Caixa Branca</i>	20
2.2.2	<i>Teste Caixa preta</i>	21
2.2.2.1	PARTICIONAMENTO POR EQUIVALÊNCIA	22
2.2.2.2	Análise valor limite	23
2.2.2.3	Grafo de causa e efeito	24
2.3	TIPOS DE TESTES	25
2.3.1	<i>Testes de Unidade</i>	25
2.3.2	<i>Teste de integração</i>	26
2.3.3	<i>Teste de Sistema</i>	27
2.3.3.1	Teste Funcional	27
2.3.3.2	Teste de Regressão	28
2.4	TESTES AUTOMATIZADOS	29
2.5	INTEGRAÇÃO CONTÍNUA	30
3	LEVANTAMENTO DO PROCESSO DE TESTE DA EMPRESA ALVO	31
3.1	FERRAMENTAS DE APOIO AO PROCESSO	31
3.2	PERFIL DA EQUIPE	32
3.3	PROCESSO DE TESTE	32
3.3.1	Planejar Testes	34
3.3.2	Preparar Testes	34

3.3.3	Executar Teste integrado.....	35
3.3.4	Executar Teste Sistemico	36
4	PROCESSO PARA AUTOMAÇÃO DE TESTES FUNCIONAIS	37
5	AVALIAÇÃO E DEFINIÇÃO DAS FERRAMENTAS	41
5.1	PLANEJAMENTO PARA AVALIAÇÃO DAS FERRAMENTAS	41
5.2	APRESENTAÇÃO DAS FERRAMENTAS.....	45
5.2.1.1	Selenium	46
5.2.1.2	Ranorex Studio.....	47
5.2.1.3	TestComplete	47
5.2.1.4	Sahi	48
5.2.1.5	IBM Rational Functional Tester	48
5.3	APLICAÇÃO DOS CRITÉRIOS DE AVALIAÇÃO E DEFINIÇÃO DAS FERRAMENTAS	48
6	APLICAÇÃO DA PROPOSTA.....	52
6.1	AMBIENTE DE PROTÓTIPO	52
6.1.1	<i>Sistema</i>	53
6.1.2	<i>Banco de Dados</i>	55
6.1.3	<i>Estrutura de Diretórios</i>	56
6.1.4	<i>Processo de Compilação Progress</i>	57
6.1.5	<i>Instalação de Ferramentas</i>	59
6.2	AUTOMAÇÃO DE CASOS DE TESTE.....	61
6.2.1	Casos de testes	61
6.2.2	Scripts de automação	64
6.2.2.1	Estrutura dos Projetos de Automação	64
6.2.2.2	Gravações dos Testes	67
6.2.2.3	Criação dos casos de teste.....	75
6.3	INTEGRAÇÃO CONTINUA	79
6.3.1	Configuração da Ferramenta	79
6.3.2	Estrutura de Jobs.....	81
7	INTEGRAÇÃO DE FERRAMENTAS.....	86
7.1	CRIAÇÃO FERRAMENTA DE CONTROLE DE BUGS.....	86
7.2	INTEGRAÇÃO JENKINS X RANOREX.....	92

7.3 AVALIAÇÃO DA INTEGRAÇÃO DE FERRAMENTAS	96
8 CONCLUSÃO	99
REFERÊNCIAS.....	102

1 INTRODUÇÃO

Neste capítulo, serão apresentados o contexto do trabalho, os principais conceitos, o problema e a questão de pesquisa, os objetivos, metodologia e estrutura do texto.

1.1 CONTEXTO

O processo de desenvolvimento de software pressupõe dois momentos bem distintos. O primeiro é a fase de coleta de informações de negócio e o planejamento da arquitetura do software. O segundo é caracterizado pela existência de um componente computacional (BARTIÉ, 2002).

Segundo (BARTIÉ, 2002), em 1979 Myers já definia testes como um processo de executar um programa com o objetivo de encontrar defeitos. É mais fácil provar que algo está funcionando, do que provar que algo não está funcionando. Se olharmos os testes apenas pela perspectiva positiva, o número de situações a serem testadas é reduzida, ao contrário de testes que são planejados com o objetivo de provar a não-adequação de algo. Quando se utiliza essa segunda perspectiva, amplia-se os cenários possíveis de utilização do que foi desenvolvido, nesse caso, passa-se a considerar um número maior de situações, tanto positivas quanto negativas (BARTIÉ, 2002, MOLINARI, 2010).

Apesar do conceito de Zero-defeito ser algo inatingível, devido à complexidade envolvida, e pelo número altíssimo de situações existentes, a qualidade de software trabalha o Zero-defeito representando a não-tolerância a erros. Com isso o objetivo é criar mecanismos de inibição a falhas (BARTIÉ, 2002).

Controlar a qualidade de um software é um grande desafio, não só pela complexidade das atividades do processo de desenvolvimento, mas também pelas questões humanas, técnicas, burocráticas, de negócio e políticas envolvidas. Como o desenvolvimento de software possui inúmeras atividades, sendo que as mesmas estão expostas a diversos problemas, o produto resultante, na maioria das vezes acaba saindo diferente do que foi planejado (PRESSMAN, 2011, BARTIÉ, 2002, MOLINARI, 2010).

Os testes manuais se utilizam de recursos humanos, para realizar todos os procedimentos de validação. Uma vez que um artefato ou funcionalidade é alterada, todo o processo de verificação deve ser refeito manualmente. Esse tipo de teste é trabalhoso, e na maioria das vezes não traz os resultados esperados (BARTIÉ, 2002, MOLINARI, 2010).

A automação de testes tem papel estratégico, uma vez que permite a realização de diversos ciclos repetidos de testes, a qualquer momento e com pouco esforço, garantindo que passos importantes não sejam ignorados por falha humana. A automação utiliza-se de ferramentas de testes, que tem por finalidade simular usuários ou atividades humanas, de forma que não requeiram procedimentos manuais no processo de execução de testes. Com isso, permite que a equipe de testes seja alocada para atividades mais produtivas, como por exemplo, validação de regras de negócio, ou testes em partes que não podem ser automatizadas. À medida que os testes são reexecutados, existe o ganho de tempo, controle, confiabilidade e redução de esforço, assim como uma melhor qualidade do produto final (BARTIÉ, 2002, MOLINARI, 2008, MOLINARI, 2010).

Apesar da automação permitir executar os testes a qualquer momento ela não controla a integração das alterações que são realizadas no software, dessa forma não se tem um feedback instantâneo sobre o impacto causado por essas mudanças, sendo que falhas só serão encontradas quando os testes forem reexecutados.

A integração contínua vem com o objetivo de suprir essa necessidade. Cada integração é verificada através de builds¹ automatizados para detectar erros de integração o mais rápido possível, além de permitir que os testes automatizados sejam disparados a cada nova integração, garantindo a estabilidade do software e dando maior segurança para que a equipe realize alterações (FOWLER, 2006).

1.2 PROBLEMA DE PESQUISA

A crescente utilização de sistemas informatizados, em todas as áreas da atividade humana, aumenta a demanda por qualidade e produtividade. Com isso

¹ Em desenvolvimento de software, o termo build é normalmente utilizado para se referir ao processo de compilação de código-fonte, ou seja, converter o código-fonte em artefatos de software autônomo que podem ser executados pelo computador.

também aumenta a pressão para o desenvolvimento de sistemas com qualidade e em curto espaço de tempo. Essa pressão gera nas organizações a necessidade de novos processos para garantir a qualidade e confiabilidade dos seus softwares.

Inúmeros estudos apontam que é indispensável a existência de um processo de testes de software efetivo. Dessa forma, quanto mais eficiente for a aplicação de testes, menor será o custo de reparo e maior será a qualidade do produto.

Testes manuais executados em ambientes complexos são uma tarefa dispendiosa e cansativa, e na maioria das vezes não abrange todas as situações que deveriam ser testadas. Muitas empresas se utilizam apenas de testes manuais, a fim de garantir a qualidade. Nesse contexto é necessária a aplicação de processos que reduzam o esforço e garantam uma maior cobertura dos cenários de testes, tanto positivos quanto negativos.

A empresa alvo deste estudo possui um sistema de gestão desenvolvido em Progress 4GL² com partes desenvolvidas em Java³.

O sistema dessa empresa é focado na área de gestão de planos de saúde, e possui um total de 28 módulos. Os cadastros existentes no sistema normalmente dividem-se em cinco programas, cada um responsável por uma funcionalidade diferente. As funcionalidades existentes geralmente são: pesquisa, inclusão, exclusão, alteração, listagem ou relatório.

Para realização dos testes no produto, a empresa conta com uma equipe de qualidade composta atualmente por sete integrantes. Essa equipe aplica testes funcionais, com estratégia de caixa preta nas etapas de teste integrado e sistêmico. Porém, os testes aplicados ainda são totalmente manuais.

Como o ambiente é complexo devido ao sistema possuir muitos módulos, isso dificulta a execução dos testes. Em muitos casos ainda existem projetos com restrição de tempo, não permitindo que os cenários levantados, sejam considerados de forma adequada. Essas situações acabam refletindo na qualidade do produto liberado para o cliente.

² Progress 4GL é uma linguagem de programação de quarta geração, baseada em eventos e proprietária, funcionando em diversos sistemas operacionais, como por exemplo, MS-DOS, Windows NT, UNIX, Novell, CTOS, e também possui um banco de dados relacional integrado a linguagem.

³ Java é uma linguagem de programação multiplataforma, interpretada e orientada a objetos. Diferentemente das linguagens de programação convencionais, que são compiladas para código nativo, o Java é compilado para byte Codes e interpretado em uma máquina virtual, isso permite que os softwares desenvolvidos em Java sejam executados em diversas plataformas sem necessidade de alteração de código.

A implementação de um processo de automação de testes com integração contínua, para auxiliar essa equipe de qualidade pode trazer os seguintes benefícios: maior abrangência de testes das funcionalidades do sistema, maior velocidade na aplicação dos testes ocorrendo assim uma melhor utilização dos recursos computacionais disponíveis, e alocação da equipe de testes para atividades mais produtivas, além de reduzir a interferência manual no ambiente de teste e possibilitar feedback instantâneo sobre cada alteração realizada no sistema.

1.3 QUESTÃO DE PESQUISA

Diante das necessidades apontadas, a questão de pesquisa a ser respondida é:

“Como melhorar o processo de testes, nas fases de teste integrado e sistêmico em um sistema de gestão, especificamente para a linguagem Progress 4GL e Java?”

1.4 OBJETIVOS

Os objetivos deste trabalho dividem-se em objetivo geral e objetivos específicos.

1.4.1 OBJETIVOS GERAIS

O objetivo deste trabalho é propor um processo de automação de testes de software com integração contínua, através da automação de testes funcionais e utilização de ferramenta de integração contínua para um sistema de gestão de planos de saúde desenvolvido na linguagem Progress 4GL. Sendo que o mesmo possui módulos desenvolvidos também em Java.

A criação do processo tem como objetivo definir um plano de testes para ser aplicado na automatização. Esse plano de testes visa estabelecer as atividades que devem ser realizadas, as etapas a serem seguidas, a ordem cronológica de execução e a construção dos cenários, casos de testes e scripts. Dessa forma permitirá definir quais funcionalidades do sistema poderão ser automatizadas, os roteiros de automação a serem seguidos, priorização de módulos do sistema para a

automação e como deverá ser dada manutenção dos ambientes e scripts automatizados, assim como a sua reutilização.

1.4.2 OBJETIVOS ESPECÍFICOS

Para alcançar o objetivo geral, será necessário desenvolver alguns objetivos específicos, que será realizado ao longo do trabalho de conclusão.

1. Verificação de tipos e técnicas de testes de software existentes para a garantia da qualidade.
2. Processo de testes utilizado pelo setor de qualidade da empresa alvo.
3. Avaliação de um processo para automatização.
4. Avaliação de ferramentas adequadas para automação de testes funcionais e integração contínua, levando em consideração o ambiente e as necessidades da empresa
5. Idealização do protótipo de automação, adequando o mesmo para o sistema da empresa
6. Configuração de ferramentas de integração contínua para dar suporte ao processo de automação
7. Integração entre as ferramentas de automação e Integração contínua

1.5 METODOLOGIA

O objetivo do trabalho será alcançado por meio da metodologia de implementação e estudo de caso, dividindo sua realização pelas seguintes etapas:

1. Avaliação dos diferentes tipos e técnicas de testes de software, empregados nas diversas etapas de garantia da qualidade.
2. Reuniões com a equipe de testes da empresa alvo, para mapear os processos de testes utilizados, e como os mesmos são aplicados.
3. Adaptação de um processo de automatização com base no processo de testes utilizado pela empresa alvo, definição de critérios de reutilização e manutenção dos scripts que serão construídos, assim como a manutenção do ambiente de automação.

4. Avaliação e seleção de ferramentas de automação adequadas para o ambiente em questão, com base na infraestrutura disponível, linguagem utilizada pela ferramenta para geração dos scripts, complexidade e custo financeiro para aquisição da ferramenta.
5. Criação de ambiente de protótipo de automação de testes, utilizando-se do processo definido, e aplicando a ferramenta selecionada.
6. Configuração de ferramenta de Integração contínua para auxiliar no processo de automação
7. Integração da ferramenta de automação selecionada com a ferramenta de integração contínua utilizada no ambiente.

1.6 ESTRUTURA DO TEXTO

O capítulo 2 desse trabalho está estruturado de forma a conceituar e contextualizar os temas necessários para o seu desenvolvimento. Serão detalhados os conceitos de testes, processo de testes, assim como as abordagens e tipos de testes que podem ser utilizados na garantia da qualidade.

No capítulo 3 é apresentado o processo atual da empresa, descrevendo o perfil técnico da equipe de testes, assim como as atividades desenvolvidas em cada uma das etapas existentes.

No capítulo 4 são apresentadas as alterações efetuadas no processo de testes atual para contemplar as atividades de automação detalhando cada uma delas.

No capítulo 5 é realizado o detalhamento do processo de escolha da ferramenta de automação, descrevendo os requisitos utilizados e a pontuação de cada ferramenta.

No capítulo 6 é descrito o processo de criação e configuração do ambiente de protótipo, detalhando a instalação das ferramentas utilizadas no processo de automação e Integração Contínua.

No capítulo 7 é detalhado o processo de integração entre as ferramentas de integração contínua e automação.

No capítulo 8 são apresentadas as conclusões e considerações sobre o projeto.

2 TESTE DE SOFTWARE

Pressman (2011), define teste de software como sendo uma função de controle de qualidade com o objetivo principal de descobrir erros. O teste de software é destinado a mostrar que o programa faz o que é proposto a fazer e para encontrar defeitos antes o seu uso. Dessa forma um teste de software bem planejado tem grande probabilidade de encontrar erros (PRESSMAN, 2011, SOMMERVILLE, 2011).

Os testes também podem ser uma estratégia de gerenciamento de riscos, onde são utilizados para verificar a consistência entre os requisitos levantados no planejamento e o produto desenvolvido. Com base nas etapas de desenvolvimento os testes de software podem ser divididos em duas formas básicas: Testes de Verificação e Testes de Validação (MOLINARI, 2010, BARTIE, 2012, PRESSMAN, 2011).

Os testes de validação são baseados no comportamento do software. Nestes testes diversas condições são simuladas, a fim de garantir que o desenvolvido está de acordo com o especificado. A principal característica dos testes de validação é a presença física do software e de seu processamento (BARTIÉ, 2012, PRESSMAN, 2011).

Figura 1 - Exemplo de testes de validação



Fonte: Bartié (2002, p.39)

Os testes não podem demonstrar se o software é livre de defeitos ou se o mesmo se comportará conforme o específico em qualquer situação, ou seja, eles demonstram apenas a presença de erros não a ausência deles. Dessa forma é

necessário que exista um processo de testes bem definido para que o número de erros existentes no produto seja próximo a zero.

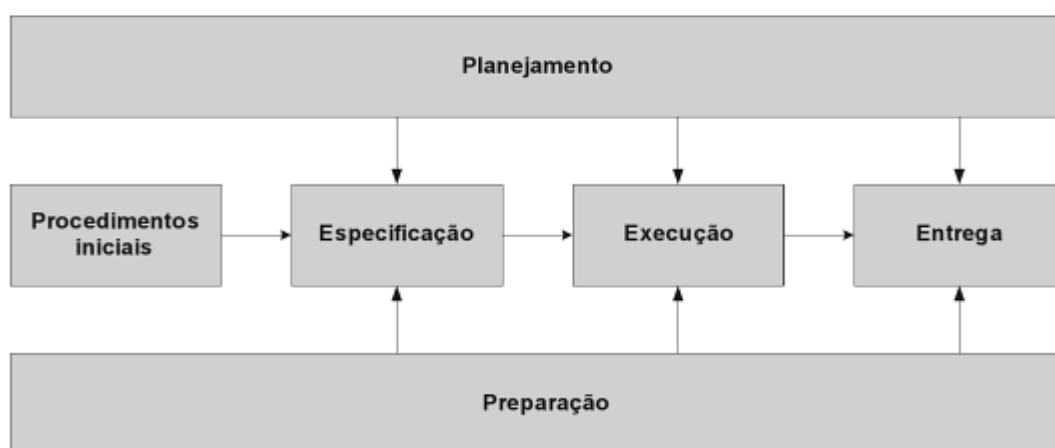
2.1 PROCESSO DE TESTE

A metodologia dos testes, segundo Rios (2012), deve ser o documento básico para organizar a atividade de testar aplicações na empresa. Assim como o desenvolvimento de software é indesejável sem um processo adequado, o mesmo também ocorre como a atividade de teste.

Quanto antes os testes iniciarem, mais barato será corrigir os defeitos encontrados. Para isso ser possível, é preciso que o processo de testes assim como o processo de desenvolvimento tenha um ciclo de vida. O processo de teste deve ser definido com uma metodologia que seja aderente ao processo de desenvolvimento (RIOS, 2012, PRESSMAN, 2011).

Com base nessas premissas Rios (2012) define uma metodologia dividida em diversas etapas ou fases para atender as atividades do processo de teste conforme ilustrado na Figura 2.

Figura 2 - Processo de testes



Fonte: Rios (2012, p.45)

- a) *Procedimentos Iniciais*: é onde ocorre o aprofundamento dos requisitos do negócio que darão origem ao sistema de informação a ser desenvolvido, garantindo que o mesmo seja completo e sem ambiguidades. Nesta etapa também ocorre o levantamento de todas as

principais atividades a serem executadas, alocação de recursos de pessoal e necessidades tecnológicas e de ambiente que darão origem ao plano de testes (RIOS, 2012);

- b) *Planejamento*: consiste em elaborar uma estratégia de teste e o Plano de Teste a serem utilizados, de forma a minimizar os principais riscos do negócio e fornecer os caminhos para as próximas etapas. Essa etapa deve ser executada em conjunto com as atividades de levantamento de requisitos e o planejamento do projeto do desenvolvimento. A etapa de planejamento é responsável pela criação do plano de teste, onde as necessidades e alocações levantadas na etapa de procedimentos iniciais são reavaliadas e oficializadas no plano de teste, que conduzirá as próximas etapas (RIOS, 2012, PRESSMAN, 2011);
- c) *Preparação*: o objetivo é preparar o ambiente de testes (equipamentos, pessoal, ferramentas de automação, hardware e software) para que os testes sejam executados corretamente. Na etapa de preparação pode-se avaliar as necessidades de treinamento da equipe com base nos requisitos do que será testado, assim como a disponibilização de ferramentas que auxiliem a equipe no processo de testes. A etapa de preparação pode ser executada em paralelo com as demais (RIOS, 2012);
- d) *Especificação*: possui dois objetivos básicos que são elaborar e revisar casos de teste e elaborar e revisar roteiros de testes. Os casos de teste e os roteiros de testes devem ser elaborados dinamicamente ao decorrer do projeto, ou seja, os casos de testes são elaborados a medida que a equipe de desenvolvimento libera alguns módulos ou funcionalidades do sistema para que os mesmos sejam validados pela equipe de testes (RIOS, 2012);
- e) *Execução*: a etapa de execução é responsável por executar os testes planejados nas etapas anteriores e registrar os resultados obtidos. Os testes deverão ser executados de acordo com os casos de teste e os roteiros de teste definidos anteriormente. Devem ser usados scripts de teste, caso seja empregada alguma ferramenta de automação de testes. Os testes deverão ser executados integralmente, por regressão

ou parcialmente, sempre que ocorrer mudança na versão dos programas, conforme previsto no plano de teste (RIOS, 2012);

- f) *Entrega*: é onde ocorre a conclusão do projeto de testes com a entrega do sistema para o ambiente de produção. Nessa etapa é realizado o arquivamento da documentação gerada pelo processo de teste, onde são relatadas todas as ocorrências do projeto que forem consideradas importante à melhoria do processo (RIOS, 2012);

2.2 ABORDAGENS DE TESTES

Segundo Pressman (2011), qualquer produto de software pode ser testado de duas formas diferentes:

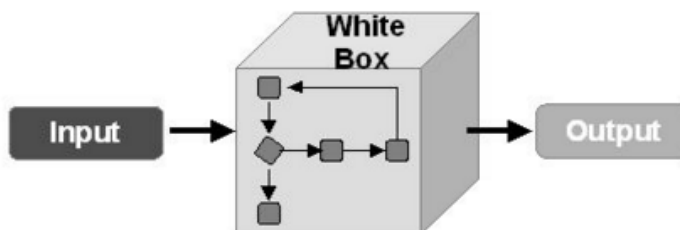
1. Conhecendo a função específica para o qual o software foi projetado para realizar, podem ser feitos testes que demonstram que cada uma das funções é totalmente operacional, embora ao mesmo tempo procurem erros em cada função
2. Conhecendo o funcionamento interno do software, podem ser realizados testes para garantir que todos os componentes funcionam corretamente.

Dessa forma as abordagens de teste podem ser divididas em: Abordagem de teste de caixa branca e abordagem de teste de caixa preta.

2.2.1 TESTE CAIXA BRANCA

O teste de caixa branca focaliza a estrutura interna do software, ou seja, utiliza a estrutura de controle do programa para derivar os casos de testes, essa estratégia tem como objetivo exercitar características específicas do projeto, como desvios, laços (loops), interfaces entre módulos, limites de armazenamento, etc. Por esses motivos os testes de caixa branca são também denominados testes estruturais (PRESSMAN, 2011).

Figura 3 - Representação teste caixa branca



Fonte: Software Testing Genius⁴

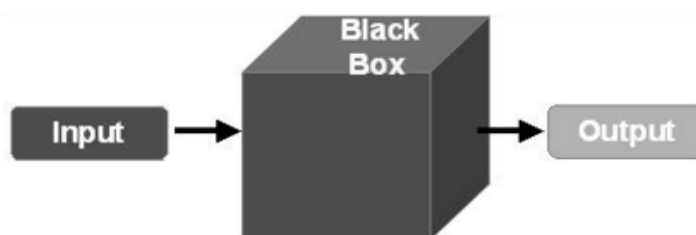
Uma desvantagem da técnica de caixa branca é que a mesma não analisa se a especificação está correta, concentrando-se apenas no código fonte e não verificando a lógica da especificação. Outra desvantagem que pode ser considerada é o número de casos de testes que são gerados a partir da análise da estrutura interna do programa, o que torna o processo de testes exaustivo (PRESSMAN, 2011).

2.2.2 TESTE CAIXA PRETA

O método de caixa preta concentra-se nos requisitos funcionais do software, ou seja, possibilita que o engenheiro de software derive conjuntos de dados de entrada, que exercitem completamente todos os requisitos funcionais para um programa.

Diferentemente da técnica de caixa branca os testes de caixa preta não levam em consideração a estrutura interna do programa, concentrando-se apenas nos domínios de informação.

Figura 4 - Ilustração teste caixa preta



Fonte: Software Testing Genius⁵

⁴ Disponível em: <<http://m.softwaretestinggenius.com/?page=details&url=white-box-unit-testing-a-bottom-up-approach-of-software-testing>> acessado em 10 nov.2015

Os testes de caixa preta não são uma alternativa aos testes de caixa branca, mas sim uma abordagem complementar, com possibilidade de descobrir uma classe de erro diferente daquela obtida com métodos de caixa branca (PRESSMAN, 2011).

Segundo Pressman (2011), os testes de caixa preta tentam identificar erros nas seguintes categorias:

1. Funções incorretas ou faltando
2. Erros de interface
3. Erros em estruturas de dados ou acesso a bases de dados externas
4. Erros de comportamento ou desempenho
5. Erros de inicialização e termino.

Outra diferença entre as abordagens caixa branca e caixa preta é o momento em que elas são aplicadas, enquanto os testes de caixa branca são aplicados antecipadamente no processo de testes, possibilitando que os mesmos sejam executados em meio a codificação das funcionalidades do software, os testes de caixa preta tendem a serem executados em estágios posteriores dos testes, pois dependem das funcionalidades para serem executados (PRESSMAN, 2011, BARTIÉ, 2002).

Os métodos de caixa preta podem ser divididos em três principais: Particionamento por Equivalência, Análise de Valor Limite e Grafo de Causa e Efeito (PRESSMAN, 2011, BARTIÉ, 2002, MOLINARI, 2008).

2.2.2.1 PARTICIONAMENTO POR EQUIVALÊNCIA

O método de particionamento por equivalência divide o domínio de entrada de um programa em classes de dados das quais os casos de testes podem ser derivados. Cada classe representa um possível erro a ser identificado, permitindo que os casos de testes redundantes de cada classe identificada sejam eliminados sem que a cobertura dos cenários existentes seja prejudicada (BARTIÉ, 2002, PRESSMAN, 2011).

⁵ Disponível em: <<http://m.softwaretestinggenius.com/?page=details&url=white-box-unit-testing-a-bottom-up-approach-of-software-testing>> acessado em 10 nov.2015

Figura 5 - Exemplo particionamento por equivalência

Entrada	Valores Permitidos	Classes	Casos de Teste
Idade	Idade entre 18 e 120	18 a 120	Idade = 18 Idade = 19 Idade = 119 Idade = 120
		< 18	Idade = 17
		> 120	Idade = 121

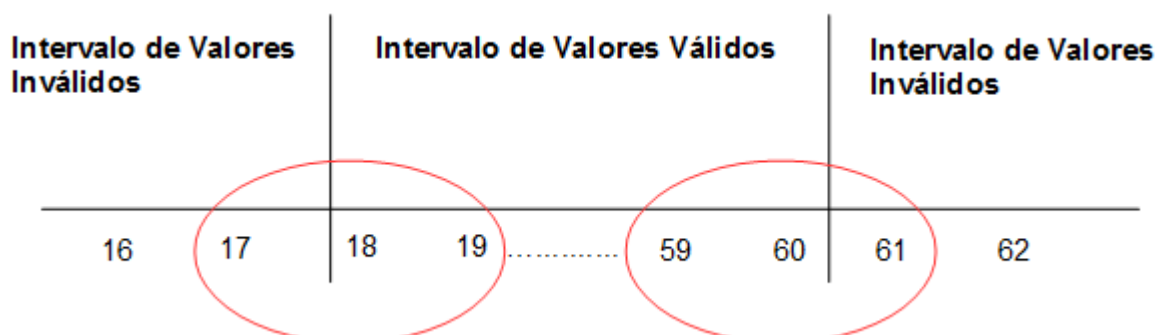
Fonte: Test Ware Quality⁶

O particionamento de equivalência tem por objetivo minimizar o número de casos de teste, selecionando apenas um caso de teste de cada classe, pois em princípio todos os elementos de uma classe devem se comportar de maneira equivalente (BARTIÉ, 2002, PRESSMAN, 2011).

2.2.2.2 ANÁLISE VALOR LIMITE

O método de Análise Valor Limite é complementar à partição por equivalência, enquanto no particionamento por equivalência qualquer valor dentro de uma classe identificada seria candidato ao caso de teste selecionando, na técnica valor limite são explorados os valores limites de cada classe (BARTIÉ, 2002, MOLINARI, 2010).

Figura 6 - Exemplo análise valor limite



Fonte: Matera Systems⁷

⁶ Disponível em: <<http://testwarequality.blogspot.com.br/p/tenicas-de-teste.html>> acessado em 20 nov.2015

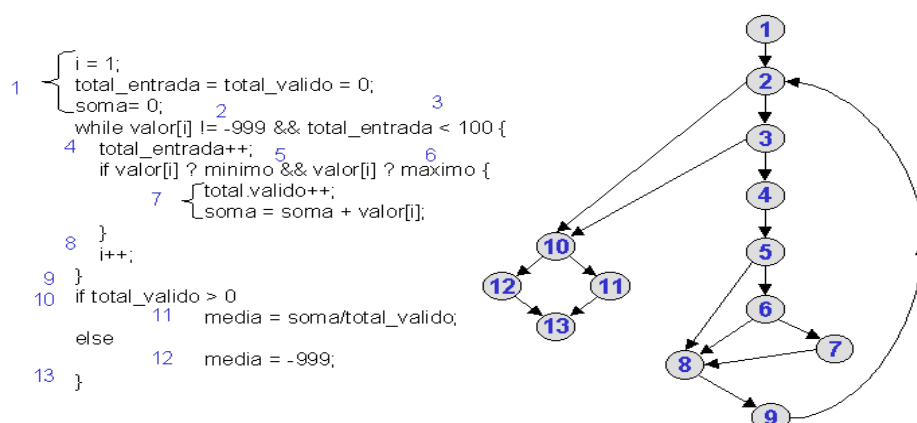
Segundo Bartié (2002), o software está mais susceptível a erros nas fronteiras do domínio de dados do que propriamente nas regiões centrais, portanto, utilizar testes que explorem valores de fronteira do domínio aumentam a probabilidade de identificação de erros. A técnica de valor limite considera tanto os valores de saída quanto os de entrada, diferentemente da técnica de particionamento de equivalência que tem foco apenas nas condições de entrada.

2.2.2.3 GRAFO DE CAUSA E EFEITO

O grafo de causa efeito é uma técnica que oferece uma representação concisa das condições lógicas e das ações correspondentes. Esse critério de teste verifica o efeito combinado de dados de entrada. As causas (condições de entrada) e os efeitos (ações) são identificados e combinados em um grafo a partir do qual é montada uma tabela de decisão, e a partir desta, são derivados os casos de teste e as saídas (PRESSMAN, 2011, MOLINARI, 2010).

Para a criação do grafo de causa e efeito são analisados os objetos que compõem o software, assim como as relações existentes entre eles. Dessa forma os objetos são apresentados por nós, e as relações são representados por ligações entre esses nós (PRESSMAN, 2011).

Figura 7 - Exemplo grafo causa e efeito



Fonte: DIMAP UFRN⁸

⁷ Disponível em: <http://www.matera.com/br/2015/11/27/tecnicas-de-testes-para-desenvolvedores-partiutestar/>
Acesso em 22 nov. 2015

⁸ Disponível em: < <https://www.dimap.ufrn.br/~jair/ES/c8.html> > Acesso em 02 dez. 2015

2.3 TIPOS DE TESTES

Existem muitos tipos de testes que podem ser utilizados para assegurar a qualidade de um software. A execução desses tipos de testes se estende desde execuções diárias quando uma nova funcionalidade é construída ou modificada, até execuções que testam todas as funcionalidades do sistema já construído.

Uma estratégia empregada pela maioria das equipes de software está entre esses dois extremos, ou seja, assume uma visão incremental dos testes, começando com os testes de unidade individuais do programa, e passando para os testes destinados a facilitar a integração de unidades e culminando com testes que usam o sistema concluído (PRESSMAN, 2011, PFLEEGER, 2004).

Dessa forma pode-se considerar três estratégias de testes: Teste de Unidade, Teste de Integração e Teste de sistema (PRESSMAN, 2011, MOLINARI, 2010, BARTIÉ, 2002).

2.3.1 TESTES DE UNIDADE

O teste de unidade tem como foco verificação da menor unidade de projeto do software o componente ou o módulo de software. O teste de unidade normalmente é considerado um auxiliar para a etapa de codificação. O projeto dos testes de unidade pode ocorrer antes de começar a codificação ou depois que o código-fonte estiver gerado. Os testes de unidade podem utilizar tanto a abordagem de teste de caixa branca quanto de caixa preta, considerando o que se deseja validar (PRESSMAN, 2011).

Pressman (2011), define alguns aspectos que os casos de teste de unidade devem considerar: Interface, Estruturas de dados Locais, Condições limite, Caminhos independentes e Caminhos de manipulação de erro.

A Interface do módulo é testada para assegurar que as informações fluam corretamente para dentro e para fora do programa que está sendo testado; a estrutura de dados local é examinada para garantir que os dados armazenados temporariamente mantêm a sua integridade durante todos os passos do algoritmo. Todos os caminhos independentes da estrutura de controle são usados para assegurar que todas as instruções em um módulo tenham sido executadas pelo menos uma vez; as condições limite são testadas para garantir que o módulo opere

adequadamente nas fronteiras estabelecidas para limitar ou restringir o processamento. Finalmente são testados todos os caminhos de manipulação de erro (PRESSMAN, 2011).

Figura 8 - Aspectos dos testes de unidade



Fonte: Pressman (2011, p.408)

Sempre que possível os testes de unidade devem ser automatizados, dessa forma um conjunto inteiro de testes frequentemente pode ser executado em alguns segundos, possibilitando reexecutar todos os testes a cada alteração realizada no software (SOMMERVILLE, 2011).

2.3.2 TESTE DE INTEGRAÇÃO

Pressman (2015) define o teste de integração como sendo uma técnica sistemática para construir a arquitetura de software ao mesmo tempo que conduz testes para descobrir erros associados com as interfaces.

Quando se tem certeza que os componentes individuais desenvolvidos estão funcionando corretamente, os mesmos são combinados em um sistema em funcionamento. Essa combinação deve ser planejada de modo que quando ocorra uma falha, seja possível identificar o componente que a gerou (PRESSMAN, 2011, PFLEEGER, 2004).

A integração de componentes pode se dar de duas maneiras: Não Incremental e Incremental. A abordagem não incremental também é conhecida como big-bang, nessa abordagem todos os componentes são combinados com

antecedência, o programa inteiro é testado como um todo (PRESSMAN, 2011, PFLEEGER, 2004).

A abordagem incremental é o oposto da big-bang, na abordagem incremental o programa é construído e testado em pequenos incrementos, dessa forma erros são mais fáceis de se isolar e identificar; as interfaces têm maior probabilidade de serem testadas completamente (PRESSMAN, 2011, PFLEEGER, 2004).

2.3.3 TESTE DE SISTEMA

O teste de sistema pode ser definido como sendo uma série de testes cuja finalidade primária é exercitar totalmente o sistema. Embora cada um dos tipos de testes de sistema tenha finalidades diferentes, todos buscam validar se todos os componentes do software foram integrados corretamente, além de assegurar que o sistema faz o que o cliente quer que ele faça (PRESSMAN, 2011, PFLEEGER, 2004).

O teste sistêmico pode ser subdividido em vários tipos de testes, sendo que cada um objetiva testar um aspecto diferente do sistema. Nessa seção serão tratados apenas testes funcionais e testes de regressão.

2.3.3.1 TESTE FUNCIONAL

O teste funcional tem por objetivo simular todos os cenários de negócio e garantir que todos os requisitos funcionais sejam implementados. Os testes funcionais são complexos, devido ao fato de exigirem profundo conhecimento da regra de negócio de uma aplicação para contemplar todas as variações de cenários existentes (PFLEEGER, 2004, BARTIÉ, 2002).

Os Testes funcionais devem ser direcionados pelos requisitos funcionais. Os documentos de requisitos descrevem o comportamento que o software deve assumir nos diversos cenários existentes em cada requisito de negócio (PFLEEGER, 2004, BARTIÉ, 2002).

Segundo Bartié (2002), é possível decompor cada requisito em três cenários distintos: Cenário Primário, Cenários Alternativos e Cenário de Exceção. O cenário primário trata-se do cenário ótimo, no qual não existem problemas ou exceções à

regra. Os cenários alternativos são as variações dentro do cenário primário, isto é, os caminhos alternativos ou situações equivalentes que conduzem ao mesmo objetivo. O cenário de exceção trata de possíveis problemas e inconsistências que impedem a finalização de determinado requisito.

2.3.3.2 TESTE DE REGRESSÃO

O teste de regressão é a reexecução do mesmo subconjunto de testes que foram executados, para assegurar que as alterações não tenham propagado efeitos colaterais indesejados. Cada vez que um novo módulo é acrescentado, o software muda, novos caminhos de fluxos de dados são estabelecidos, podem ocorrer novas entradas e saídas e novas lógicas de controle podem ser chamadas. Essas alterações podem causar problemas em funções que antes funcionavam corretamente. Sempre que um software é corrigido ou modificado resulta em alteração de algum aspecto de configuração do software (PRESSMAN, 2011, PFLEEGER, 2004).

O teste de regressão ajuda a garantir que as alterações (acréscimo de novos módulos, correções ou por qualquer alteração) não introduzam comportamentos indesejados ou erros adicionais. A medida que o teste de integração progride o número de casos de testes pode crescer muito (PRESSMAN, 2011, PFLEEGER, 2004).

Portanto, os testes de regressão devem ser projetados de forma a minimizar esse número de casos de teste incluindo somente testes que contemplem mais de uma classe de erro em cada uma das funções principais do programa. É impraticável reexecutar todos os testes para todas as funções do programa quando ocorrem mudanças (PRESSMAN, 2011).

O teste de regressão poder ser executado manualmente, reexecutando um subconjunto de casos de teste ou aplicar automação de testes utilizando ferramentas de captura/reexecução. Geralmente o número de casos de testes é grande, por esse motivo é indicado que os testes sejam automatizados, dessa forma é possível cobrir um número maior de casos de testes em um curto intervalo de tempo (SOMMERVILLE, 2011, PRESSMAN, 2011).

2.4 TESTES AUTOMATIZADOS

O teste é responsável por garantir os vários aspectos da qualidade do software que vai desde o levantamento de requisitos e arquitetura até a codificação, integração de componentes e validação de funcionalidades. Por isso, é indispensável que exista uma maneira fácil e ágil de executar todos os testes em qualquer momento, e isso só é viável com o auxílio de testes automatizados (MOLINARI, 2008, BARTIÉ, 2002, PRESSMAN, 2011).

A automação de testes dá segurança para a equipe realizar alterações no software, sejam elas de refatoração, manutenção ou desenvolvimento de novas funcionalidades, pois permite a elaboração de casos de testes mais complexos e que podem ser executados repetidas vezes de forma rápida e com o mínimo de esforço, garantindo dessa forma uma melhor qualidade (MOLINARI, 2008).

Assim como os testes manuais os testes automatizados têm como objetivo melhorar a qualidade do produto desenvolvido através da validação e verificação, ainda que um grande grau da automação não substitua um processo bem organizado e racional de qualidade, a automação de testes permite um melhor uso dos recursos humanos disponíveis (PRESSMAN, 2011, MOLINARI, 2008).

Nem todas as atividades de testes devem ou podem ser automatizadas. Existem tarefas que só podem ser realizadas de forma manual, como criação de casos de testes e depuração de código. Além disso as atividades que devem ser automatizadas são as que darão maior retorno ou que são mais impactantes e repetitivas no processo de testes (COSTA, 2004).

A automação é uma atividade que requer dedicação total de tempo e esforço, assim como investimento em ferramentas e capacitação da equipe. Segundo Costa (2004), a automação pode ter um custo de três a dez vezes em relação aos testes manuais para criar, verificar e documentar. Devido ao alto custo envolvido um projeto de automação deve ser planejado de modo a reduzir os custos e riscos envolvidos (MOLINARI, 2008, BARTIÉ, 2002, COSTA, 2004).

Apesar de demandar investimento, dedicação total e tempo da equipe, a automação traz grandes vantagens a médio e longo prazo. Permite que testes antes repetitivos e onerosos sejam executados repetidas vezes e de forma rápida e ágil. Recursos humanos antes utilizados para executadas essas tarefas dispendiosas podem ser alocadas de forma mais produtiva na equipe, como, por exemplo, criar

cenários e casos e de testes mais elaborados ou validar de forma concisa a regras de negócio que o software deve atender (MOLINARI, 2010, COSTA, 2004, PRESSMAN, 2011, MOLINARI, 2008).

Com a reexecução dos testes ocorre uma inversão onde se passa mais tempo testando o sistema do que corrigindo o mesmo, isso reflete na qualidade do produto final disponibilizado para o cliente (MOLINARI, 2008, BARTIÉ, 2002).

A automação também permite que erros sejam identificados de forma prematura no processo de testes, isso evita que esses erros migrem para as próximas fases do processo de testes, reduzindo assim o custo e o impacto que poderiam ser gerados no sistema (MOLINARI, 2008, BARTIÉ, 2002).

Assim como o processo de teste manual se utiliza de recursos humanos para executar os testes, o processo de automação se utiliza de ferramentas computacionais para executar essas atividades. O processo de automação pode se estender desde a fase de documentação até a fase de testes considerados de nível superior, existindo ferramentas específicas que cobrem cada fase do processo da garantia da qualidade (PRESSMAN, 2011, MOLINARI, 2010, RIOS, 2012).

2.5 INTEGRAÇÃO CONTÍNUA

Integração Contínua é definida como uma pratica de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada uma dessas integrações é validada por um build automatizado que geralmente compila os códigos-fontes verificando erros de integração entre componentes do software, sendo que esses builds também podem ser a execução de testes (FOWLER, 2006).

A grande vantagem que o processo de integração contínua traz é o feedback instantâneo, onde cada integração realizada é verificada automaticamente pelos builds, sendo que os mesmos podem ser execuções de testes, possibilitando assim que erros possam detectados rapidamente e comunicados a equipe evitando maiores impactos no software, isso traz maior segurança para se realizar alterações e manutenções no sistema (FOWLER, 2006).

3 LEVANTAMENTO DO PROCESSO DE TESTE DA EMPRESA ALVO

Nessa seção será apresentado o processo de desenvolvimento da empresa alvo, a partir das informações levantadas com a área de qualidade. Para mapear o processo existente foram realizadas reuniões com a equipe responsável pelas atividades de teste. O levantamento foi guiado por um questionário de caráter qualitativo conforme *Anexo A*. As respostas do questionário foram compiladas onde os participantes foram identificados com letras de A até F, e elencadas as respostas de cada participante para cada uma das perguntas, conforme *Anexo B*. Com bases nas informações levantadas foi possível identificar o perfil da equipe, características do processo de teste e ferramentas utilizadas.

3.1 FERRAMENTAS DE APOIO AO PROCESSO

No processo atual a empresa faz uso de diversas ferramentas que auxiliam no gerenciamento dos projetos, controle de atividades das equipes, documentação dos artefatos gerados durante o projeto, versionamento de código fonte, integração contínua entre outros.

Para o gerenciamento de projetos, controle de atividades, workflows de aprovação e documentação de artefatos a empresa utiliza a ferramenta Jira⁹. O controle de versionamento do código fonte é feito através do Team Foundation Server¹⁰. O processo de integração contínua faz uso da ferramenta Jenkins.¹¹

Um ponto importante levantado sobre o uso das ferramentas é o fato de que o planejamento de teste assim como as demais etapas não faz uso de ferramentas que apoiem na execução dessas atividades. Até o momento os cenários e casos de testes são controlados por meio de planilhas conforme *Anexo E*.

⁹ Ferramenta comercial desenvolvida pela empresa australiana Atlassian. O Jira permite monitorar e acompanhar as tarefas e projetos de forma centralizada.

¹⁰ O Team Foundation Server ou TFS é um produto da Microsoft que permite o gerenciamento dos códigos fonte, compilações automatizadas entre outros.

¹¹ O Jenkins é um servidor de automação fonte aberto desenvolvido em Java. O Jenkins permite construir e testar software continuamente, sendo um facilitador no processo de desenvolvimento e integração contínua

3.2 PERFIL DA EQUIPE

A equipe de testes da empresa é composta por 7 integrantes sendo um deles o coordenador da área. Baseado nas reuniões e respostas do questionário, pode-se evidenciar que a equipe não é focada na área técnica, mas sim voltada para área de negócio. Dos 6 respondentes apenas 2 tem conhecimento em alguma linguagem de programação. Outro ponto importante é o fato de o nível de conhecimento sobre o sistema ser básico e intermediário.

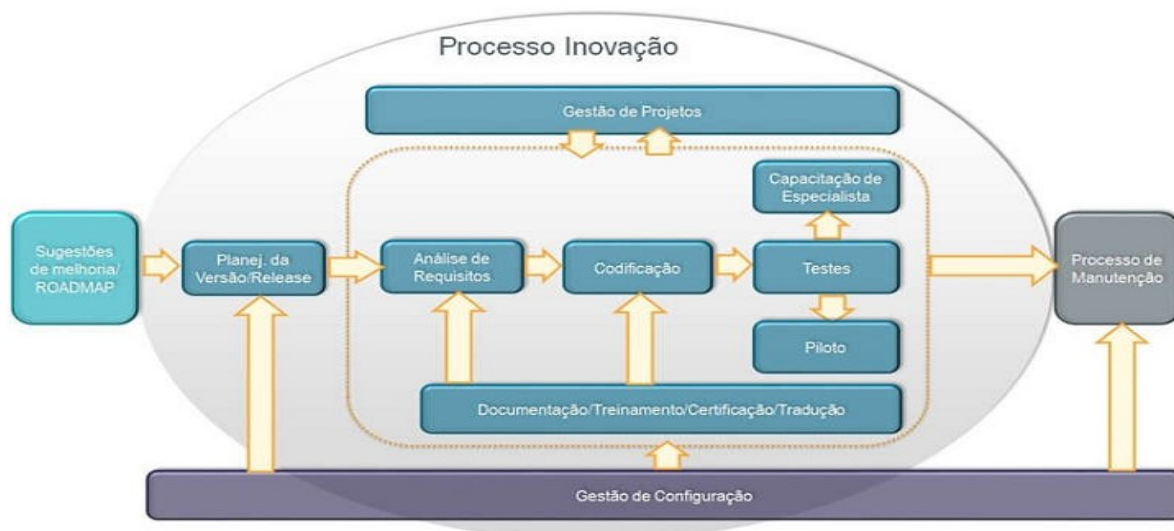
Os responsáveis das etapas do processo de teste podem variar de acordo com os conhecimentos das regras de negócio, particularidades do projeto e disponibilidade dos recursos. O Nível de conhecimento dos integrantes em relação a automação de testes é reduzido, pois até o momento a empresa não se utiliza desse recurso para garantir a qualidade do produto.

3.3 PROCESSO DE TESTE

Identificou-se que a empresa possui um processo de desenvolvimento bem definido. O processo atual segue o modelo em cascata, onde os testes são realizados após o fim da etapa de codificação. Para cada etapa existem workflows de aprovação, onde são alocados recursos para validarem os artefatos gerados, caso sejam evidenciadas inconsistências a etapa é rejeitada para que sejam realizadas as devidas adequações.

O processo existente contempla todo o ciclo de vida do software, que se estende desde o planejamento de novas funcionalidades até a manutenção do produto. Como o foco desse trabalho é automação de testes, apenas a etapa intitulada como Testes será abordada nessa seção. O processo utilizado pela empresa é representado na Figura 9.

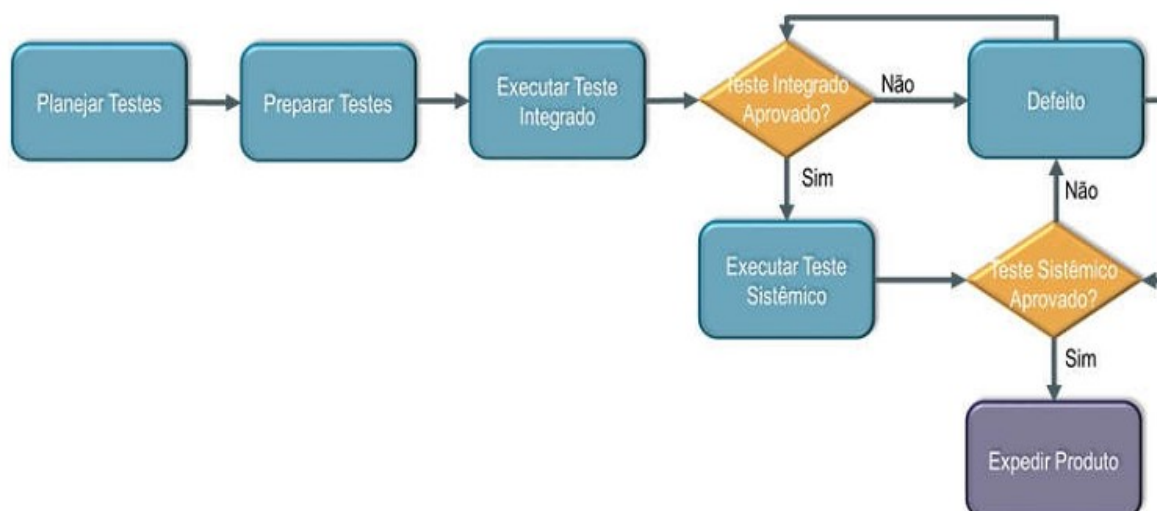
Figura 9 - Processo de desenvolvimento da empresa alvo



Fonte: Totvs (2016)

A etapa de Testes é a responsável pela execução dos testes integrados e sistêmicos, é nessa etapa que as atividades da equipe de testes estão focadas. Apesar de o ambiente de validação ser complexo o processo de garantia de qualidade ainda é manual. As etapas do processo de testes são: Planejar Testes, Preparar Testes, Executar Teste Integrado e Executar Teste Sistemico. O processo de testes da empresa é representado na Figura 10.

Figura 10 - Processo de testes da empresa alvo



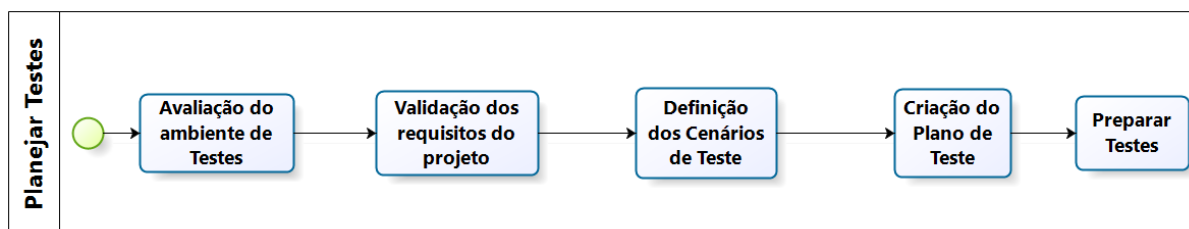
Fonte: Totvs (2016)

3.3.1 PLANEJAR TESTES

Conforme descrito no processo da empresa, essa etapa tem como objetivo dimensionar o esforço, e os recursos necessários para realizar as atividades de testes, assim como definir as diretrizes do que será testado. Essa etapa pode ser executada em conjunto com a atividade de análise de requisitos, mas sempre deve ser executada antes da etapa de preparar testes.

Na etapa de planejamento é realizada uma avaliação em conjunto com o responsável pelo projeto a ser testado com o objetivo de levar as necessidades referentes ao ambiente de testes. São avaliados os requisitos do projeto dentro do contexto do produto e também é realizada a criação do plano de teste seguindo um modelo predefinido conforme *Anexo D*. É na etapa de planejamento que os cenários de testes são levantados, por esse motivo geralmente essa etapa é executada por um analista de teste mais experiente. As atividades da etapa de Preparar Testes estão representadas na figura 11.

Figura 11 - Atividades Etapa Planejar Testes



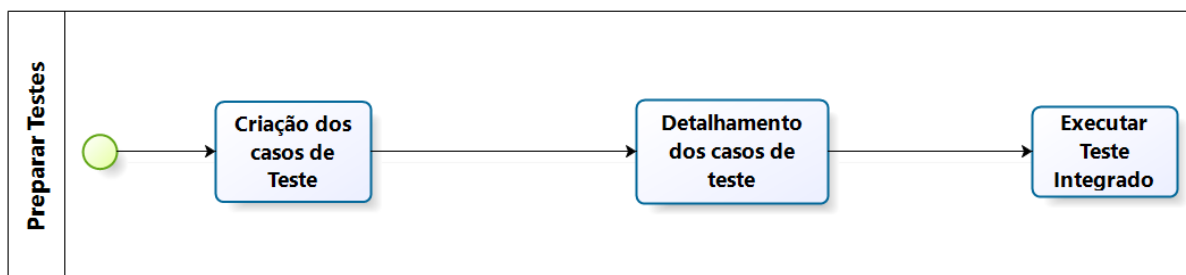
Fonte: O autor, 2016

3.3.2 PREPARAR TESTES

Na etapa de preparação de testes são identificados e detalhados os casos de teste necessários para validação do sistema com base nos cenários definidos no planejamento. Os casos de teste identificados são registrados em uma planilha de controle. Após a identificação esses casos de testes são detalhados com as informações necessárias para a sua execução conforme exemplificado no *Anexo F*, seguindo o modelo definido pela empresa e passando pelas etapas de aprovação previamente definidas.

Conforme informações levantadas a identificação e detalhamento dos casos de testes se dá através da decomposição dos requisitos da engenharia, já o refinamento é realizado com base na experiência e conhecimento do analista sobre o sistema. A figura 12 ilustra as atividades da etapa de Preparar Testes

Figura 12 - Atividades Preparar Testes



Fonte: O autor, 2016

3.3.3 EXECUTAR TESTE INTEGRADO

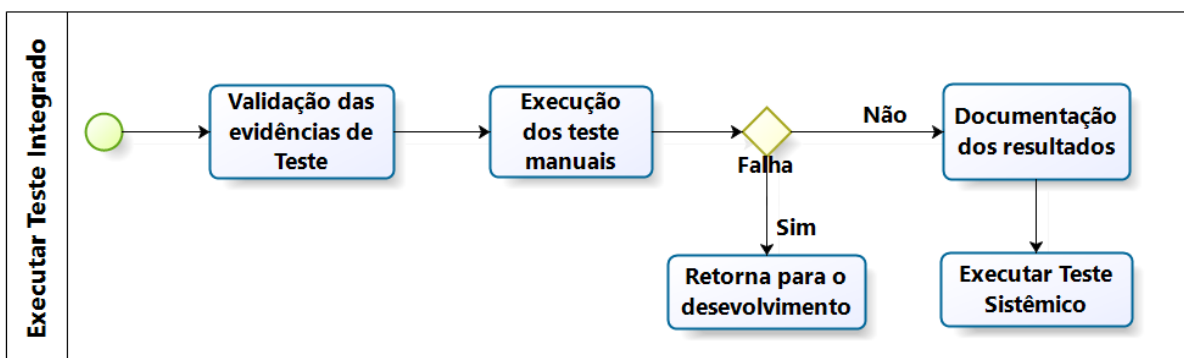
O teste integrado é a fase de teste em que os módulos são combinados e testados em grupo. Essa etapa deve anteceder o teste sistêmico.

Baseado nas informações levantadas o teste integrado utiliza a abordagem não-incremental e todos os testes executados são caixa preta com foco na validação das funcionalidades do sistema, ou seja, não existe verificação de código por parte da equipe de testes.

Os testes executados nessa etapa são casos de testes identificados e detalhados na etapa de preparação, sendo que novos casos de testes podem considerados caso o analista responsável julgue necessário. Além da execução dos testes é nessa etapa que as evidências dos testes funcionais disponibilizadas pelo desenvolvedor são validadas.

Os testes executados nessa etapa são totalmente manuais, assim como a geração das evidências de teste. Outro ponto importante é fato do ambiente de teste utilizado nessa etapa ser compartilhado com as equipes de desenvolvimento e manutenção, ou seja, não é um ambiente isolado. A figura 13 representa as atividades desta etapa.

Figura 13 - Atividades Executar Teste Integrado



Fonte: O autor, 2016

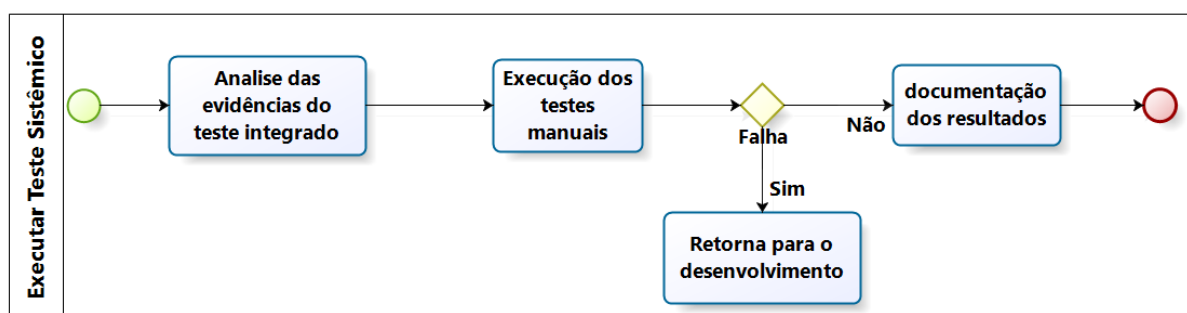
3.3.4 EXECUTAR TESTE SISTÊMICO

O teste sistêmico é a fase do processo de testes em que o sistema já integrado é verificado quanto a seus requisitos num ambiente correspondente ou mais próximo possível do ambiente de produção. Diferentemente da etapa de teste integrado o teste sistêmico é executado em um ambiente isolado.

Os testes executados nessa etapa são os casos de testes contemplados em um roteiro de testes criado especificamente para essa fase. Esse roteiro contém os cenários de teste mais críticos levantados em conjunto com os clientes, porém, novos casos de testes podem ser adicionados conforme a evolução das funcionalidades.

Assim como as demais fases o teste sistêmico é executado de forma manual e a documentação gerada nessa fase é arquivada na ferramenta de gerenciamento de projetos utilizada pela empresa. As atividades executadas na etapa de planejar testes são ilustradas na figura 14.

Figura 14 - Atividades Executar Teste Sistemico



Fonte: O autor, 2016

4 PROCESSO PARA AUTOMAÇÃO DE TESTES FUNCIONAIS

O objetivo de implementação desse trabalho é propor um processo de teste com integração contínua que atenda a automação funcional em ambiente *desktop* (Windows) e web conforme as necessidades da empresa alvo. A partir do processo de teste existente descrito no capítulo 3, será necessária a alteração nas atividades de cada etapa afim contemplar as tarefas de automação.

Além de propor mudanças no processo é necessário o levantamento dos requisitos para a avaliação e escolha das ferramentas de apoio ao processo de gerenciamento de testes e automação de testes funcionais.

Conforme as informações levantadas na seção 3, identificou-se que a empresa possui um processo de desenvolvimento bem estruturado. Com base no processo já existente foram realizadas adequações nas etapas de teste acrescentando-se novas atividades que atendam as demandas dos testes automatizados.

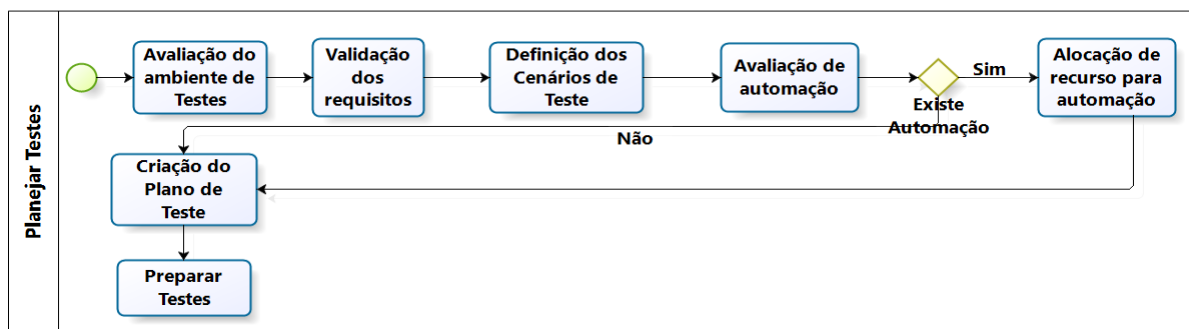
As atividades adicionadas ao processo visam identificar os cenários e casos de testes a serem automatizados, realizar a criação e versionamento dos scripts de automação, validar e documentar os resultados gerados a partir da execução das rotinas automatizadas.

Na etapa de planejamento foram adicionadas as tarefas de avaliação de automação e alocação de recurso para automação. Na avaliação de automação é verificada a necessidade de automatizar os cenários com base na criticidade do mesmo sobre o processo de negócio do cliente, assim como o impacto negativo que pode ser gerado sobre o sistema.

Caso exista a necessidade de automação a tarefa de alocação de recurso para a automação deve ser executada. Para essa tarefa são elencando os recursos com maior conhecimento técnico para desenvolver a atividade de teste integrado, uma vez que serão esses recursos que irão realizar todas as tarefas pertinentes a automação dos casos de testes, validação e versionamento dos scripts.

A figura 15 ilustra todas as atividades que fazem parte da etapa de planejar testes considerando as novas tarefas de automação criadas e inseridas no processo já existente.

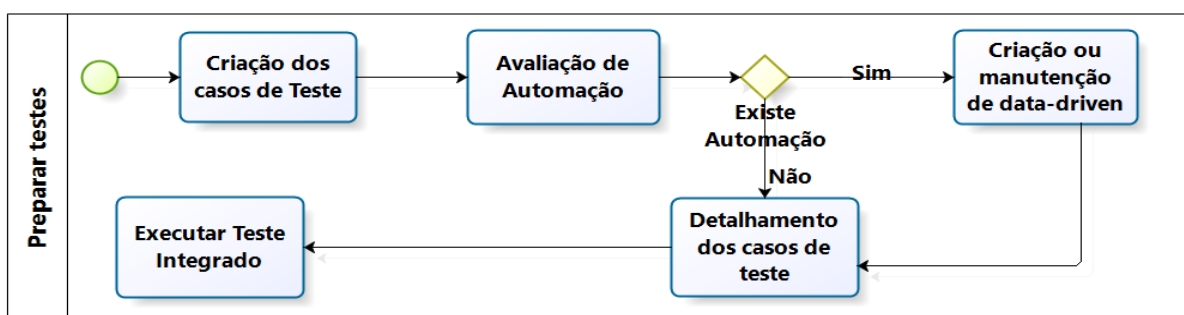
Figura 15 - Planejar Testes com atividades de automação



Fonte: O autor, 2016

Para a etapa de preparação de testes acrescentou-se as tarefas de avaliação de automação e criação ou manutenção de data-driven¹². Assim como na etapa de planejamento a tarefa de avaliação de automação avalia se existem casos de testes a serem automatizados ou atualizados, caso seja evidenciada essa necessidade a tarefa de criação ou manutenção dos data-driven é executada com a finalidade de gerar os dados que serão utilizados nas variações dos testes automatizados. A figura 16 representa as atividades desse processo considerando o processo automação.

Figura 16 - Preparar Testes com atividades de automação



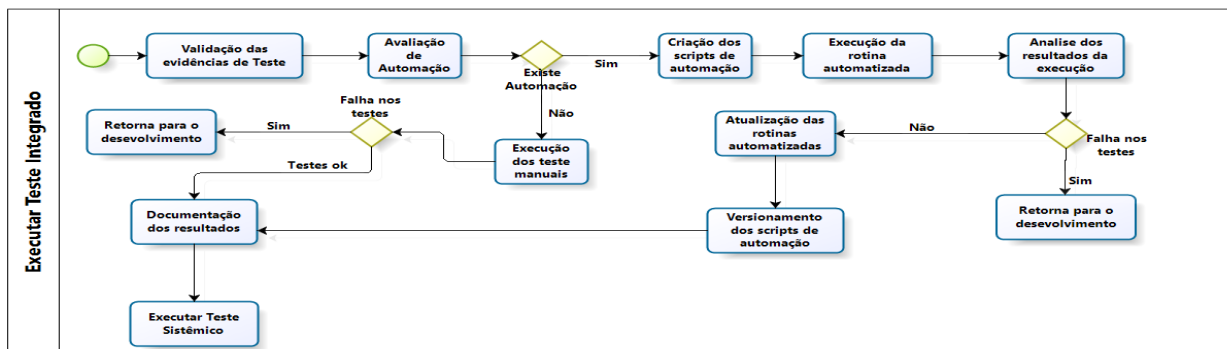
Fonte: O autor, 2016

A etapa de teste integrado foi que sofreu maiores mudanças, pois é nessa etapa que os scripts de automação serão gerados e validados. As tarefas adicionadas nessa etapa foram: Avaliação de automação, Criação dos scripts de automação, execução da rotina automatizada, Análise dos resultados da execução, Atualização das rotinas automatizadas e versionamento dos scripts automação. As

¹² Data-Driven, ou orientação a dados é uma técnica onde a entrada de dados para os scripts de automação é feita de forma externa, ou seja, os dados são carregados de arquivos ou banco de dados permitindo um maior reaproveitamento desses scripts e uma maior variação de casos de testes.

tarefas dessa etapa considerando também as atividades de automação podem ser visualizadas na figura 17.

Figura 17 - Executar Teste Integrado com atividades de automação

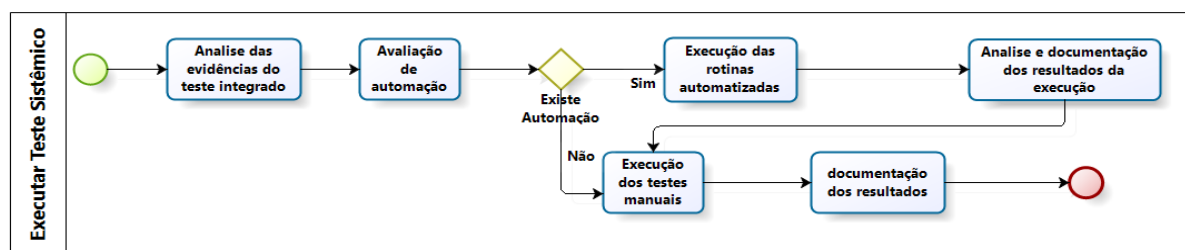


Fonte: O autor, 2016

Se existirem casos de testes para automação então é executada a tarefa de criação dos scripts, onde o teste funcional será gravado gerando o script de automação. Com a geração do script o teste é executado novamente e verificado se o mesmo está funcionando adequadamente, não se evidenciando inconsistências no script gerado deverá se atualizar o registro das rotinas automatizadas informando o novo cenário criado. Ao fim da execução de todas essas tarefas o script é versionado para que possa ser reutilizado futuramente.

Na etapa de testes sistêmicos foram adicionadas as seguintes tarefas: Avaliação de automação para verificar se existem rotinas automatizadas; Execução das rotinas automatizadas onde os scripts gerados no teste integrado são reexecutados e análise e documentação dos resultados da execução, onde é verificado o sucesso ou falha do teste. A figura 18 representas as atividades do teste sistêmico.

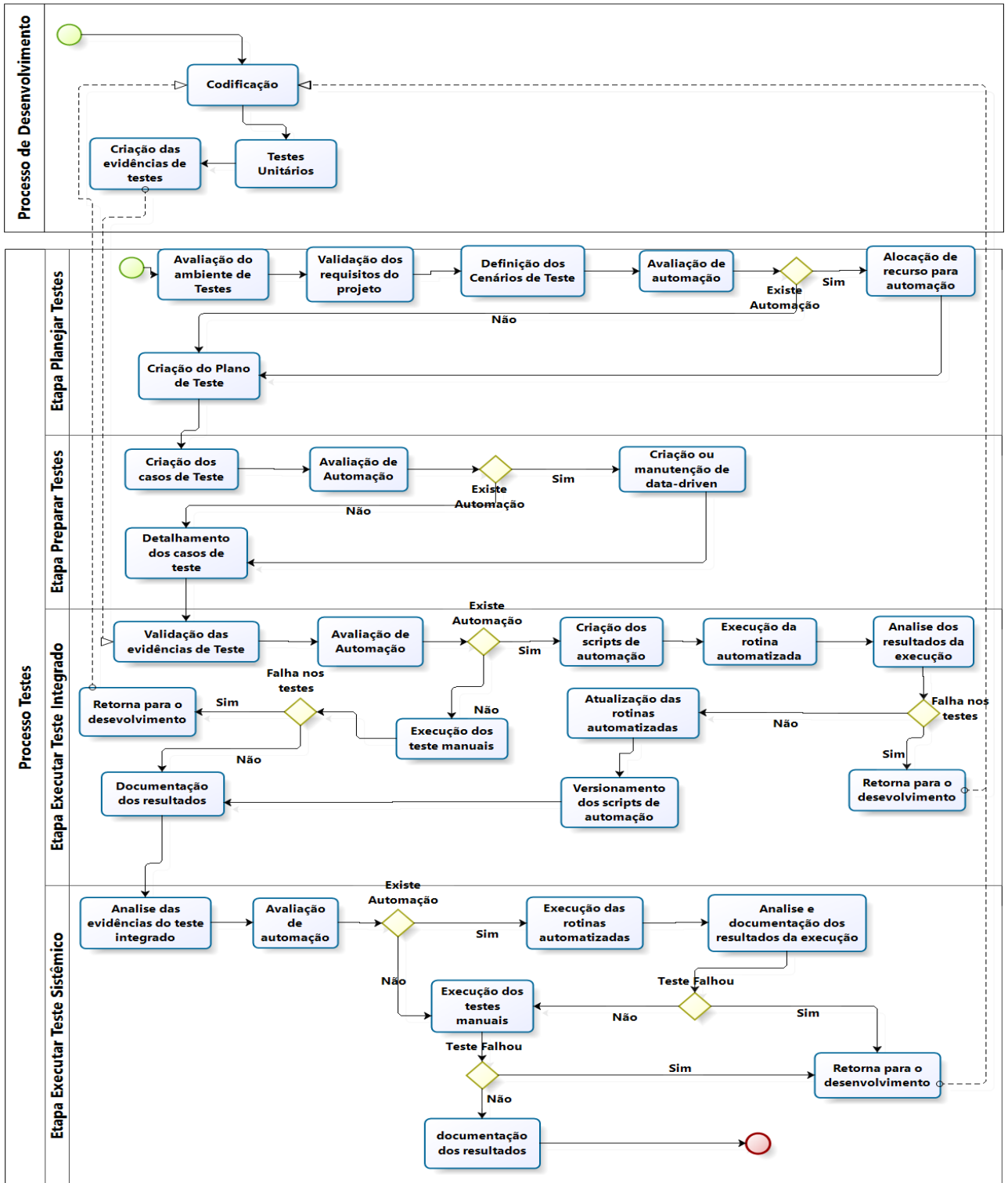
Figura 18 - Executar Teste Sistemico com atividades de automação



Fonte: O autor, 2016

A figura 19 representa todas as atividades que compõem o novo processo de testes que foi proposto. O processo de desenvolvimento foi representado apenas para se ter uma visão geral da integração entre o processo de testes e desenvolvimento.

Figura 19 - Processo para automação de testes



5 AVALIAÇÃO E DEFINIÇÃO DAS FERRAMENTAS

Nesta seção será apresentado o processo de definição dos critérios de avaliação de *software*, a partir das normas NBR ISO/IEC 9126-1 (Qualidade de Produto) e NBR ISO/IEC 14598-4 (Avaliação de Produto) complementadas com algumas diretrizes de seleção de ferramentas definidas por Molinari (2008), baseadas nos requisitos levantados na seção 4. Após serão apresentadas ferramentas de automação funcional necessárias para o processo de automação.

5.1 PLANEJAMENTO PARA AVALIAÇÃO DAS FERRAMENTAS

Para a escolha de ferramentas de automação de testes, Molinari (2008), descreve alguns passos básicos que devem ser seguidos:

- Definir requerimentos iniciais considerando itens como compatibilidade com o ambiente de criação da aplicação, sistema operacional, perfil de quem usará a ferramenta, assim como a disponibilidade para aquisição de ferramentas proprietária e particularidades do negócio.
- Investigar opiniões sobre a ferramenta considerando a documentação técnica disponível, opinião de usuários, tendências de revista especializadas, conferências entre outros.
- Refinar os requerimentos apresentando as ferramentas pré-selecionadas para os envolvidos, com o intuito de verificar se as mesmas vão de encontro com os requerimentos funcionais e orçamentários.
- Refinar a lista de ferramentas avaliando e pontuando cada uma de acordo os critérios definidos e informações levantadas.

A NBR ISO/IEC 14598-4, define que a quantidade de ferramentas selecionadas consideradas adequadas para posterior avaliação, pode ser limitada por meio de revisões, observações, relatos de experiências de usuários, comentários e estudos em periódicos. Ainda de acordo com a ISO 14598, critérios de avaliação para a escolha do *software* devem ser definidos e pontuados. Os critérios a serem utilizados devem ser baseados em trabalhos já realizados,

documentação do fornecedor de *software*, testes prévios de funcionalidades, bem como experiência e conhecimento do avaliador.

Com base em trabalhos anteriores e no perfil da equipe de testes da empresa alvo, os critérios utilizados para a seleção das ferramentas foram classificados em Fundamental, Importante e Desejável, considerando a relevância de cada um para o processo de avaliação. Para o peso das notas de cada critério foram utilizados valores de ordem cúbica seguindo a metodologia de Molinari (2008), onde o mesmo afirma que esse tipo de técnica reflete de forma mais clara, a importância dos itens que estão sendo avaliados. Os Critérios classificados como fundamentais tem peso igual a 9, critérios importantes têm peso 3 e critérios desejáveis têm peso 1. Os atributos de cada ferramenta são pontuados com valores de 0 a 10, sendo que esse valor é multiplicado pelo peso do atributo definido através da sua relevância. Os critérios incluem características funcionais e não funcionais baseados nos requisitos enumerados na seção 4.

Os critérios classificados como fundamentais são:

- * *Record-Playback* funcionalidade necessária para gravação dos testes funcionais e geração de scripts de automação
- * Automação de aplicações *Desktop* (Windows) para possibilitar automação em partes do sistema que são executadas como cliente server
- * Automação de aplicações web e navegadores suportados possibilitando a automação das funcionalidades do sistema projetados para web assim como fornecer suporte aos diversos navegadores que possam ser utilizados pelos usuários do sistema.
- * Suporte a Data-Driven para possibilitar diversas entradas de dados para o mesmo cenário sem a necessidade de recriar o teste
- * Integração com outras ferramentas que possam vir a ser utilizadas no processo de testes como ferramentas de integração contínua e planejamento de testes

Os critérios considerados como importantes são:

- * Linguagem de script preferencialmente deve ser Java ou C# para facilitar o uso na empresa alvo
- * Recursos de relatórios com gráficos e informações detalhadas para auxiliar na análise dos dados gerados a partir execução dos scripts
- * Suporte a debug para depuração dos scripts caso necessário. A documentação disponível sobre a ferramenta como documentação técnica, fóruns, tutoriais a fim de auxiliar no uso e treinamento de quem manipulará a ferramenta
- * Custo financeiro e a política de licenciamento, não existe a necessidade que a ferramenta seja sem custo ou código aberto, mas o valor para aquisição deve ser o menor possível

Os critérios considerados desejáveis possuem menor relevância na escolha. Não são considerados indispensáveis mas ajudam a pontuar melhor a ferramenta, os requisitos classificados como desejáveis são:

- * Usabilidade e Desempenho da ferramenta com o objetivo de contribuir para o uso mais eficiente da ferramenta
- * Customização é um item a ser considerado, pois sempre existirão demandas específicas que não são atendidas pelas funcionalidades padrão da ferramenta selecionada

Tabela 1 - Critérios de avaliação das ferramentas de automação

Característica	Descrição do Critério	Classificação	Peso	Ideal
<i>Record-Playback</i>	Suporte à gravação e reprodução de ações executadas na aplicação com geração de scripts	Fundamental	9	10
Suporte a aplicações <i>desktop</i>	Automatizar aplicações <i>desktop</i> Windows	Fundamental	9	10
Navegadores Suportados	Navegadores suportados para automação web	Fundamental	9	10
Suporte a aplicações web	Automação de aplicações web	Fundamental	9	10
Linguagem de script	Suporte a Java e C#	Importante	3	10
Recursos de Relatório	Relatórios de acompanhamento	Importante	3	10
Suporte a Debug	Suporte ao mecanismo de debug	Importante	3	10
Código Aberto	Política de licença da ferramenta	Desejável	3	10
Usabilidade	Interface intuitiva e de fácil uso	Desejável	1	10
Desempenho	Uso das funcionalidades sem lentidão e travamentos	Desejável	1	10
Documentação	Documentação sobre a ferramenta	Importante	3	10
Custo Financeiro	Custo financeiro da ferramenta	Importante	3	10
Integração	Integração com outras ferramentas	Importante	9	10
Customização	Customização da ferramenta	Desejável	1	10
Data-Driven	Construção de testes com entrada de dados externas	Fundamental	9	10
Pontuação Máxima				730

Fonte: O autor, 2016

A pontuação máxima que uma ferramenta pode receber, para ser a que melhor atende os requerimentos especificados, é de 730 pontos. Como alguns critérios são características qualitativas, e a nota pode variar de acordo com a força e intensidade de cada atributo, não necessariamente a pontuação da ferramenta será a máxima permitida na avaliação.

Após a pontuação de cada ferramenta, foram consideradas as duas com maior pontuação para realização de testes na empresa alvo para definir qual delas melhor se integra ao ambiente que será automatizado.

5.2 APRESENTAÇÃO DAS FERRAMENTAS

A seleção das ferramentas se baseou em trabalhos correlatos ao tema de automação de testes funcionais visando buscar ferramentas que atendam aos requisitos básicos definidos para a avaliação.

Kaur & Gupta (2013) realizaram um estudo comparativo entre as ferramentas AutomatedQA e TestComplete com base em critérios como esforço envolvido, capacidade de geração de scripts, opções de relatório, velocidade e custo objetivando analisar os recursos suportados pelas ferramentas, minimizar os recursos na manutenção de roteiros de testes, além de aumentar a eficiência na reutilização dos scripts.

Dubey & Shiwani (2013) analisaram as ferramentas TestComplete e Ranorex com o objetivo de identificar os recursos suportados pelas duas ferramentas na automação de teste. Nessa análise foram considerados itens como capacidade de geração de scripts, opções de relatório e também a capacidade de reprodução dos scripts gerados

F da Silva & G Moreno (2012) realizaram um estudo sobre automação em processos ágeis onde foram elencadas várias ferramentas que podem ser utilizadas no processo de automação de testes. Nesse estudo foram consideradas as ferramentas de automação web Selenium e Sahi.

Xiaochun, Zhu e (2008) criaram um processo de automação de testes funcionais onde utilizaram a ferramenta Rational Functional integrando o processo de automação com o gerenciamento dos casos de testes.

Com base nesses trabalhos as ferramentas selecionadas para avaliação foram: Selenium, Ranorex, TestComplete, Sahi e Rational Functional Tester.

5.2.1.1 SELENIUM

Segundo Molinari (2011), o Selenium é uma ferramenta de automação de testes funcionais web, sendo que ela compõe um pacote maior denominado “Open QA Selenium”. O Selenium é uma ferramenta open source que nasceu com o objetivo de ser uma alternativa para as ferramentas pagas que dominavam o mercado (MOLINARI, 2010, SELENIUM).

O pacote Open QA Selenium está dividido em três módulos diferentes:

- **Selenium IDE:** é uma ferramenta criada como um plug-in do navegador Mozilla Firefox, permite gravar as ações executadas nesse browser em um script cujo conteúdo tem forma de tabela e assim repetir as ações gravadas.
- **Selenium Remote Control (RC):** permite salvar o que está gravado em outras linguagens, tais como C# e Java. A ferramenta possibilita alterar o que foi gravado anteriormente acrescentando loops, funções e bibliotecas que venham a ser criadas.
- **Selenium Grid:** permite distribuir os testes em diversos computadores executando-os de forma coordenada, reduzindo assim o tempo necessário para se testar vários navegadores ou sistemas operacionais

Figura 20 - Ilustração da ferramenta Selenium

Selenium WebDriver



Selenium IDE



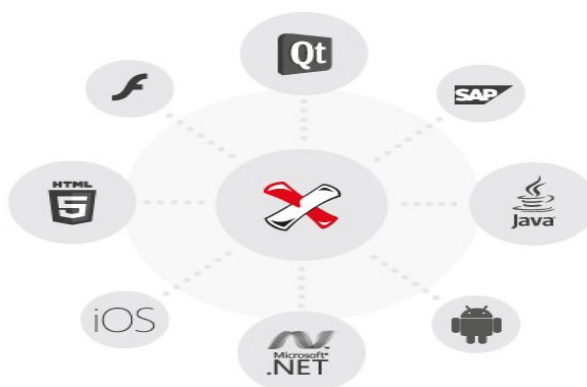
Fonte: <http://www.seleniumhq.org/>

5.2.1.2 RANOREX STUDIO

Ranorex é uma ferramenta de automação proprietária que possui suporte para testes funcionais *desktop*, *web* e *mobile*. Possui ferramenta de *Record-Playback* permitindo gravar as ações executadas no *desktop* ou no navegador sejam repetidas posteriormente.

Os scripts de automação podem ser gerados em C#, VBNet ou em linguagem própria da ferramenta. Além de suporte a testes funcionais a ferramenta também possui a opção de automação para testes data-driven, regressão entre outros.

Figura 21 - Ilustração ferramenta Ranorex



Fonte: <http://www.Ranorex.com/>

5.2.1.3 TESTCOMPLETE

TestComplete é uma ferramenta proprietária para automação de testes funcionais de *desktop*, *web* e *mobile*. Possui *Record-Playback* que permite gravar as ações executadas possibilitando gerar os scripts de automação em Python, VBScript, JScript, DelphiScript, C++Script e C#Script.

Figura 22 - Ilustração ferramenta TestComplete



Fonte: <http://smartbear.com/product/testcomplete>

5.2.1.4 SAHI

Sahi é uma ferramenta para automação de aplicações web semelhante ao Selenium. O Sahi utiliza um proxy para injetar Java script em páginas web, para executar os testes, possibilitando automação em todos os navegadores (SAHI).

Inicialmente a SAHI foi concebida como uma ferramenta open source, porém atualmente é uma ferramenta proprietária.

Figura 23 - Ilustração ferramenta Sahi



Fonte: <http://sahipro.com/>

5.2.1.5 IBM RATIONAL FUNCTIONAL TESTER

É uma ferramenta proprietária de automação testes funcionais e de regressão. Permite automatizar aplicações web, Java, .Net entre outras. Possui ferramenta de *Record-Playback* permitindo gravar as ações executadas gerando script que podem ser customizados posteriormente. Os scripts de automação podem ser gerados em Java ou VB.NET.

Figura 24 - Ilustração ferramenta IBM Rational



Fonte: <https://www.ibm.com/>

5.3 APLICAÇÃO DOS CRITÉRIOS DE AVALIAÇÃO E DEFINIÇÃO DAS FERRAMENTAS

Após analisar cada ferramenta seguindo as diretrizes descritas por Molinari (2008) unidas aos procedimentos definidos pelas normas NBR ISO/IEC

9126-1 (Qualidade de Produto) e NBR ISO/IEC 14598-4 (Avaliação de Produto), cada critério foi pontuado para cada uma das ferramentas. Os critérios de avaliação foram definidos de acordo com as necessidades e particularidades do projeto.

A pontuação varia de acordo com o nível de suporte que a ferramenta fornece para cada um dos atributos avaliados. Por exemplo, para as linguagens de scripts utilizadas, quanto maior for o suporte para Java e C# maior será a nota atribuída a ferramenta. Caso a ferramenta não tenha suporte a determinado atributo a nota atribuída será zero, como, por exemplo, no caso do atributo de código aberto.

Tabela 2: Pontuação da avaliação das ferramentas de automação

Característica	Peso	Selenium		Ranorex	TestComplete	Sahi	IBM Rational
<i>Record-Playback</i>	9	9		9	8	8	8
Suporte a aplicações <i>desktop</i>	9	0		10	9	0	9
Navegadores Suportados	9	10		10	8	8	8
Suporte a aplicações web	9	10		10	9	10	9
Linguagem de script	3	9		9	10	9	9
Recursos de Relatório	3	9		10	9	9	9
Suporte a Debug	3	10		10	9	9	9
Código Aberto	1	10		0	0	0	0
Usabilidade	1	10		10	9	9	8
Desempenho	1	10		10	9	10	8
Documentação	3	10		10	9	8	7

Custo Financeiro	3	10		6	6	6	6
Integração	9	9		10	8	6	8
Customização	1	10		4	10	8	8
Data-Driven	9	10		10	7	10	10
Pontuação Máxima	730	616		690	598	528	612

Fonte: O autor, 2016

Conforme a tabela comparativa nenhuma das ferramentas analisadas atingiu a pontuação máxima de 730 pontos. A grande maioria das ferramentas avaliadas são ferramentas proprietárias, sendo que apenas uma das ferramentas selecionadas possui código aberto. Algumas dão suporte para automação apenas em ambiente web enquanto outras dão suporte para ambos os ambientes (*desktop* e *web*). Outro fator, foi o custo de cada ferramenta variando de acordo com cada fornecedor.

Todas as ferramentas selecionadas dão suporte para Java ou C#, sendo que em alguns casos a ferramenta suporta as duas linguagens. Todas possuem mecanismo de *Record-Playback*, assim como dão suporte a Data-Driven. Possibilitam também a integração com outras ferramentas que podem auxiliar no processo de testes como, por exemplo, ferramentas de versionamento de código fonte.

As duas ferramentas melhores pontuadas foram a Selenium e a Ranorex. Apesar de a Selenium ser código aberto, a mesma só atende as interfaces web. A Ranorex por sua vez suporta a interface web e *desktop*, além de possibilitar a integração com várias das ferramentas já utilizadas na empresa como Jenkins, Jira e Team Foundation Server. Possui interface intuitiva e de fácil uso além de ter uma vasta documentação incluindo tutoriais em vídeo e documentação técnica disponibilizados pelo próprio fornecedor. Permite gerar scripts de automação em C# e VBNet que são linguagens que alguns dos integrantes da equipe de testes tem conhecimento. A ferramenta tem boas opções de relatório para auxiliar na validação dos resultados gerados através dos scripts automatizados, conta também com a opção de data-driven o que auxilia na criação de variações nos casos de teste.

Também foram realizados alguns testes superficiais com o Ranorex, objetivando verificar o comportamento da mesma no ambiente da empresa alvo do estudo. Nos testes a ferramenta reproduziu os scripts automatizados sem apresentar problemas.

Por esses motivos e baseado na pontuação, a Ranorex foi a ferramenta selecionada para auxiliar no processo de automação.

6 APLICAÇÃO DA PROPOSTA

Na segunda etapa deste trabalho foram automatizados os casos de testes através da ferramenta selecionada anteriormente, utilizada integração contínua para auxiliar no processo de automação, desenvolvida ferramenta de controle de bugs e realizada integração entre ferramentas.

Para ser possível a aplicação da proposta, foram configurados todos os recursos necessários para o desenvolvimento de um ambiente de protótipo. Nesse protótipo foram instalados os sistemas alvos da automação, configurados os bancos de dados das aplicações. Também foram criadas estruturas de repositórios para melhor organizar os códigos fontes e configurada ferramenta para compilação de programas Progress.

Além disso foi instalada a ferramenta Ranorex utilizada para criação dos scripts de automação, e configurado o Jenkins para dar suporte ao processo de integração contínua.

Durante a aplicação da proposta surgiram algumas necessidades como a integração entre as ferramentas de automação e integração contínua, assim como o desenvolvimento de uma ferramenta para controle de bugs. A integração entre ferramentas é detalhada no capítulo 7 deste trabalho.

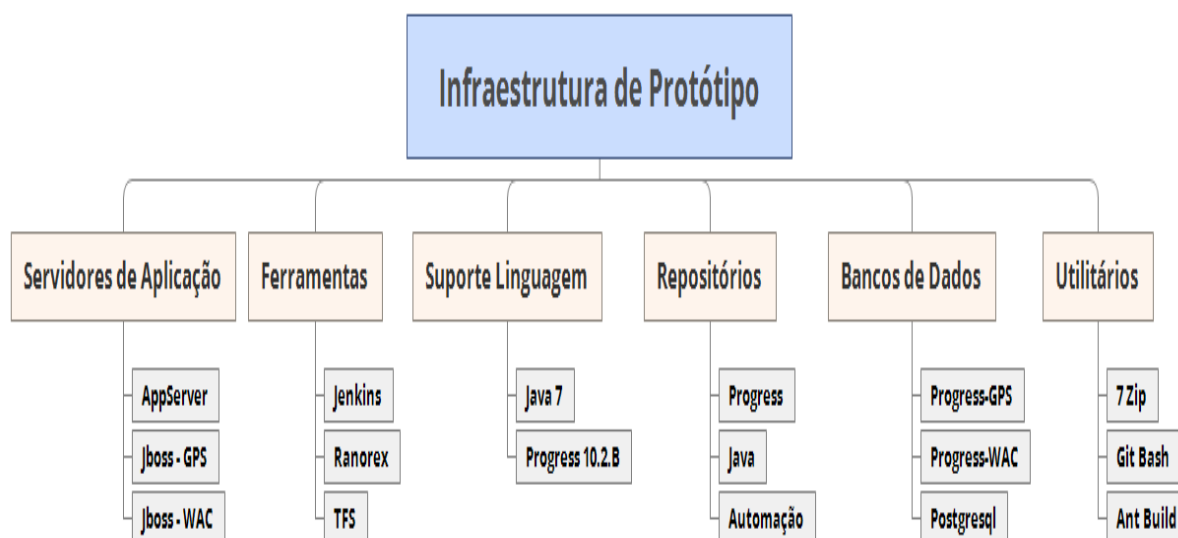
As atividades para a aplicação da proposta se dividiram em três partes: Ambiente de Protótipo, Automação de Casos de Teste e Integração Contínua.

6.1 AMBIENTE DE PROTÓTIPO

Seguindo as boas práticas de automação foi criado um ambiente de testes isolado para garantir a estabilidade da aplicação a ser testada. Para isso a empresa alvo disponibilizou um Servidor com Windows Server R2.

Neste servidor foram instalados os servidores de aplicação, banco de dados, ferramentas de integração continua, versionamento de código fonte e automação. Também foram criados repositórios para compilação e scripts de automação, e instaladas ferramentas utilitárias. A infraestrutura do protótipo criado é representada na figura 25.

Figura 25 - Infraestrutura do Protótipo



Fonte: O autor, 2016

Considerando a estrutura necessária para dar suporte ao processo de automação e integração contínua a criação do ambiente de protótipo foi dividido em cinco partes: Sistemas, Banco de Dados, Estrutura de Diretórios, Compilação Progress e Instalação de Ferramentas.

6.1.1 SISTEMA

A empresa alvo do estudo desenvolve sistemas voltados para Gestão de Planos de Saúde. Esses sistemas abrangem desde a parte cadastral de beneficiários, coberturas dos planos de saúde, atendimentos realizados pela operadora e prestadores vinculados a mesma, até o pagamento e cobranças sobre usos e serviços prestados.

Os produtos da empresa são divididos em módulos conforme a afinidade com as regras de negócio. Apesar dos sistemas possuírem partes em plataforma web a grande maioria de suas funcionalidades ainda é *desktop*, sendo que essas são desenvolvidas em Progress 4GL. Os módulos web são desenvolvidos em diversas tecnologias tendo o Java como principal.

Para dar suporte aos sistemas a serem automatizados foram configurados três servidores de aplicação, sendo que desses, dois são JBOSS para aplicações Java e um Progress Application Server para a parte Progress. Conforme orientações de

instalação da empresa foi utilizado o JBOSS versão 4.2 juntamente com o Java versão 7. O servidor de aplicação Progress utilizado foi o da versão 10.2B.

Os servidores de aplicação JBOSS dão suporte para dois sistemas distintos, sendo um desses desenvolvido em Progress 4GL, mas possuindo partes também desenvolvidas em Java, Flex e Html5, e o outro sistema desenvolvido exclusivamente em Java. Apesar de serem sistemas diferentes existem bancos de dados compartilhados entre as aplicações, assim como maior parte do processamento ainda é realizado por uma camada Progress.

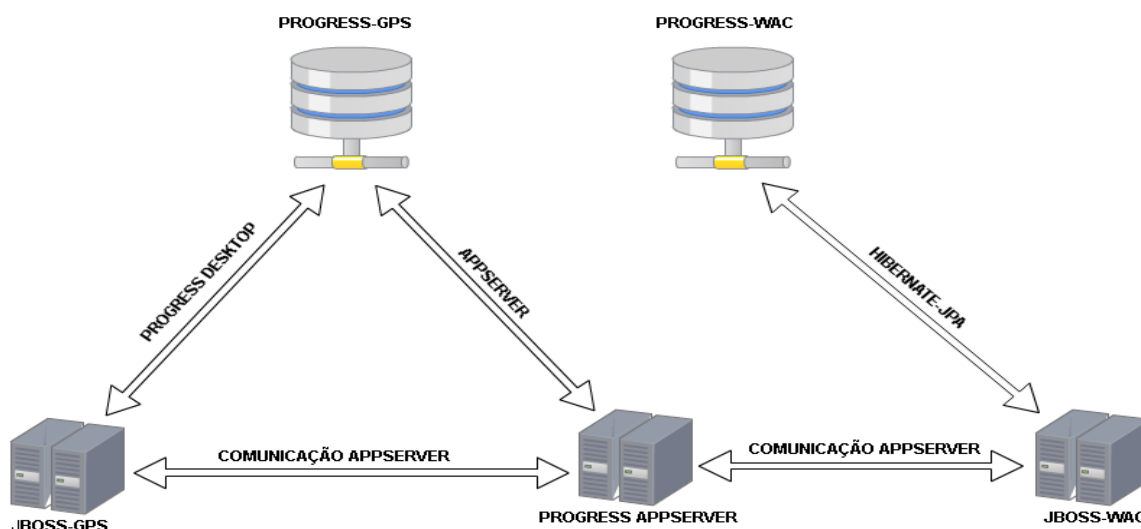
Considerando que existem soluções desenvolvidas em Progress presentes nos dois sistemas, foi necessária configuração de um Progress Application Server (AppServer). Este servidor se integra aos servidores JBOSS para processar as requisições que necessitam de interação com a camada Progress.

Levando em consideração a arquitetura dos dois sistemas, a comunicação com o banco de dados pode ser realizada de 3 formas:

1. A primeira é a comunicação da parte Progress *Desktop* com o banco de dados, ou seja, não existe interação entre o JBOSS e o AppServer, mas sim uma interação entre o Progress *Desktop* e o JBOSS.
2. A segunda é o tipo de persistência que utiliza o AppServer para se comunicar com o banco de dados, ou seja, são as aplicações desenvolvidas em Java ou outras linguagens que não interagem diretamente com o banco de dados necessitando da camada Progress para processar as informações.
3. Por fim se tem um terceiro tipo de conexão onde a camada Java interage diretamente com o banco de dados utilizando JPA e Hibernate, sem a necessidade de chamadas para o AppServer.

A figura 26 é uma representação do fluxo de comunicação entre as aplicações configuradas e os bancos de dados utilizados pelas mesmas.

Figura 26 - Comunicação com o Banco de Dados



Fonte: O autor, 2016

6.1.2 BANCO DE DADOS

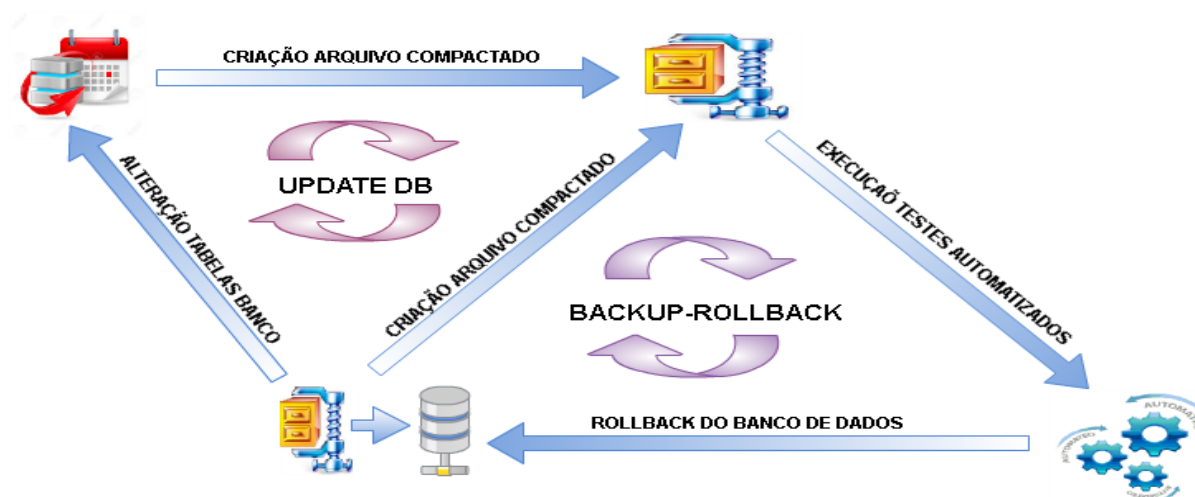
Os Bancos de dados utilizados no protótipo foram bancos de dados Progress, a instalação e configuração foi realizada conforme as necessidades dos sistemas a serem testados. Para popular os bancos de dados das aplicações foram utilizados dados preexistentes, optou-se por isso devido à complexidade das regras de negócio envolvidas, e também pela quantidade de informações que seriam necessárias cadastrar para que o ambiente se tornasse funcional.

Além da instalação dos bancos de dados também foi criada uma estrutura de backup para que seja possível realizar o Rollback das informações inseridas pelos testes automatizados. O processo de criação de backup e Rollback consiste basicamente em criar um arquivo compactado do banco de dados e posteriormente sobrescrever os dados com o arquivo criado. Dessa forma é possível reexecutar os testes quantas vezes forem necessárias.

Considerando o tamanho e o tempo para criação do arquivo de backup esse processo não é executado diariamente, ou seja, o arquivo é criado apenas quando existe a necessidade de uma nova cópia com dados persistidos, ou quando o esquema do banco é alterado. Uma vez o arquivo criado, o mesmo é reutilizado nos processos diários de Rollback.

Quando existem alterações no banco de dados proveniente de novas funcionalidades ou manutenção no sistema, essas alterações são replicadas para o ambiente de protótipo executando-se os scripts de atualização do banco de dados. Após esse processo é necessária a criação de um novo arquivo para Rollback, uma vez que o esquema de tabelas foi alterado, com o arquivo criado o processo de Rollback é executado normalmente. A criação do arquivo de backup e a atualização do banco de dados é executada manualmente pelo analista de infraestrutura. O processo de atualização e Rollback do banco de dados é representado na figura 27.

Figura 27 - Rollback e Update do Banco de Dados



Fonte: O autor, 2016

6.1.3 ESTRUTURA DE DIRETÓRIOS

Com a finalidade de melhor estruturar as pastas que contém os códigos fontes que são compilados, e posteriormente atualizados nas aplicações a serem testadas, foram definidos alguns diretórios padrões. Todos os diretórios e projetos vinculados a automação foram criados em uma unidade de disco separada.

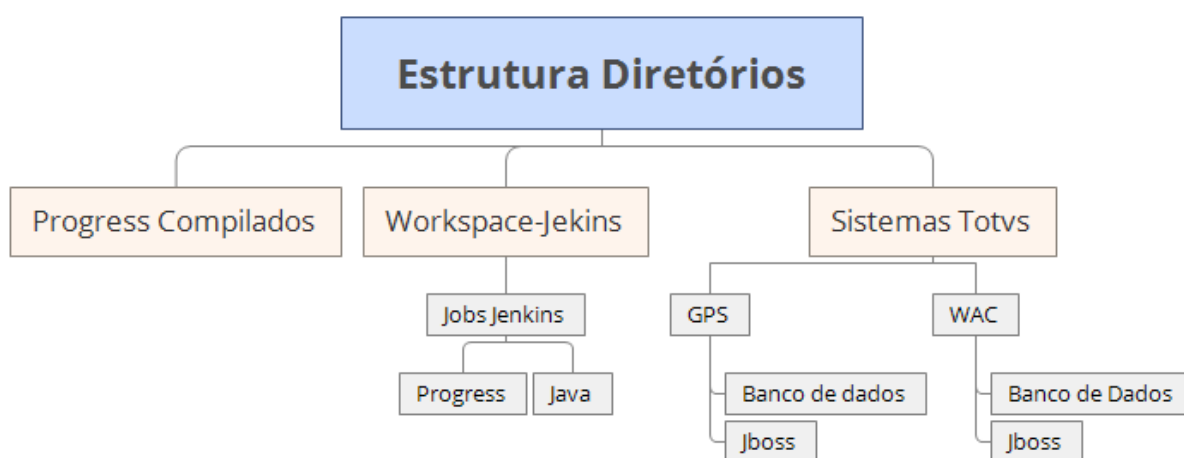
Como é o processo de integração contínua que faz o download e a compilação dos fontes, foi definida uma workspace para esse processo. Nessa workspace¹³ o Jenkins que é a ferramenta de integração contínua cria os diretórios para download dos fontes de cada um de seus Jobs. Os fontes Progress não compilados e fontes

¹³ Workspaces são estruturas de diretórios utilizados pelo Jenkins para armazenar e controlar os artefados gerados pelos seu projetos, como por exemplo, códigos fontes de integração contínua, scripts gerados pela ferramenta entre outros.

Java ficam nessa estrutura de pastas criadas pelo Jenkins, já os fontes Progress compilados ficam em um diretório fora dessa workspace.

Nessa unidade de disco também foram configurados os sistemas a serem automatizados. Essa configuração consiste em instalar o JBOSS e configurar o banco de dados de cada umas das aplicações. As estruturas de pastas para os sistemas foram criadas seguindo os padrões de instalação definidos pela empresa. A figura 28 exemplifica a estrutura de diretórios criado.

Figura 28 - Estrutura de Diretórios



Fonte: O autor, 2016

6.1.4 PROCESSO DE COMPILAÇÃO PROGRESS

Para compilação dos programas Progress a empresa faz uso de uma ferramenta desenvolvida internamente, denominada SyncProgress. Essa ferramenta utiliza dois diretórios, um contendo os fontes a serem compilados, e outro para os programas que passaram pelo processo de compilação. Além disso, possui um banco de dados que gerencia as atualizações desses diretórios. Os fontes só passam pelo processo de compilação caso tenham sido modificados ou não tenham sido compilados.

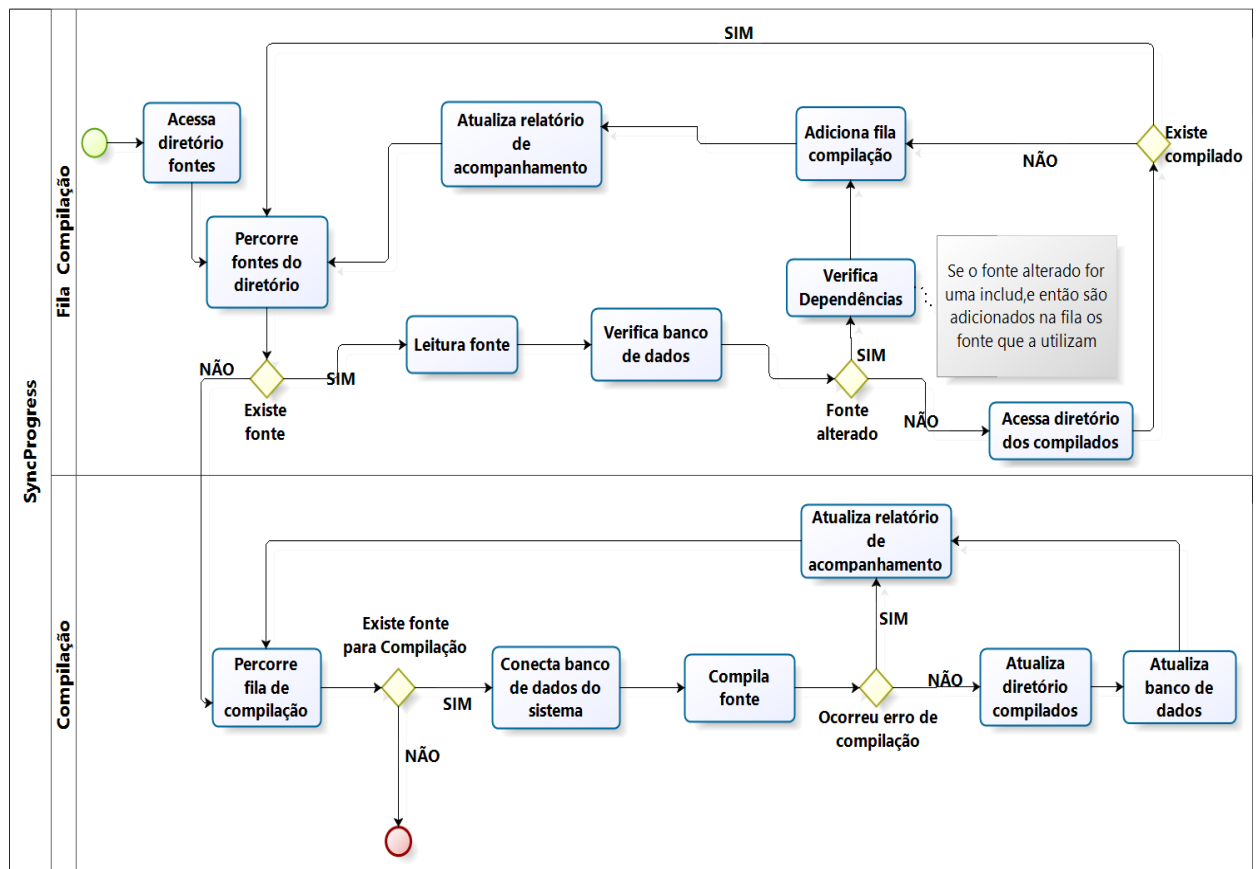
Uma vez iniciado o processo para compilação a ferramenta faz uma varredura no diretório de fontes a serem compilados, comparando a data de modificação dos arquivos com os registros do banco de dados. Caso estejam diferentes, ou não possuam registros os programas são adicionados na fila de compilação.

Também são verificadas bibliotecas que podem ser utilizadas por vários programas, se as mesmas foram modificadas, todos os programas que as utilizam são adicionados na fila de compilação.

No ambiente em questão, para os fontes não compilados, utilizou-se os diretórios do Jenkins, que são atualizados pelo processo de integração contínua. Para os compilados foi parametrizado um diretório criado fora da workspace do Jenkins.

Com a identificação dos fontes que devem ser compilados, o SyncProgress conecta-se com o banco de dados e inicia o processo de compilação, criando o arquivo compilado no diretório parametrizado para este fim. No protótipo o SyncProgress utiliza os bancos de dados dos sistemas a serem automatizados para a compilação. Por esse motivo, é importante que os bancos de dados se mantenham sempre atualizados. O processo do SyncProgress é ilustrado na figura 29.

Figura 29 - Processo de compilação do SyncProgress



A ferramenta também gera um relatório de acompanhamento listando quais programas foram considerados para compilação e listando erros caso existam. A figura 30 exemplifica os dados gerados no relatório pela ferramenta. Um exemplo completo desse relatório encontra-se nos *Anexos H e I*.

Figura 30 - Relatório de Acompanhamento SyncProgress

```
15/11/2016 19:18:05 - afp\af0800b.p nao possui .r equivalente no directorio de compilados. Sera compilado.
15/11/2016 19:18:05 - api\api-administrativeintegration.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:32 - fch\fchsauMD\fchsausecondcontractrequest.p sera recompilado pois declara bosau/bosaubenefs.i
15/11/2016 19:18:32 - fch\fchsauMD\fchsauunifyperson.p sera recompilado pois declara bosau/bosaubenefs.i
15/11/2016 19:18:45 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800a.p. Salvar .r em: C:\compilados\afp\Situa#Eo programa: 0
15/11/2016 19:18:48 - ** Unknown or ambiguous table dwf-emit. (725)
15/11/2016 19:18:48 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800a.p Nao entendi a linha 1711. (196)
15/11/2016 19:22:05 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atp\at0112c1.p. Salvar .r em: C:\compilados\atp\Situa#Eo programa: 0
15/11/2016 19:22:06 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atp\at0112c2.p. Salvar .r em: C:\compilados\atp\Situa#Eo programa: 0
```

Fonte: O autor, 2016

6.1.5 INSTALAÇÃO DE FERRAMENTAS

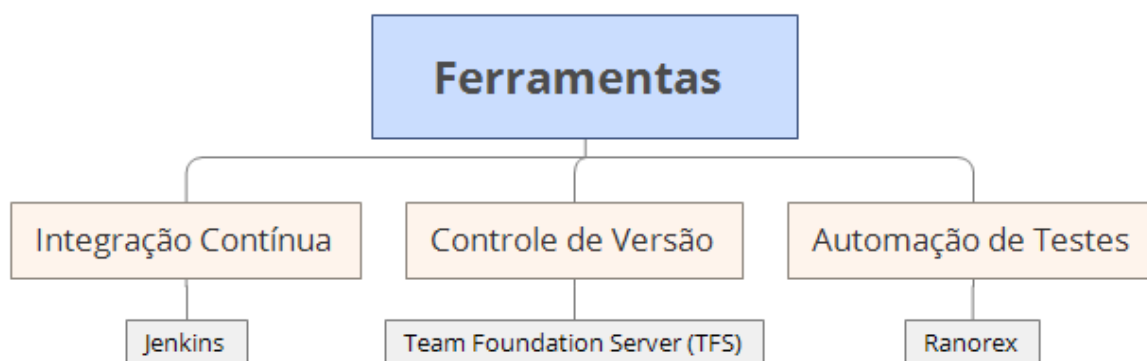
Neste ambiente também foram instaladas as ferramentas necessárias para o processo de automação e integração contínua. O Jenkins foi instalado e configurado para dar suporte ao processo de integração contínua, a instalação e configuração dessa ferramenta é descrita na sessão 6.3.

O controle de versões dos sistemas da empresa é realizado através do Team Foundation Server (TFS), dessa forma foi instalado um Client dessa ferramenta para dar acesso aos códigos fontes dos sistemas a serem testados, assim como realizar versionamento dos scripts de automação.

A automação dos testes foi realizada através do Ranorex, a instalação e configuração foram seguindo as orientações disponibilizada pelo fornecedor¹⁴. A figura 31 representa as ferramentas instaladas no ambiente.

¹⁴ <http://www.Ranorex.com/>

Figura 31 - Principais Ferramentas do Protótipo



Fonte: O autor, 2016

Como ferramentas de apoio foi configurado o Ant Build¹⁵ para auxiliar na construção de projetos Java e criação de scripts que auxiliam na integração entre o Jenkins e o Ranorex. O 7 Zip¹⁶ foi instalado para o processo de Rollback do banco de dados, e o Git Bash¹⁷ para criação de scripts que auxiliam na atualização dos servidores de aplicação.

Durante o processo de automação e integração entre ferramentas, foi verificada a necessidade de que os dados dos testes executados deviam ser persistidos para análises futuras. Dessa forma, foi desenvolvida uma ferramenta que utiliza o banco de dados PostgreSQL para armazenar essas informações. A instalação desse banco de dados foi feita conforme orientação do fornecedor¹⁸, e a descrição da criação e funcionamento da ferramenta é descrita no capítulo 7.

Figura 32 - Ferramentas de Apoio



Fonte: O autor, 2016

¹⁵O Ant é uma ferramenta de automação de construção de software desenvolvida em java. A documentação sobre essa ferramenta pode ser acessada através do endereço: <http://ant.apache.org/>

¹⁶ <http://www.7-zip.org/>

¹⁷ <https://git-for-windows.github.io/>

¹⁸ <https://www.postgresql.org/>

6.2 AUTOMAÇÃO DE CASOS DE TESTE

Após a configuração dos sistemas a serem automatizados e da instalação das ferramentas de automação foi iniciado o processo de criação dos scripts de automação. Esse processo foi dividido em 2 partes Casos de testes e Scripts de automação.

6.2.1 CASOS DE TESTES

Para gerenciar e documentar os casos de testes a empresa utiliza a ferramenta Jira, que possui um plug-in denominado Kanoah¹⁹ que auxilia nesta tarefa. O Kanoah possibilita criar casos de testes de forma detalhada e sequencial, assim como informar se o teste é automatizado ou não.

Como essa ferramenta está dentro do processo corporativo de testes, optou-se por não alterar a forma como a equipe cria e gerencia os casos de teste. A única alteração neste processo, foi a de sinalizar na própria ferramenta que o caso de teste é automatizado, assim é possível realizar os ajustes necessários caso ocorram alterações que impactem nos scripts de automação.

No Kanoah, os casos de teste estão divididos em módulos que compõem o sistema da empresa, de forma a facilitar o gerenciamento por parte da equipe. Cada caso de teste possui um nome, objetivo, pré-condição, o módulo ao qual pertence, prioridade e informação se é automatizado.

Além dessas informações possui um script detalhado do passo a passo para execução do caso de testes no sistema. Esse script contém nomes de telas, dados que devem ser inseridos, e quais as funcionalidades devem ser avaliadas nos testes.

Também possui registros dos resultados das execuções manuais, onde é informado o status da execução dos testes, ou seja, falha ou sucesso. A figura 33 exemplifica um caso de teste criado no plug-in Kanoah.

¹⁹ O Kanoah é um plug-in desenvolvido para o Jira permitindo o gerenciamento de todo o processo de testes na ferramenta. Mais informações sobre a ferramenta podem ser acessadas no seguinte endereço: <https://www.kanoah.com/>

Figura 33 - Caso de Teste no Kanoah

Gestão de Testes - Saúde / Test Cases / G TSAU-T488

PR0110M – Manutenção Graus de Parentesco

Details Test Script Test Results Coverage Attachments Change History

▼ Name

PR0110M – Manutenção Graus de Parentesco

▼ Objective

Neste cadastro são definidos os graus de parentesco que serão disponibilizados nos contratos para posterior relacionamento com os beneficiários;

▼ Precondition

Click to type the precondition

▼ Details

Folder	Status	Priority	Component	Owner
/Datasul/HPR	Liberado	Normal	HPR	Daniel Luciano de ...

Labels

HPR *Click to add labels*

▼ More Information

Teste Automatizado

Fonte: Totvs (2016)

Um ponto importante sobre os casos de testes registrados, é que os mesmos representam várias ações sobre uma determinada tela, ou seja, na maioria das vezes o caso de teste representa todas ações de um cadastro como incluir dados, alterar e excluir. Essas ações são detalhadas na aba Test Script dos casos de testes existentes no Kanoah. Pelo fato dos registros serem genéricos os mesmos foram divididos em casos de testes menores no momento da automação, ou ficaria inviável automatizar.

Na criação dos scripts automatizados um caso registrado na ferramenta pode derivar vários casos de testes automatizados, quanto menor for o caso de teste automatizado maior é o controle sobre as possíveis falhas que podem ocorrer no momento da execução. Por isso é importante que quando ocorram alterações em

um caso de teste registrado do Kanoah sejam revisados os casos automatizados vinculados a ele.

Para um maior controle sobre a automação foi sugerido que os novos casos de testes criados no Kanoah não sejam genéricos. Os mesmos devem representar ações menores facilitando assim a geração e atualização dos scripts automatizados, uma vez que um caso de testes do Kanoah, estará vinculado a apenas um caso de teste automatizado. A figura 34 é um exemplo de como o script do passo a passo é detalhado no Kanoah.

Figura 34 - Script de Teste

Gestão de Testes - Saúde / Test Cases / G TSAU-T488

PR0110M – Manutenção Graus de Parentesco

Details **Test Script** Test Results Coverage Attachments Change History

STEP

O programa deve possuir as seguintes funções:

- Detalhar
- Adicionar
- Alterar
- Remover

Os seguintes campos devem ser apresentados:

- **Cód Grau de Parentesco** - Código do grau de parentesco.
- **Descrição** - Descrição do grau de parentesco.
- **Tipo dependência** - Tipo de dependência do grau de parentesco. Devem ser apresentadas as seguintes opções de seleção:
 - 1 - RESPONSAVEL
 - 2 - CONJUGE/COMPANHEIRO
 - 3 - FILHO
 - 4 - FILHO ADOTIVO
 - 5 - PAI/MAE
 - 6 - OUTROS
- **Gera Parto Coberto** - Indicador de parto coberto.
- **Observação** - Campo de observações do registro.

Fonte: Totvs (2016)

Para a criação dos scripts foram selecionados alguns casos de casos de testes que compõem um processo de negócio da empresa. Não foram selecionados casos

considerados complexos, onde para que se fosse possível realizar estes testes seria necessário o cadastro de uma grande quantidade de dados para a sua execução. E também como não se tinha um domínio total sobre a ferramenta de automação, isso tornaria o processo improdutivo.

Outro ponto importante que deve ser salientado, é que os casos de testes foram selecionados com o objetivo de demonstrar a viabilidade e funcionamento da ferramenta no ambiente da empresa.

Os casos de testes existentes foram divididos em casos menores conforme as funcionalidades básicas das telas do sistema, ou seja, cada caso de teste do Kanoah gerou três scripts automatizados representando as ações de inclusão de dados, alteração e exclusão.

6.2.2 SCRIPTS DE AUTOMAÇÃO

Com a instalação da Ranorex que é ferramenta utilizada para a automação e definidos os casos de testes iniciais que seriam automatizados se deu início a criação dos scripts.

O processo de criação dos scripts de automação foi dividido em três partes: Estrutura dos Projetos de Automação, Gravações das Funcionalidades e Criação de Casos Testes.

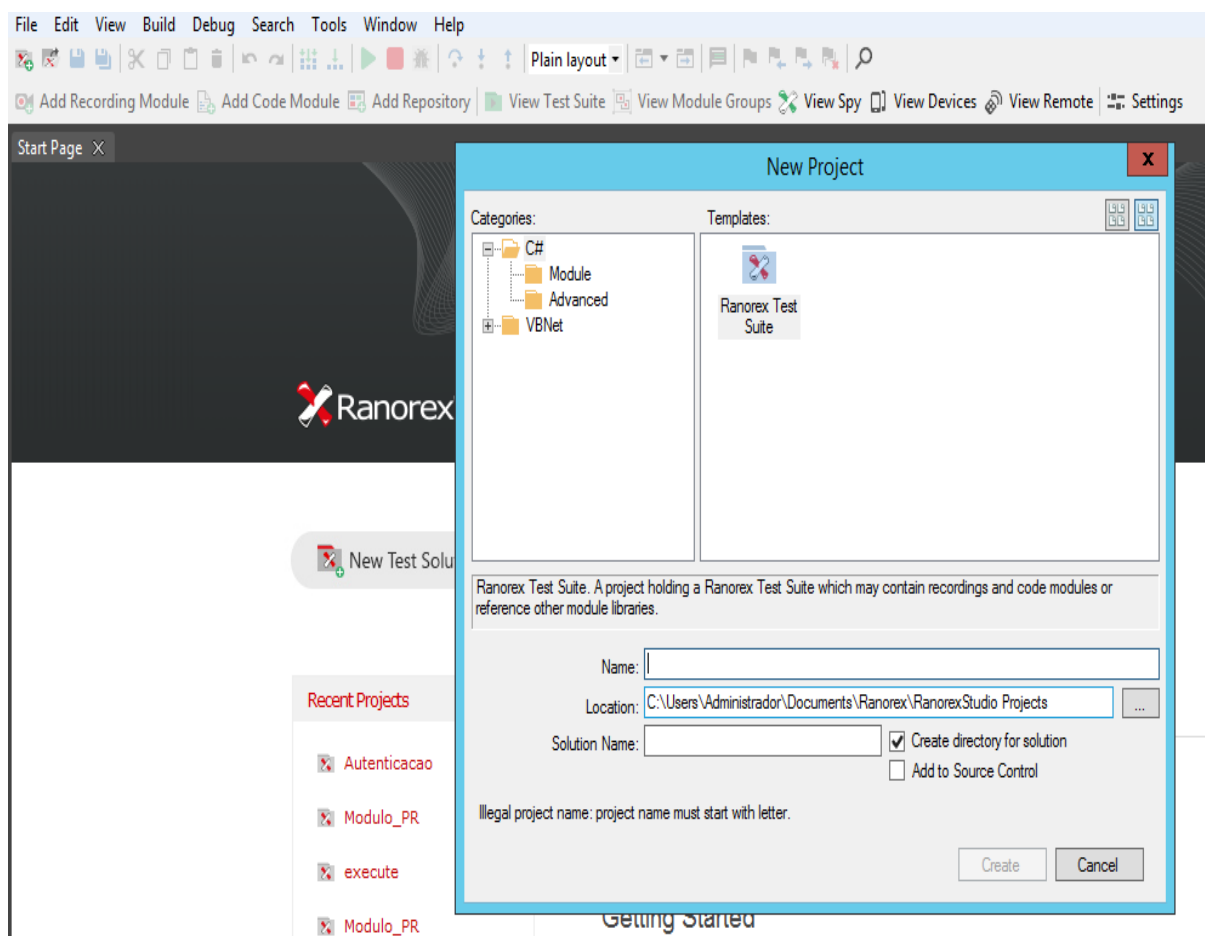
6.2.2.1 ESTRUTURA DOS PROJETOS DE AUTOMAÇÃO

A ferramenta possui uma interface intuitiva com uma curva de aprendizagem consideravelmente rápida. Ela utiliza o conceito de Test Solution para agrupar os projetos de automação. Cada Test Solution pode conter um ou mais projetos de automação, porém cada projeto só pode ser criado dentro de uma Test Solution.

O Ranorex trabalha com duas linguagens de programação em seus projetos: C# ou VBNet. Além de Solutions ela também possibilita a criação de módulos e bibliotecas nessas duas linguagens.

Considerando as linguagens de programação suportadas pela ferramenta optou-se por utilizar o C# para a criação das Tests Solutions e Projetos devido a ser linguagem que os membros da equipe de testes possuíam maior familiaridade. A figura 35 representa a interface de criação de projetos na ferramenta.

Figura 35 - Criação de Projetos Ranorex



Fonte: O autor, 2016

Os projetos do Ranorex basicamente são divididos em repositório de componentes de tela, arquivos de gravação de testes, componentes de referência que podem ser plug-ins de integração com outras ferramentas, diretório de Reports que são os logs de execução dos casos de teste, CodeModule que são os scripts criados manualmente e Test Suite que são os casos de testes criados na ferramenta.

Considerando que todos os arquivos por padrão são criados na raiz do projeto foram definidas algumas boas práticas de estruturação, definindo pastas padrões para organizar melhor os componentes gerados pela ferramenta. Os diretórios definidos foram:

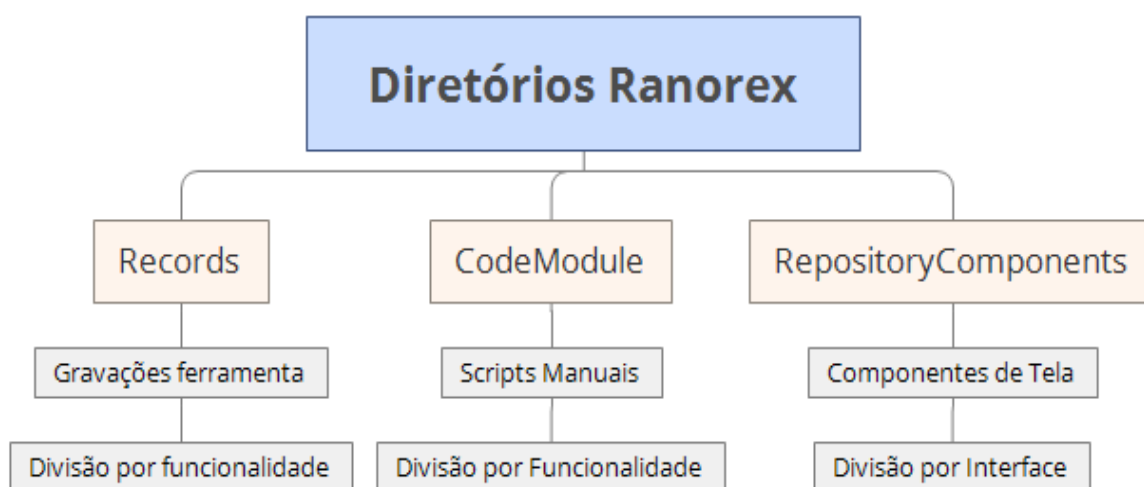
- *Records*: onde devem ficar as gravações dos testes da ferramenta, sendo que esse diretório deve ser subdivido por tela do sistema, ou seja, as

gravações de uma determinada tela devem ficar em um subpasta com o seu nome

- CodeModule: diretório para agrupar os scripts gerados manualmente, este diretório também deve ser subdivido conforme as telas nas as quais o script é utilizado.
- RepositoryComponents: onde devem ser criados os repositórios dos componentes de tela. O nome dos repositórios deve seguir a seguinte nomenclatura nomeTela + Repository.

As pastas *Records* e *CodeModule* devem possuir um subpasta com o nome de útil onde devem ficar os arquivos que são utilizados por vários processos. A figura 36 representa a estrutura de diretórios definidos.

Figura 36 - Estrutura de Diretórios dos Projetos do Ranorex



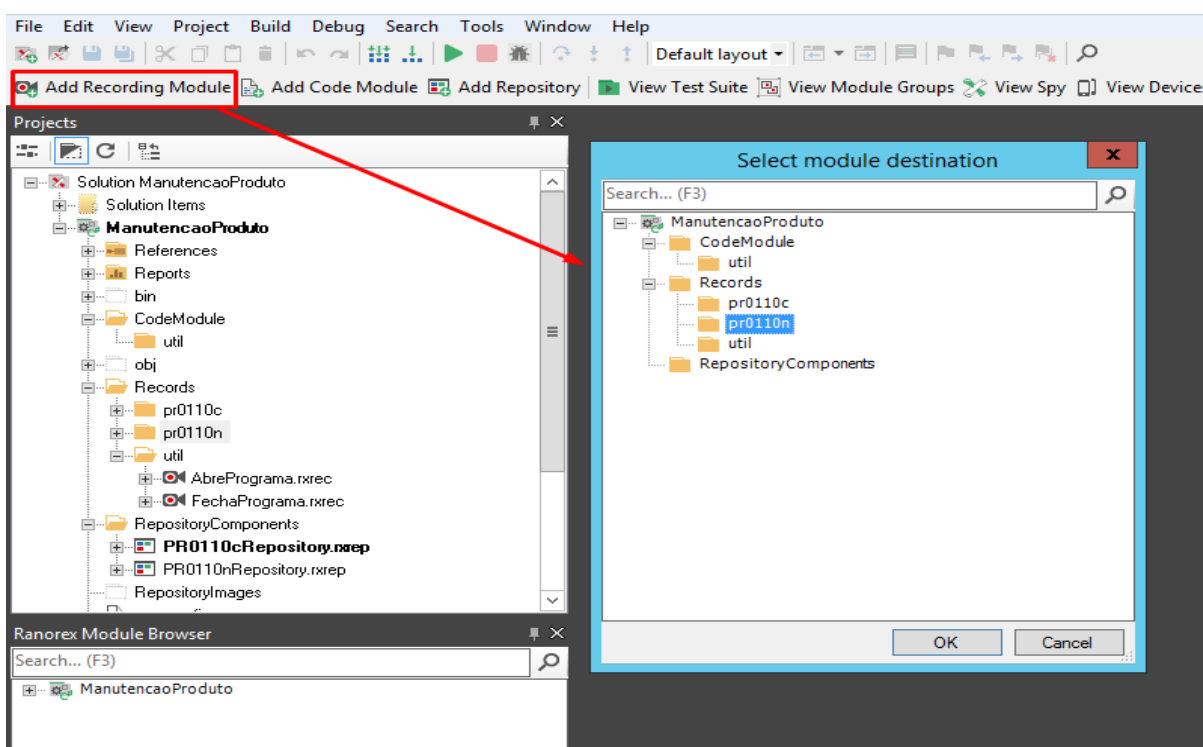
Fonte: O autor, 2016

Seguindo as boas práticas de programação os nomes das gravações e scripts criados devem ter nomes que descrevam a sua funcionalidade, por exemplo uma gravação que abre ou fecha uma determinada tela pode seguir o padrão de nome fechaProgramaNomePrograma. Dessa forma visualmente já se tem ideia de qual a utilidade do script.

6.2.2.2 GRAVAÇÕES DOS TESTES

Com a estrutura do projeto definido, foi iniciada a criação das gravações que derivaram os casos de testes automatizados. O processo de gravação disponibilizado pela ferramenta é de simples entendimento. Quando selecionada a opção de realizar nova gravação é solicitado o diretório onde a mesma deve ser salva. Conforme definido anteriormente as gravações devem ser ficar no diretório Reports na pasta correspondente a tela a qual irá se fazer a gravação.

Figura 37 - Criação de Gravação Ranorex



Fonte: O autor, 2016

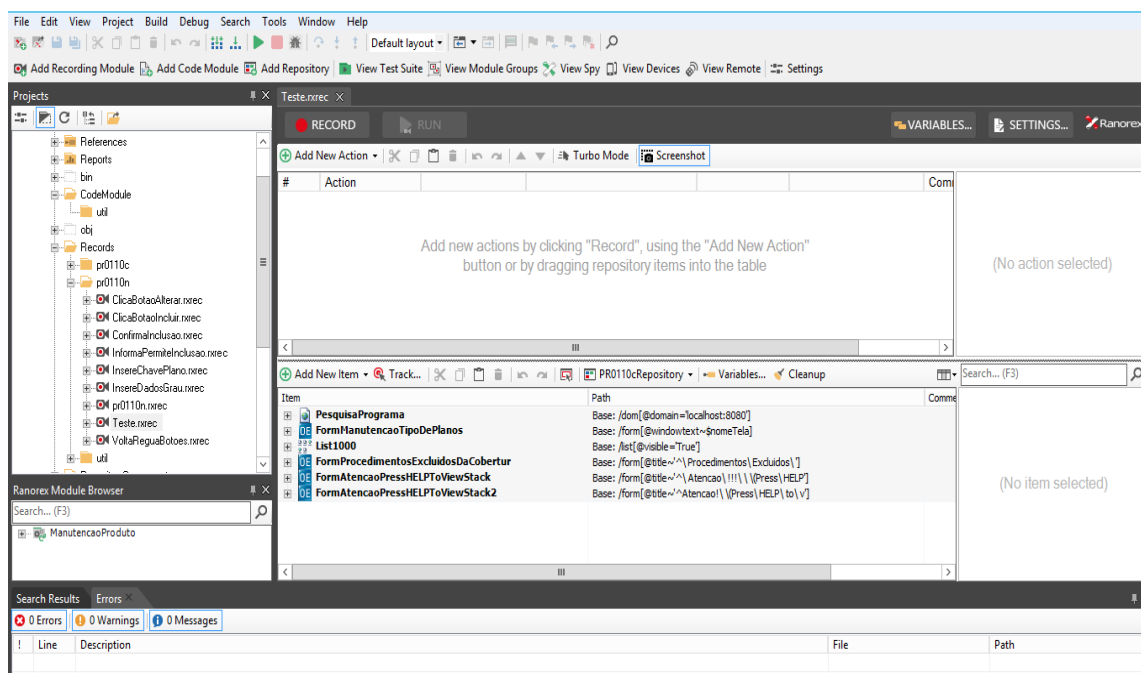
Com o arquivo de gravação criado a ferramenta direciona para a tela de edição de *Record*. Nela é possível gravar as ações do usuário, alterar gravações já existente incluindo, alterando ou excluindo componentes e ações do script. Também é possível criar ou modificar repositórios de componentes que são utilizados na gravação.

Na edição de *records*²⁰ são definidas as variáveis dos scripts²¹, que são utilizadas com os arquivos de data-driven²², para inserção de valor nos campos que

²⁰ Os arquivos de *record* na Ranorex, se referem as gravações das ações dos usuários capturadas pela ferramenta

solicitam entrada de dados. Nessa tela também é possível executar, particionar ou depurar os scripts das gravações. A tela de gravação da ferramenta pode ser visualizada na figura 38.

Figura 38 - Edição de *Records*



Fonte: O autor, 2016

Como definido anteriormente, cada tela gravada deve possuir seu repositório de componentes, pois no momento da gravação a ferramenta mapeia um número grande de elementos de tela, e caso todos sejam salvos no mesmo repositório, isso inviabilizaria o gerenciamento dos mesmos, além de dificultar a edição dos *records* e criação de scripts manuais. Caso a gravação esteja sendo feita para uma tela que ainda não possui repositório o mesmo deve ser criado, caso já exista, deve ser reutilizado.

Uma vez criado o arquivo de *record* e definido o repositório que será utilizado pode-se iniciar a gravação das ações do usuário. Antes de iniciar, a ferramenta solicita qual o tipo de aplicação está sendo gravada. As opções disponibilizadas são:

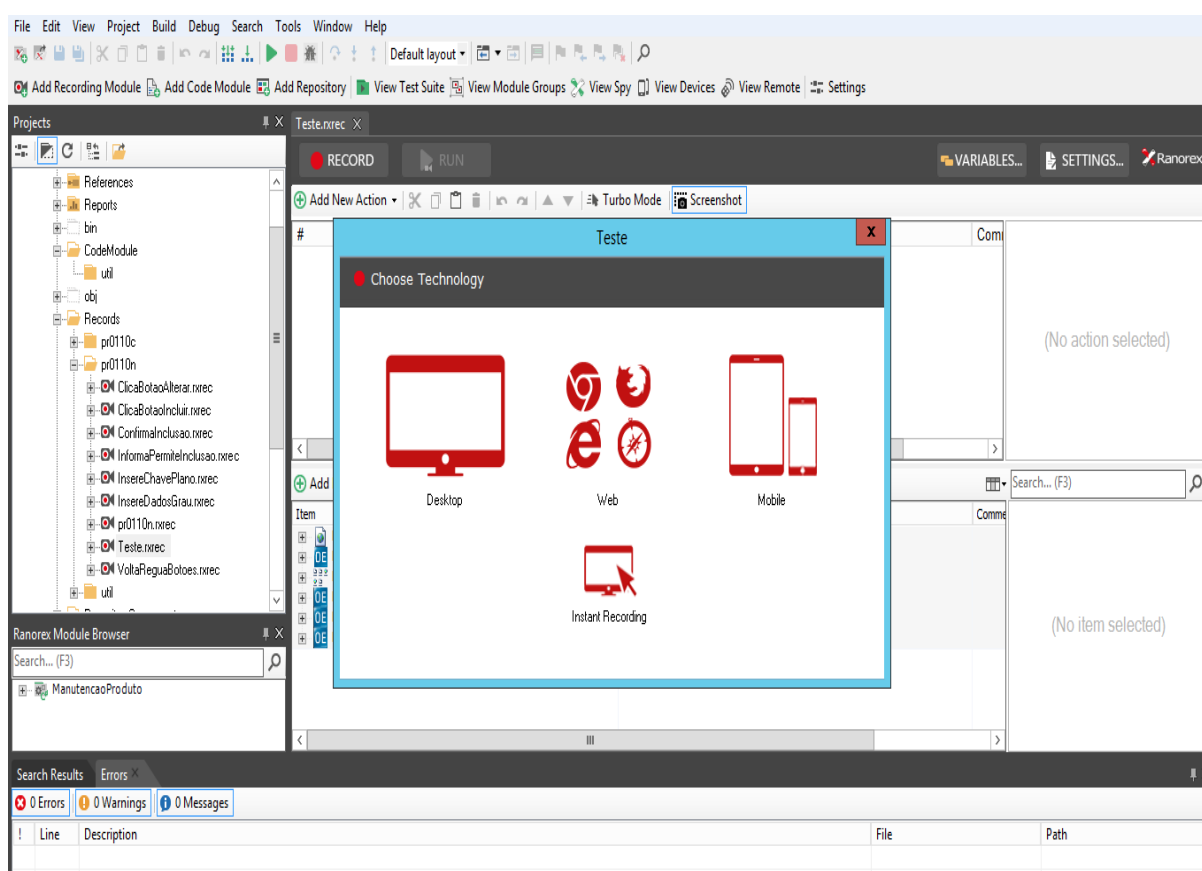
²¹ Scripts na Ranorex são as ações de tela do usuário que são convertidas para componentes da ferramenta, onde é possível manipula-los da forma que for necessária.

²² Data-driven é o conceito de entrada de dados de forma externa, para os scripts gerados na ferramenta. Dessa forma os dados de entrada não fixos nos scripts criados, mas informados através de arquivos que podem ser importados para a ferramenta, e utilizados nos casos de testes, permitindo um a grande variação de execuções.

Desktop, *Web*, *Mobile* ou *Instante Record* onde a ferramenta identifica automaticamente o tipo de aplicação.

Para os sistemas que foram automatizados, foi utilizada a opção de *Instante Record* devido ao fato de se acessar telas de plataformas distintas no momento da gravação. Nos casos onde não foi utilizada essa opção, os scripts não foram gerados corretamente sendo necessário recriá-los. A figura 39 é referente a seleção de plataforma para gravação.

Figura 39 - Seleção do Tipo de Plataforma para Gravação



Fonte: O autor, 2016

Iniciando a gravação, a ferramenta irá capturar todas as ações executadas em tela pelo usuário. Com isso, o caso de teste selecionado para automação, deve ser executado normalmente para que a ferramenta capture as ações e identifique componentes.

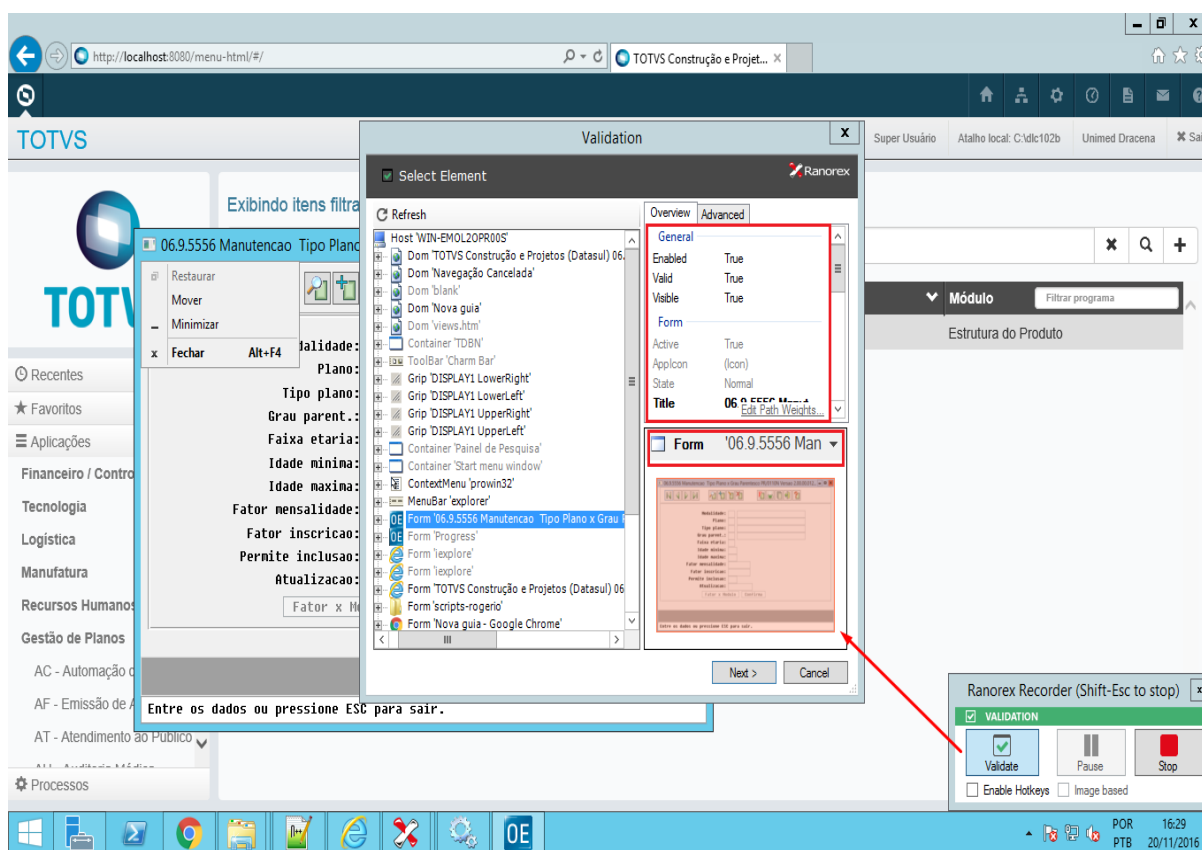
Para as telas desenvolvidas em Progress, é necessário que sejam clicados em todos os campos que solicitam entradas de dados, caso contrário podem ocorrer

erros no momento da ferramenta identificar os componentes, havendo a necessidade de recriar o script gerado.

Durante a gravação, é possível adicionar validações para as ações executadas, garantindo assim a integridade do caso de teste. Essas validações são criadas através do mecanismo de Validation disponibilizada pela Ranorex.

No momento de criação da validação pode-se definir quais critérios serão considerados para a verificação, como por exemplo, uma determina tela estar visível ou um campo que deve possuir um determinado valor. As validações também podem ser definidas manualmente na edição do script gerado, porém elas são mais trabalhosas de serem criadas. Na figura 40 é representado o mecanismo de *Validation* da Ranorex.

Figura 40 - Criação de Validation na Ranorex



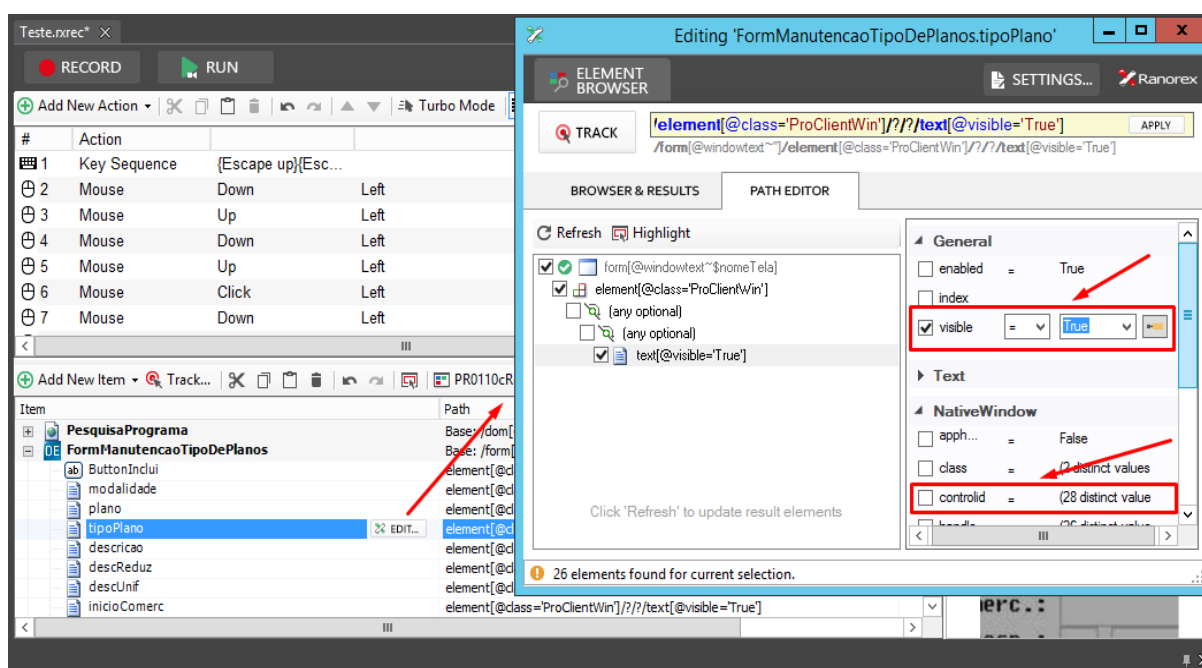
Fonte: O autor, 2016

Com o termino da gravação das ações de tela, são necessários ajustes no script gerado, assim como nos componentes mapeados de tela. As telas desenvolvidas em Progress não possuem identificadores para os componentes. Por isso, no momento da gravação, a ferramenta adiciona um identificador de sessão

para cada elemento, mas caso o script seja executado para a mesma tela em uma sessão diferente, o mesmo não irá funcionar, pois não conseguirá localizar o elemento baseado nesse identificador.

Considerando essa particularidade, todos os elementos de tela mapeados devem ser alterados para não considerar o Id de sessão, e sim se o componente está visível ou não. Essa alteração pode ser realizada de forma simples, selecionando o elemento no repositório e editando o mesmo. Na figura 41 é possível ver o processo para se editar o componente.

Figura 41 - Edição de Componente do Repositório



Fonte: O autor, 2016

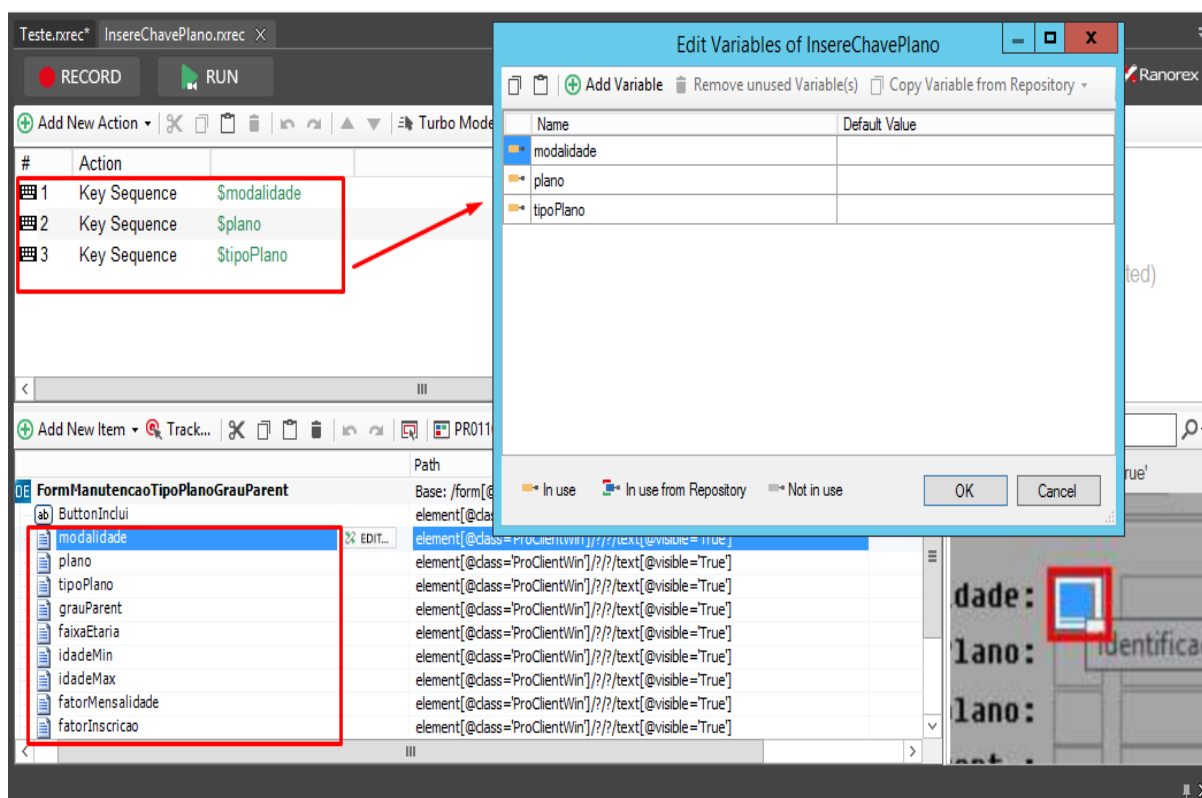
Além da remoção do identificador de sessão dos elementos, foram definidas algumas boas práticas para a identificação de componentes, e para controlar a entrada de dados.

Todos os campos e botões identificados devem ser renomeados com o mesmo nome do campo de tela do sistema, isso foi definido para que se possa criar os scripts manuais de forma mais fácil. A Ranorex cria os componentes com um nome sequencial, e caso os mesmos não sejam renomeados, é quase impossível localizá-los no repositório ou utilizá-los nos scripts.

Outra alteração importante que deve ser feita, é que para cada campo que tem entrada de dados, deve-se definir uma variável na ferramenta, para que

posteriormente seja possível utilizar data-driven para inserir os valores nos campos. Essas variáveis podem ou não possuir um valor padrão. Dessa forma os scripts podem ser executados com dados fixos, sem utilizar entrada de dados externos. É importante salientar que a criação dos arquivos de data-driven se baseiam nas várias definidas para cada um dos scripts gerados. Um exemplo de variáveis criadas na Ranorex se encontra na figura 42.

Figura 42 - Variáveis na Ranorex



Fonte: O autor, 2016

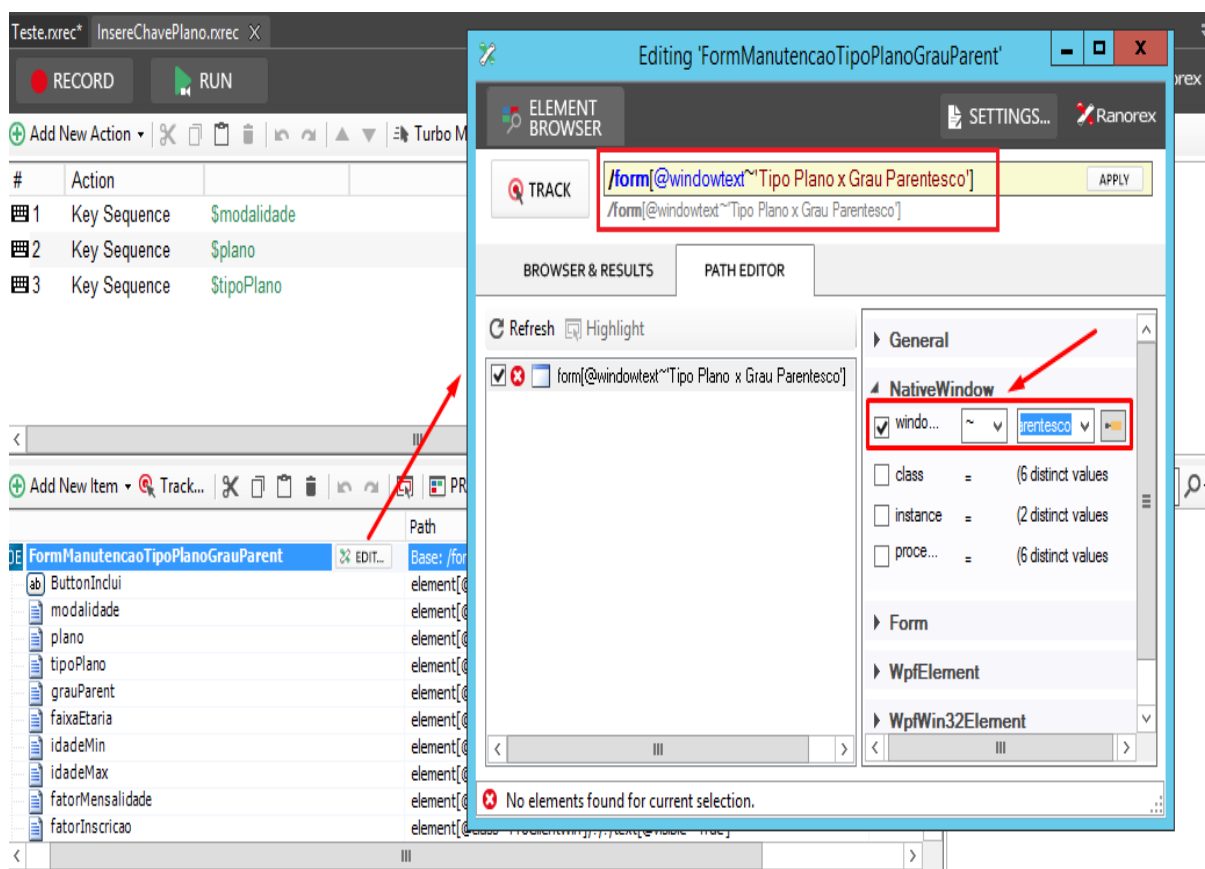
Para os elementos do tipo Form mapeados na gravação, foi realizado um ajuste com a finalidade de reduzir o número de manutenções nos scripts, e aumentar a reutilização de componentes.

No sistema da empresa, os formulários Progress, além do nome da tela também possuem o número da versão, existindo um Form para inclusão, alteração, exclusão entre outros. O nome da tela, juntamente com a versão é utilizado pelo Ranorex para identificar esses componentes, ou seja, tem que se executar uma gravação para uma das ações no cadastro, e recriar os scripts caso a versão seja alterada.

Tendo em vista que os componentes de tela são os mesmos para todas as ações do cadastro, não haveria a necessidade de recriar esses componentes. Por isso, para os elementos Form foi modificada a forma de identificação, passando a considerar apenas o nome da tela ignorando a ação, e a versão no sistema. Assim se criou um Form genérico, que pode ser utilizado para todas ações de um determinado cadastro.

No Ranorex, essa alteração foi feita de simples utilizando Regexp²³ para o campo windowText, e informando no nome da tela para identificação. Além da alteração da identificação, também foi modificado o nome do formulário para que o mesmo represente todo o cadastro. Dessa forma, cada tela possui apenas um elemento Form com todos os componentes. Essa alteração reduz o número de componentes, assim como otimiza a geração dos scripts. A alteração do componente Form é representado na figura 43.

Figura 43 - Alteração de Componente Form no Ranorex



Fonte: O autor, 2016

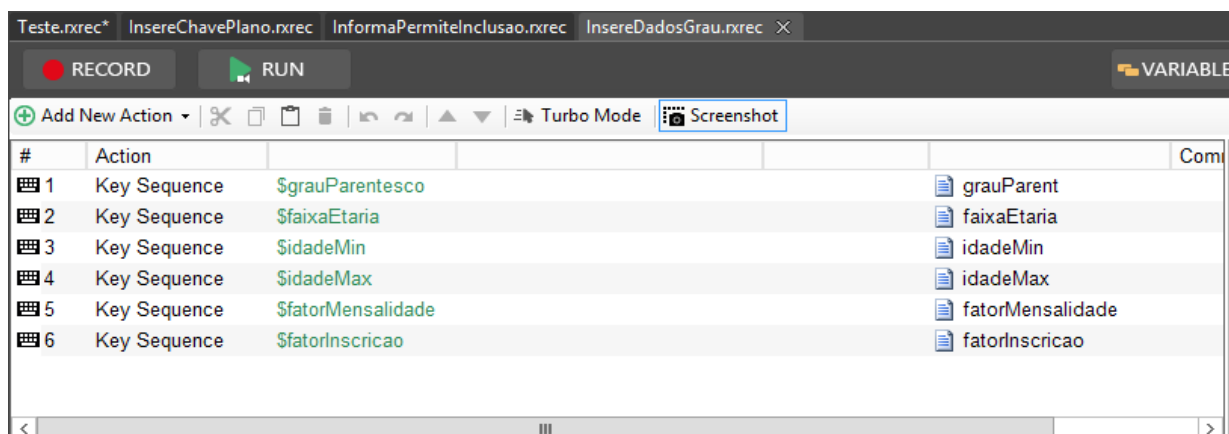
²³ O Ranorex fornece uma opção para identificar os componentes criados através de expressão regular, onde é informado um texto que se refere ao *label* do componente criado pela ferramenta.

Aplicando as alterações necessárias aos componentes criados durante a gravação, foi dado início aos ajustes no script gerado. Dessa forma foram definidos alguns passos que devem seguidos:

1. O primeiro passo, foi substituir as entradas de dados pelas variáveis criadas para a tela.
2. Removidos os *clicks* de mouse nos campos de entrada de dados para melhorar o tempo de execução dos *scripts*, sendo que são ações desnecessárias.
3. Quando existem *clicks* em botões, foi adicionado um novo evento movendo o *mouse* para o botão, e na sequência executando a ação de *click*.
4. Para as ações de abertura ou fechamento de telas, foram adicionados *delays* para que a ferramenta espere a tela fechar, ou carregar para executar as próximas ações.
5. Para telas Progress que possuem *subforms*, foi necessário substituir a ação de seleção dos valores dos *combo-box* de *click* de *mouse* para seleção pelas setas do teclado. Essa alteração foi necessária, pois a ferramenta não consegue identificar esses componentes para *click*, quando os mesmos não estão no Form principal.
6. Criadas validações adicionais nos casos que exista a necessidade para tal.

Com o *script* ajustado o mesmo é executado verificando se está funcionando corretamente. A figura 44 é um exemplo de um *script* criado e ajustado.

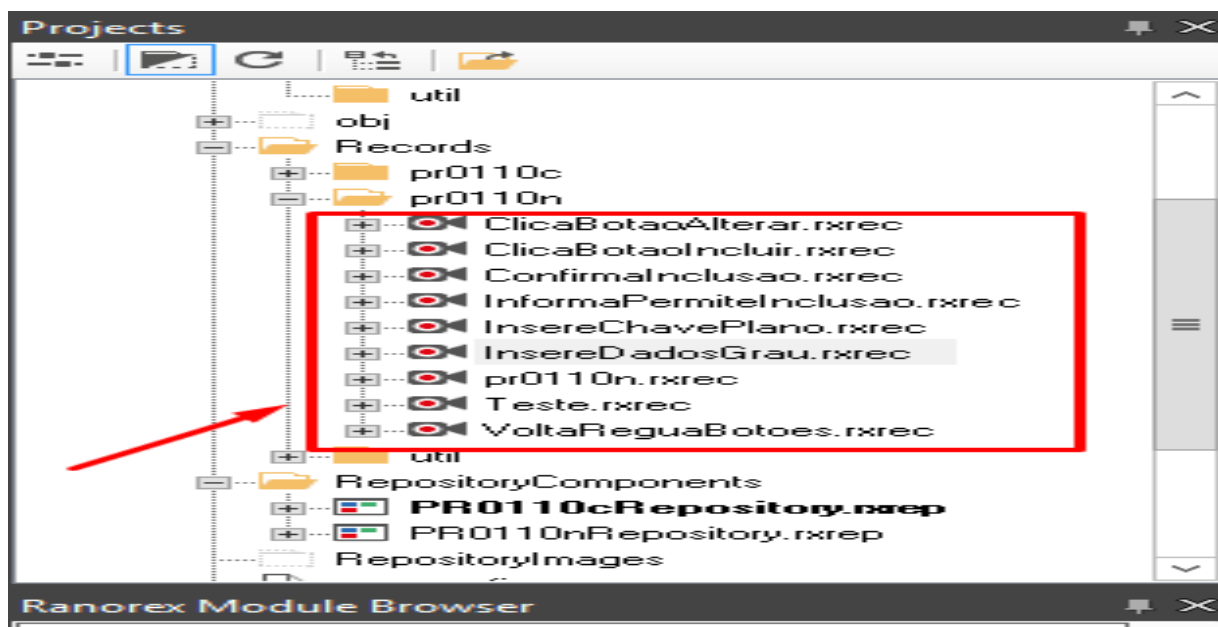
Figura 44 - Script de automação



Um outro ajuste importante realizado, foi subdividir os scripts em ações menores, exemplo disso são telas que para a inclusão, alteração e exclusão utilizam os dados mesmo dados de entrada. Dessa forma foi criado um script que é utilizado em todas ações, modificando apenas o data-driven utilizado por cada uma.

Para telas que possuem grid para inserção de dados, foram gerados scripts menores para facilitar a utilização da entrada de dados externos. Com essas mudanças é possível reutilizar uma gravação em vários casos de testes. Um exemplo de particionamento dos scripts está na figura 45.

Figura 45 - Scripts Particionados



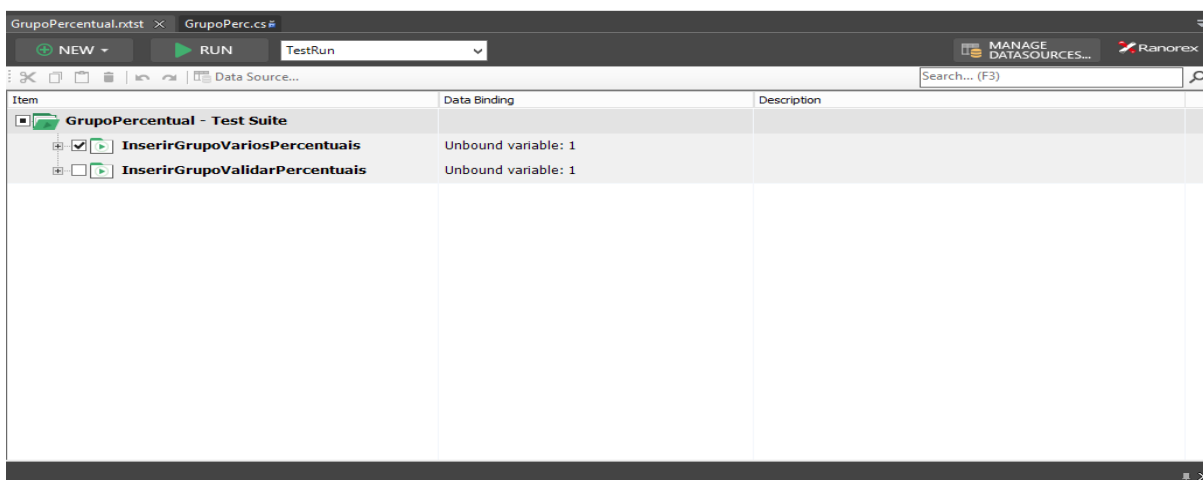
Fonte: O autor, 2016 1

6.2.2.3 CRIAÇÃO DOS CASOS DE TESTE

Somente a criação dos scripts automatizados, não é suficiente para a execução da automação. O Ranorex trabalha com o conceito de Tests Cases, que são agrupados na Test Suite da ferramenta. Cada projeto possui apenas uma Test Suite, sendo que a mesma pode conter quantos casos de testes forem necessários.

Com os scripts de automação criados e funcionando, foi iniciado o processo de criação dos casos de teste automatizados. Os casos de testes podem ser criados utilizando as gravações e scripts manuais gerados na ferramenta. A Suit Test da ferramenta pode ser visualizada na figura 46.

Figura 46 - Suit Test Ranorex

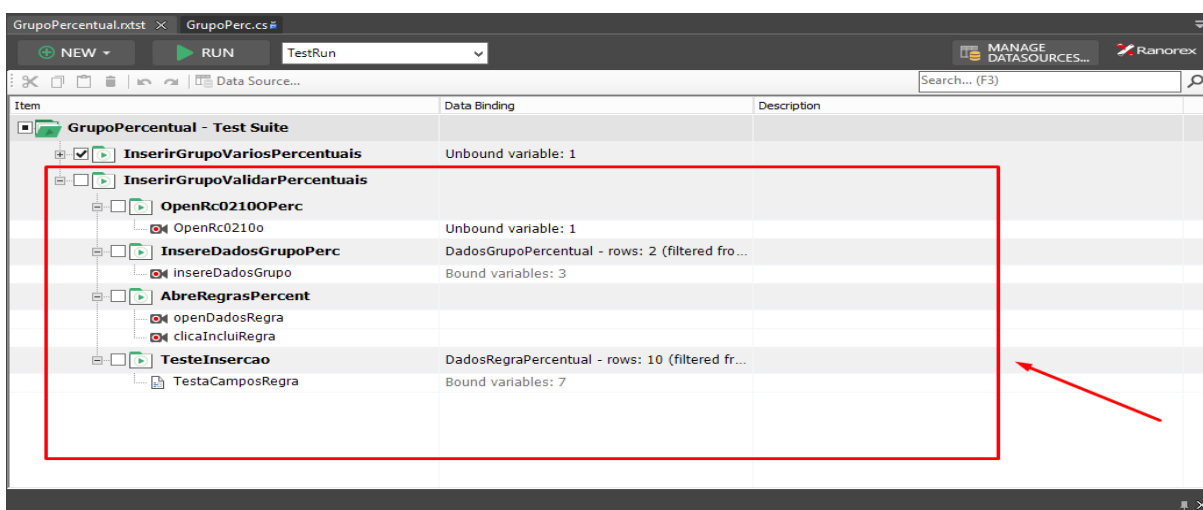


Fonte: O autor, 2016

O primeiro passo na criação dos casos de testes, foi unir os scrips criados para juntos executarem todas as ações necessária. Essa união foi feita, pois as gravações foram divididas em ações menores para maximizar o aproveitamento dos scrips, e reduzir o esforço para a criação dos casos de teste.

Importante salientar, que os componentes dos casos de testes, devem serem organizados de forma a suportar a entrada de dados externa. No Ranorex um caso de teste pode conter vários subcasos de teste, sendo possível definir entradas de dados específicas para cada um. Dessa a ferramenta possibilita criar casos de testes encadeados utilizando data-driven. A figura 47 representa os casos de teste no Ranorex

Figura 47 - Caso de Teste Ranorex

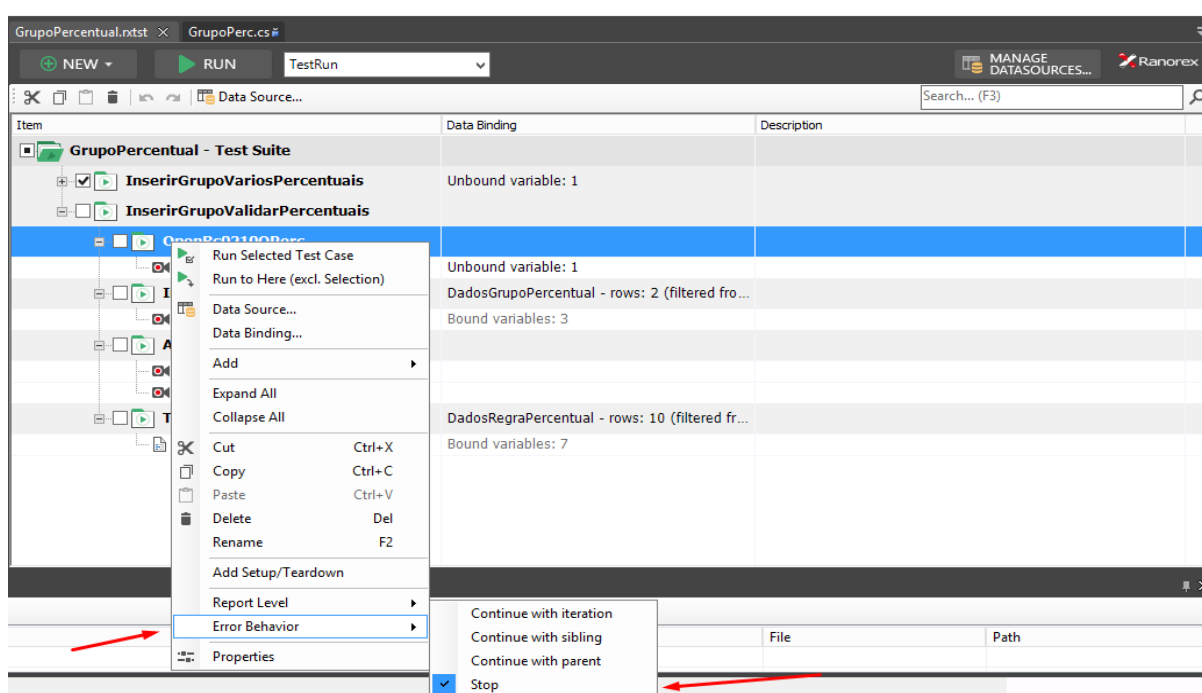


Fonte: O autor, 2016

No Ranorex, é possível definir níveis de erros para os casos de testes. Como um caso de teste pode conter vários sub-casos, é importante definir qual ação a ferramenta deve tomar mediante um erro de execução.

Para os casos de testes criados, foi utilizado apenas uma das opções disponibilizadas pela ferramenta, que é a de abortar a execução. Foi definida dessa forma, pois uma vez que não encontra um determinado componente, ou ocorram erros de inserção dados as demais ações também vão falhar.

Figura 48 – Nível de erro do Casos de Teste



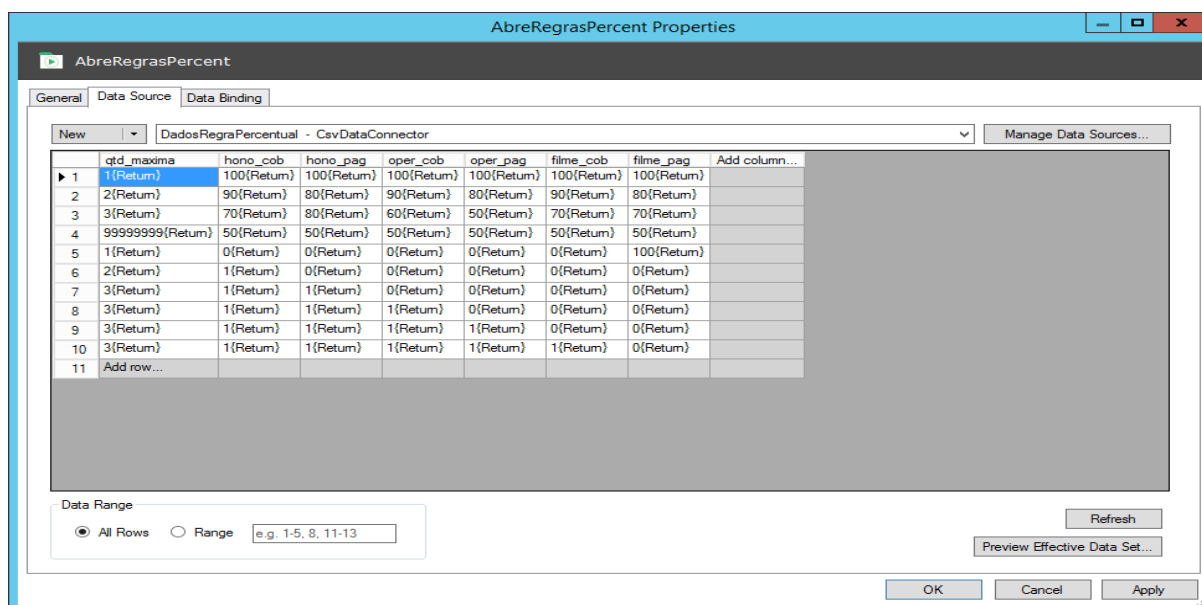
Fonte: O autor, 2016

Com os casos de testes estruturados, foram criados os data-driven, que são arquivos de entradas de dados externos. A criação desses arquivos, foi baseada nas variáveis definidas para cada uma das gravações. O Ranorex aceita três tipos de entradas de dados: csv, excell ou conexão com banco de dados.

Uma vez o arquivo criado, o mesmo deve ser importado para o DataSource do caso de teste, e realizar o mapeamento dos valores para as variáveis definidas nas gravações. O Ranorex possui uma opção de automapeamento, porém o nome dos campos no arquivo, devem ser os mesmos das variáveis existentes na ferramenta.

Para os testes criados, optou-se por utilizar o formato csv uma vez que a criação e manutenção são simples. Para cada caso de teste, é possível informar quais dados do arquivo devem ser utilizados na execução. Dessa forma, um mesmo data-driven pode ser reutilizado em vários casos de testes. A figura 49 representa um arquivo de data-driven importado no Ranorex.

Figura 49 - Importação de Data-Driven Ranorex



Fonte: O autor, 2016

A ferramenta, permite definir níveis para os logs de execução, porém para os testes de casos criados foi utilizada a configuração padrão, onde são listados todas as ações executadas, assim como a criação automática de prints de tela.

Para a Test Suite, é possível informar um diretório para a geração dos logs de acompanhamento. Esse diretório não foi informado, pois isso será feito no momento da integração do Ranorex com o Jenkins.

Após finalizar a criação e configuração dos casos de testes, os mesmos são testados, e analisada a execução. Uma vez finalizada com sucesso, é realizado o versionamento do projeto na ferramenta Team Foundation Server (TFS) utilizada pela empresa. Dessa forma é possível controlar as alterações, e criações dos casos de testes. Além dos projetos os data-drivers também são versionados para evitar perdas de dados.

6.3 INTEGRAÇÃO CONTINUA

A integração continua já faz parte do processo de desenvolvimento da empresa, sendo que é utilizado o Jenkins como ferramenta de automação para essas atividades. Por esse motivo optou-se também por utilizá-lo como ferramenta de integração continua para o ambiente de automação de testes.

O processo de integração continua foi dividida em duas etapas: Configuração da Ferramenta e Estrutura de Jobs.

6.3.1 CONFIGURAÇÃO DA FERRAMENTA

O Jenkins, permite que seja configurado um sistema de builds automatizados, assim como a instalação de diversos plug-ins para comunicação com outras ferramentas que auxiliam no processo de automação, como por exemplo plug-ins para o Ant, controle de versionamento, envios de e-mails entre outras.

A versão utilizada do Jenkins foi a versão 2.9, disponível para download no site do fornecedor²⁴. Após fazer o download da ferramenta, foi criada uma variável de ambiente no Windows com o nome de JENKINS_HOME, contendo o caminho do diretório raiz da aplicação.

Conforme orientação da Ranorex, para que seja possível integrar as duas ferramentas, o Jenkins não pode ser executado como serviço do Windows, mas sim via console. Como os Jobs também são utilizados para iniciar serviços de banco de dados, caso o Jenkins seja iniciado como serviço do Windows, após a execução desses Jobs o processo de banco de dados também é encerrado.

Para iniciar o Jenkins, foi criado um arquivo com extensão .bat contendo a linha de comando conforme a figura 50.

Figura 50 - Linha de Comando para Iniciar o Jenkins

```
java -jar -Dhudson.util.ProcessTree.disable=true %JENKINS_HOME%\jenkins.war --httpPort=8181 --webroot=%JENKINS_HOME%\war
```

Fonte: O autor, 2016

²⁴ <https://jenkins.io/>

Para que fosse possível a integração com algumas outras utilizadas no processo, foi necessária a instalação de alguns plug-ins disponibilizados para o Jenkins. Os plug-ins instalados foram:

- *Team Foundation Server Plug-in*²⁵: possibilita a comunicação com o TFS para atualização dos repositórios de código fonte dos produtos da empresa.
- *Email-ext plug-in*²⁶: permite configurar e-mails de notificação quando ocorre quebra nos Jobs executados pela ferramenta.
- *Ant Plug-in*: adiciona o Ant Build nas diretivas da ferramenta permitindo que build sejam disparados utilizando-se o Ant.
- *MSBuild Plugin*²⁷: esse plug-in possibilita que sejam criados Jobs agendados para acionar o Ranorex e executar os testes automatizados.
- *Environment Injector Plugin*²⁸: utilizado para criar variáveis de ambientes para os builds do Jenkins.

Além dessas configurações também foram criados usuários com permissões distintas para utilizar o Jenkins. Um perfil permite que as configurações da ferramenta sejam modificadas e criados novos Jobs²⁹, permitindo instalação e remoção de plug-ins entre outras.

O outro perfil criado permite apenas que sejam visualizados os Jobs executados, possibilitando que o usuário verifique os logs das execuções de forma detalhada, esse perfil é utilizado para acompanhamento da atualização e compilação de fontes assim como execução dos testes automatizados, as configurações de criação de perfil de usuário foram realizadas seguindo a documentação disponibilizada no site da ferramenta.

Na figura 51 é possível visualizar as características de instalação e configuração do Jenkins, assim como os plug-ins que foram instalados para dar suporte ao processo de integração contínua e automação.

²⁵ <https://wiki.jenkins-ci.org/display/JENKINS/Team+Foundation+Server+Plugin>

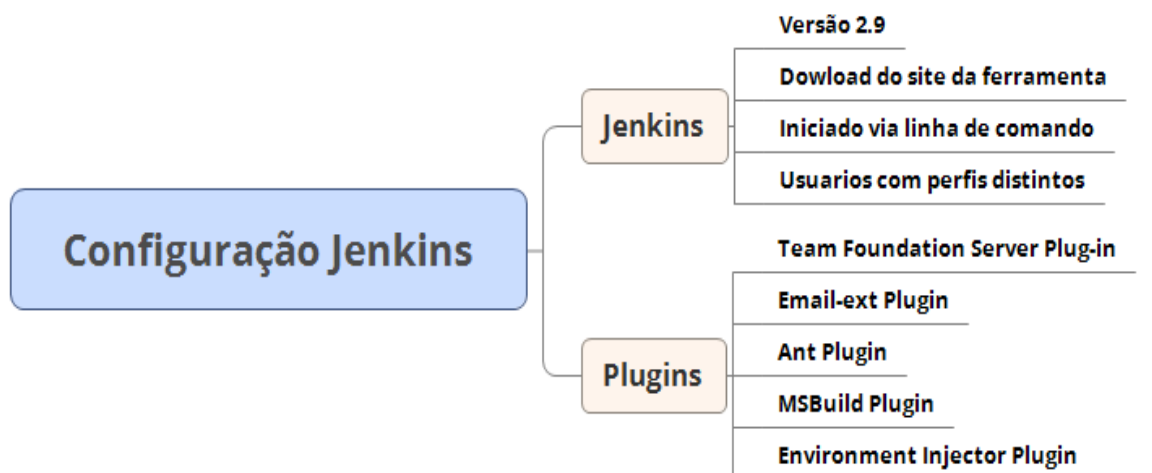
²⁶ <https://wiki.jenkins-ci.org/display/JENKINS/Email-ext+plugin>

²⁷ <https://wiki.jenkins-ci.org/display/JENKINS/MSBuild+Plugin>

²⁸ <https://wiki.jenkins-ci.org/display/JENKINS/EnvInject+Plugin>

²⁹ O Jobs no Jenkins se referem as tarefas que são criadas na ferramenta para executar uma determinada ação, seja compilar códigos fontes, atualizar diretórios, executar scripts entre outros.

Figura 51 - Configuração do Jenkins



Fonte: O autor, 2016

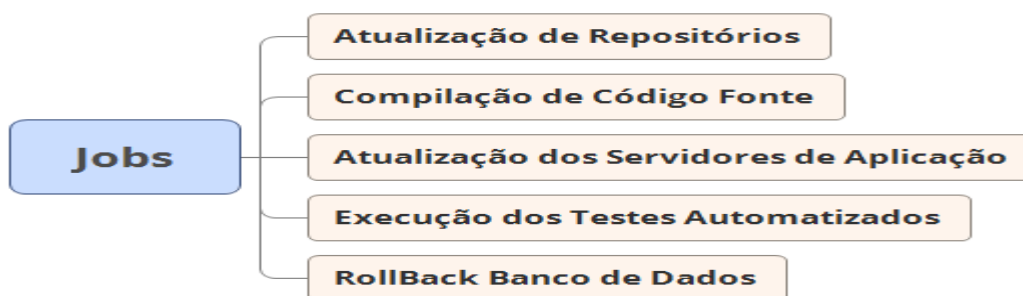
6.3.2 ESTRUTURA DE JOBS

No ambiente de testes o Jenkins assumiu um papel fundamental, pois se integra com todas as ferramentas necessárias para a automação, realizando a atualização dos diretórios de código fontes, compilação dos programas Progress e Java e atualização dos servidores de aplicação com base nas compilações realizadas. Possui Jobs para disparar a execução agendada dos scripts de automação, envios de e-mail quando a quebra nos builds de atualização, compilação ou testes.

Os Jobs criados no Jenkins para suprir as necessidades do processo de integração contínua, e de automação foram divididos em Jobs de atualização de repositórios, compilação de código fonte, atualização de servidores de aplicação, execução de testes automatizados e Rollback de banco de dados.

Desconsiderando os Jobs de automação foram criados 15 Jobs no total. A criação e configuração desses Jobs foi realizada conforme documentação disponibilizada pela ferramenta. A figura 52 ilustra a divisão dos Jobs no Jenkins.

Figura 52 - Divisão dos Jobs no Jenkins



Fonte: O autor, 2016

Os Jobs foram criados em cascata, ou seja, são executados de forma sequencial, caso ocorra erro em algum dos processos os demais Jobs não são executados, e e-mail é enviado informando a equipe que o processo falhou. No e-mail consta qual Job ocorreu falha, assim como o log detalhado da situação possibilitando que seja identificado o agente causador do problema, seja ele um commit que quebrou o build, erro de ambiente, falha nos servidores entre outros.

Como os produtos da empresa possuem dependências com sistemas de outros segmentos da corporação, foi necessária a criação de diversos Jobs para atualizar e compilar os códigos fontes dos sistemas a serem automatizados. No total foram criados seis Jobs para atualização de repositórios, sendo cinco desses para os programas desenvolvidos em Progress e um para o sistema desenvolvido exclusivamente em Java. Na figura 52 estão os Jobs de atualização de repositórios Progress.

Figura 53 - Jobs de Atualização de Fontes

		ATUALIZACAO SERVIDORES	BANCO DE DADOS	COMPILACAO	FONTES PROGRESS	TESTES AUTOMATIZADOS	Tudo	+
S	W	Nome ↓	Último sucesso	Última falha	Última duração			
		DDK-GP	5 dias 5 horas - #59	N/D	0,89 segundos			
		EMS2-GP	5 dias 5 horas - #66	N/D	18 segundos			
		EMS5-GP	5 dias 5 horas - #70	N/D	17 segundos			
		FLUIG-GP	5 dias 5 horas - #62	N/D	13 segundos			
		GPS-FONTES-PROG	5 dias 5 horas - #60	N/D	15 segundos			

Ícone: [S](#) [M](#) [L](#)

Legenda [RSS de tudo](#) [RSS das falhas](#) [RSS apenas para os últimos builds](#)

Fonte: O autor, 2016

A atualização dos fontes, se dá por meio de conexão com o um servidor TFS utilizado pela empresa para o controle de versão dos seus sistemas. Na execução das atualizações, é gerado um log contendo quais fontes foram atualizados, quem foi o usuário que realizou alteração assim como a descrição que foi inserida no TFS no momento do versionamento. Dessa forma caso ocorra erro na compilação é possível identificar o possível usuário que quebrou o build, permitindo que o erro seja solucionado de forma rápida evitando maiores impactos no produto. O Anexo J contém um exemplo do log de atualização gerado pela ferramenta. Na figura 54 é possível verificar como as informações de acompanhamento são geradas.

Figura 54 - Log de Atualização de Diretórios

```

...
26. TVAFA6
27. Alteração core do extratores
28. FO TWCLWJ - Ponto de CPC "VALIDA-VENC"
29. Correção compilação.
30. Projeto UX - Correções de busca na mov-insu e alterações no campo medico e enfermeiro auditor
31. TWD129 - Correções no processo de replicação de beneficiários.
32. TVZZWI - Ajuste na visualização dos tipos de atendimento do HMR no HAT
33. TWBTVF - Ajustes TISS/PTU Online 6.0
34. TWBTVF - Ajustes TISS/PTU Online 6.0
35. TVSFOQ - Ajuste negativa de guias no Auditoria Médica
36. TVXMWR - Considerar o grupo de auditoria da guia de atendimento.

Version 465837 by jv01\_ismael.reche:
MSAU-1320 - Ajustes TISS/PTU Online 6.0
📄 \$/GP/Fontes\_Doc/Sustentacao/V11/V11/progress/src/api/api-solicita-ptu60.p \(diff\)
📄 \$/GP/Fontes\_Doc/Sustentacao/V11/V11/progress/src/atp/at0110r.p \(diff\)

Version 465825 by jv01\_ronaldo.todeschini:
SAUGPS03-170 - Correção valor
📄 \$/GP/Fontes\_Doc/Sustentacao/V11/V11/progress/src/rcp/rc0212t.p \(diff\)
📄 \$/GP/Fontes\_Doc/Sustentacao/V11/V11/progress/src/rtp/rtpvalori.p \(diff\)

```

Fonte: O autor, 2016

Para a compilação foram criados dois Jobs um para a parte Progress e outro para a parte Java. Os códigos fontes Progress são compilados através do SyncProgress encaminhando via e-mail o log de acompanhamento gerado pela ferramenta caso ocorram erros na compilação.

A compilação do projeto desenvolvido exclusivamente em Java se dá por meio do plug-in do Ant. Esse Job é um pouco diferente dos Jobs Progress, nos quais o processo de atualização e compilação ocorrem de formas separadas. Para os projetos em Java, a atualização e compilação são executadas no mesmo processo, sendo que o log de acompanhamento é listado na ferramenta como ocorre com os demais processos. Na figura 25 é possível ver os Jobs de compilação criados.

Figura 55 - Jobs de Compilação de Fontes

[Adicionar descrição](#)

ATUALIZACAO SERVIDORES BANCO DE DADOS COMPILACAO FONTES PROGRESS TESTES AUTOMATIZADOS Tudo +						
S	W	Nome ↓	Último sucesso	Última falha	Última duração	
		SYNC-PROGRESS	18 dias - #64	5 dias 6 horas - #66	8,2 segundos	
		WAC-GP	5 dias 5 horas - #63	ND	3 minutos 3 segundos	

Ícone: [S](#) [M](#) [L](#)

[Legenda](#)
 [RSS de tudo](#)
 [RSS das falhas](#)
 [RSS apenas para os últimos builds](#)

Fonte: O autor, 2016

Após a execução dos processos de compilação são disparados os Jobs de atualização dos servidores de aplicação. Essas atualizações são realizadas por dois Jobs, um para atualizar o servidor que possui partes Progress e outro para o sistema desenvolvido exclusivamente em Java. Nesse processo são atualizados os dois servidores JBOSS, o sistema desenvolvido em Java é atualizado com os arquivos gerados pela compilação do Jenkins.

O servidor que dá suporte ao sistema em Progress é atualizado com arquivos gerados por outros servidores corporativos. Para atualizar o AppServer não houve a necessidade de criação de um Job uma vez que o processo do SyncProgress faz isso automaticamente. Na figura 56 são representados os Jobs de atualização de servidores.

Figura 56 - Jobs Atualização de Servidores

[Adicionar descrição](#)

ATUALIZACAO SERVIDORES BANCO DE DADOS COMPILACAO FONTES PROGRESS TESTES AUTOMATIZADOS Tudo +						
S	W	Nome ↓	Último sucesso	Última falha	Última duração	
		ATUA-JBOSS-GP	22 dias - #71	50 minutos - #77	4 minutos 3 segundos	
		ATUA-JBOSS-WAC	22 dias - #68	5 dias 5 horas - #73	21 segundos	

Ícone: [S](#) [M](#) [L](#)

[Legenda](#)
 [RSS de tudo](#)
 [RSS das falhas](#)
 [RSS apenas para os últimos builds](#)

Fonte: O autor, 2016

Com a atualização dos servidores de aplicação, são disparados os Jobs que chamam as rotinas de testes automatizados. Para isso os scripts foram versionados no TFS e criados Jobs no Jenkins que executam os mesmos. A criação dos scripts de testes e configuração no Jenkins é detalhada no capítulo 7.

Os últimos Jobs a serem executados na estrutura criada, são os Job de Rollback do Banco de dados. Para esse processo foram criados seis Jobs, sendo que são responsáveis por pararem os bancos de dados de cada um dos sistemas, sobrescreverem os dados do banco, e iniciarem todos os serviços das aplicações novamente. Com a execução desses Jobs o ciclo da integração continua se encerra. Na figura 57 é possível visualizar os Jobs criados para esse processo.

Figura 57 - Jobs de Banco de Dados

[Adicionar descrição](#)

ATUALIZACAO SERVIDORES		BANCO DE DADOS	COMPILACAO	FONTES PROGRESS	TESTES AUTOMATIZADOS	Tudo	+
S	W	Nome ↓	Último sucesso	Última falha	Última duração		
		ROLL-BACK-BD-GPS	20 dias - #50	N/D	15 minutos		
		ROLL-BACK-BD-WAC	20 dias - #2	N/D	0,57 segundos		
		START-BANCO-GPS	19 dias - #4	N/D	1 minuto 26 segundos		
		START-BANCO-WAC	20 dias - #4	27 dias - #1	2,1 segundos		
		STOP-BANCO-GPS	20 dias - #3	N/D	2 minutos 45 segundos		
		STOP-BANCO-WAC	20 dias - #4	N/D	9,2 segundos		

Ícone: [S](#) [M](#) [L](#)

Legenda [RSS de tudo](#) [RSS das falhas](#) [RSS apenas para os últimos builds](#)

Fonte: O autor, 2016

Inicialmente a estrutura de Jobs foi configurada para ser executada a cada três horas, porém como os processos são demorados, isso acabava gerando conflitos entre as execuções. Dessa forma optou-se para que a execução do processo de integração continua no ambiente de testes, fosse disparada diariamente às 19 horas. Para isso foi definido um Job principal que inicia toda a cadeia de execução. Esse Job foi intitulado de Plano de Teste e é detalhado no capítulo 7 deste trabalho.

7 INTEGRAÇÃO DE FERRAMENTAS

Mesmo criando os scripts de automação, ainda havia interferência manual para executá-los, e os resultados não eram armazenados para consulta futuras, se perdendo os históricos das execuções.

A ferramenta de automação selecionada, por si só não possui um mecanismo de execução automática. A Ranorex, deve ser integrada a outras ferramentas que possuam essa funcionalidade. Com o objetivo de maximizar a automação do processo e reduzir ainda mais a interferência manual, surgiu a necessidade de realizar a integração entre ferramentas no ambiente de protótipo.

Como o Jenkins já é utilizado como ferramenta de integração contínua, o mesmo também foi utilizado para executar os scripts de automação gerados pela Ranorex.

A persistência dos resultados das execuções pode ser feita pela própria ferramenta de automação, implementando um *ReportLogger*³⁰ conforme documentação do fornecedor. Porém, com o objetivo de criar um cenário de integração de ferramentas mais genérico, que possa reaproveitado com outras ferramentas de automação, decidiu-se criar uma aplicação para realizar essa persistência.

Considerando essas duas necessidades, o processo de integração de ferramentas foi dividido em três partes: Criação Ferramenta de Controle Bugs, Integração Jenkins X Ranorex e Avaliação da Integração das Ferramentas.

7.1 CRIAÇÃO FERRAMENTA DE CONTROLE DE BUGS

A ferramenta desenvolvida possui duas funcionalidades básicas: Persistir os dados dos planos e casos de testes e gerar relatório de acompanhamento baseado nos dados gravados.

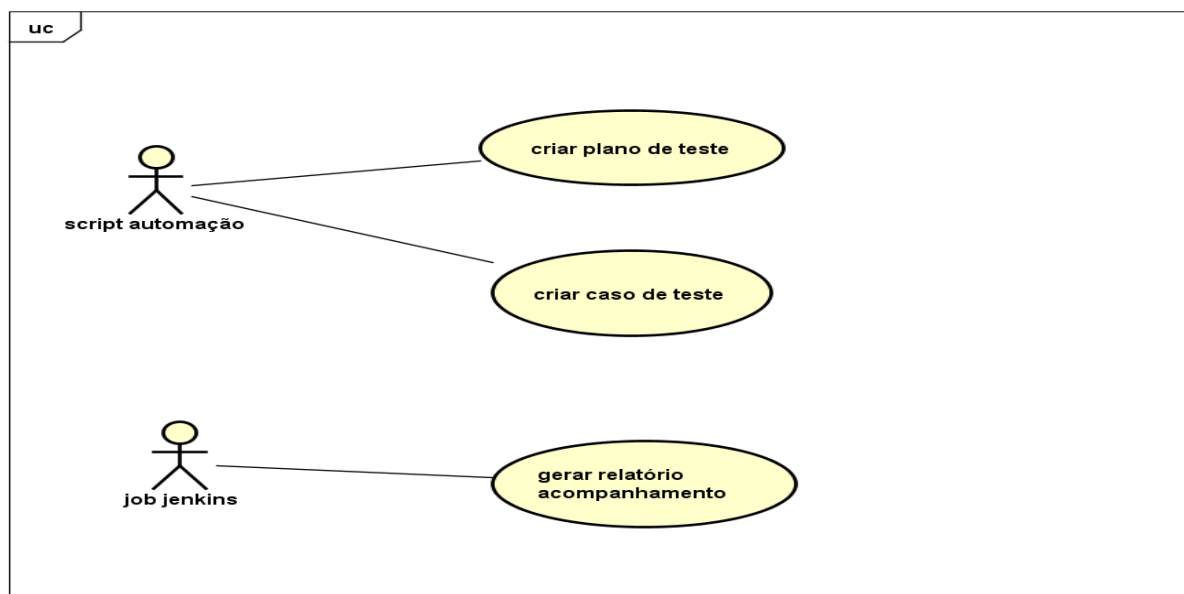
Considerando as funcionalidades básicas da aplicação, foram gerados três arquivos *jar*³¹ para as chamadas via console. Cada arquivo gerado é utilizado para executar umas das funcionalidades existentes, ou seja, existe um arquivo para

³⁰ O *ReportLogger* é uma interface disponibilizada pela Ranorex, contendo os métodos que devem ser implementados pela classe que irá realizar as persistências das execuções no banco de dados.

³¹ JAR (Java *Archive*) é um arquivo compactado contendo um conjunto de classes Java, um projeto java entre outros.

criação dos Planos de testes, um arquivo para criação de casos de teste e por fim um arquivo para geração de relatório. Esses arquivos criados são utilizados pelos scripts de integração para persistência e consulta das execuções dos scripts automatizados. A figura 58 representa os casos de testes da aplicação.

Figura 58 – Casos de Usos Ferramenta Controle de Bugs



Fonte: O autor, 2016

Para o desenvolvimento da ferramenta, foi utilizada a linguagem Java, juntamente com framework ORM³² Hibernate, optou-se por essas tecnologias, por serem as que se possuía maior conhecimento para programação.

A aplicação criada é *desktop*, não possuindo interface gráfica para o usuário, uma vez que o objetivo é auxiliar no processo de automação, e integração de ferramentas. Dessa forma o acesso as funcionalidades se dá via console.

Seguindo algumas boas práticas de programação, a aplicação foi desenvolvida em camadas utilizando o modelo MVC, e para persistência de dados foi utilizado o padrão de projeto DAO.

Como os sistemas da empresa, ainda utilizam o Java versão 7, para evitar incompatibilidades a ferramenta de controle de bugs foi desenvolvida nesta versão utilizando o Hibernate na versão 5.1.2.

³² ORM é o Mapeamento Objeto Relacional, onde as tabelas do banco de dados são representadas através de classes e cada registro do banco são representados como instancias dessas classes.

O banco de dados utilizado foi o PostgreSQL na versão 9.6.1, sendo que a instalação e configuração, foi realizada conforme orientação disponível no site³³ da ferramenta. A figura 59, representa algumas das características da ferramenta de controle bugs.

Figura 59 - Características da Ferramenta de Controle de Bugs



Fonte: O autor, 2016

Apesar de se ter utilizado o banco de dados PostgreSQL, a aplicação foi criada para que se possa utilizar outros bancos de dados, com isso, o arquivo de conexão com o banco, foi configurado de forma externa. Neste arquivo é informado o banco que será utilizado, endereço de conexão e dados de acesso.

O arquivo de conexão tem o nome de *connection.properties*, e o mesmo deve ser criado dentro da pasta *banco* no caminho especificado pela variável de ambiente AUTOMACAO_HOME. O detalhamento da estrutura de pastas e da variável de ambiente é feito na sessão 7.2. A figura 60 é um exemplo dos dados do arquivo de conexão.

Figura 60 - Arquivo de Conexão com o Banco de Dados

```

javax.persistence.jdbc.driver = org.postgresql.Driver
javax.persistence.jdbc.url = jdbc:postgresql://localhost:5432/automacao
javax.persistence.jdbc.user = postgres
javax.persistence.jdbc.password = automacao
hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
hibernate.hbm2ddl.auto = update
hibernate.show_sql = true
hibernate.format_sql = true
hibernate.connection.autocommit = false

```

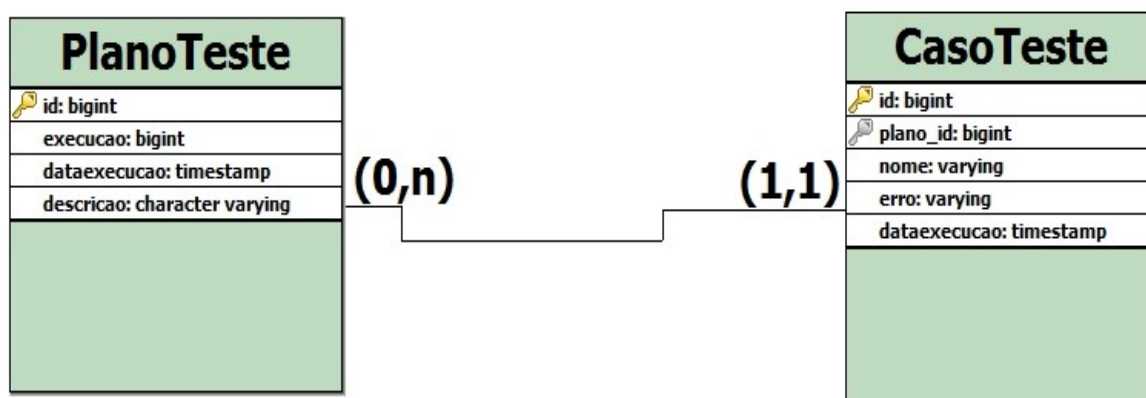
Fonte: O autor, 2016

³³ <https://www.postgresql.org/>

Para melhor estruturar os dados das execuções, foram criadas duas tabelas no banco de dados. Uma para as informações dos Planos de Teste, e outra para as informações dos Casos de Teste.

No desenvolvimento essas tabelas foram mapeadas para entidades da aplicação onde a relação entre elas ficou da seguinte forma: Um Plano de Teste pode conter uma ou vários casos de teste, porém um caso de teste pertence apenas a um Plano de teste. A persistência dos dados no banco é feita através do Hibernate. A figura 61 representa as tabelas criadas no banco de dados.

Figura 61 - Tabelas Ferramenta Controle de Bugs



Fonte: O autor, 2016

Todos os planos de testes criados pela ferramenta, possuem um Id do banco de dados que é gerado automaticamente, possuem um número e data de execução assim como a descrição do plano de testes. No ambiente de protótipo para o número da execução do plano, foi utilizado o número da execução do Job no Jenkins. No momento de criar o plano de testes, essa informação é passada via linha de comando para a ferramenta.

Assim como os planos os casos de testes, possuem um id gerado automaticamente, possuem também o nome do caso de teste, a data de execução, status de falha ou sucesso, e uma descrição de erro caso a execução falhe. Os casos de teste sempre são vinculados ao último plano de teste persistido, por isso, sempre deve ser realizada a criação do plano de teste antes da criação dos casos.

As informações do nome do caso de teste e do status da execução, são passados via linha de comando para ferramenta, caso o teste tenha falhado a

ferramenta faz a leitura do log de execução, que é gerado pelos scripts de integração e procura pela linha que contém a informação *[error]*.

Uma vez localizada essa linha, ela é gravada como sendo o identificador do erro do caso de teste. Nesse processo a ferramenta procura o arquivo com o nome de *exec-output.txt* na estrutura de pastas *tools/error* dentro diretório definido na variável de ambiente *AUTOMACAO_HOME*.

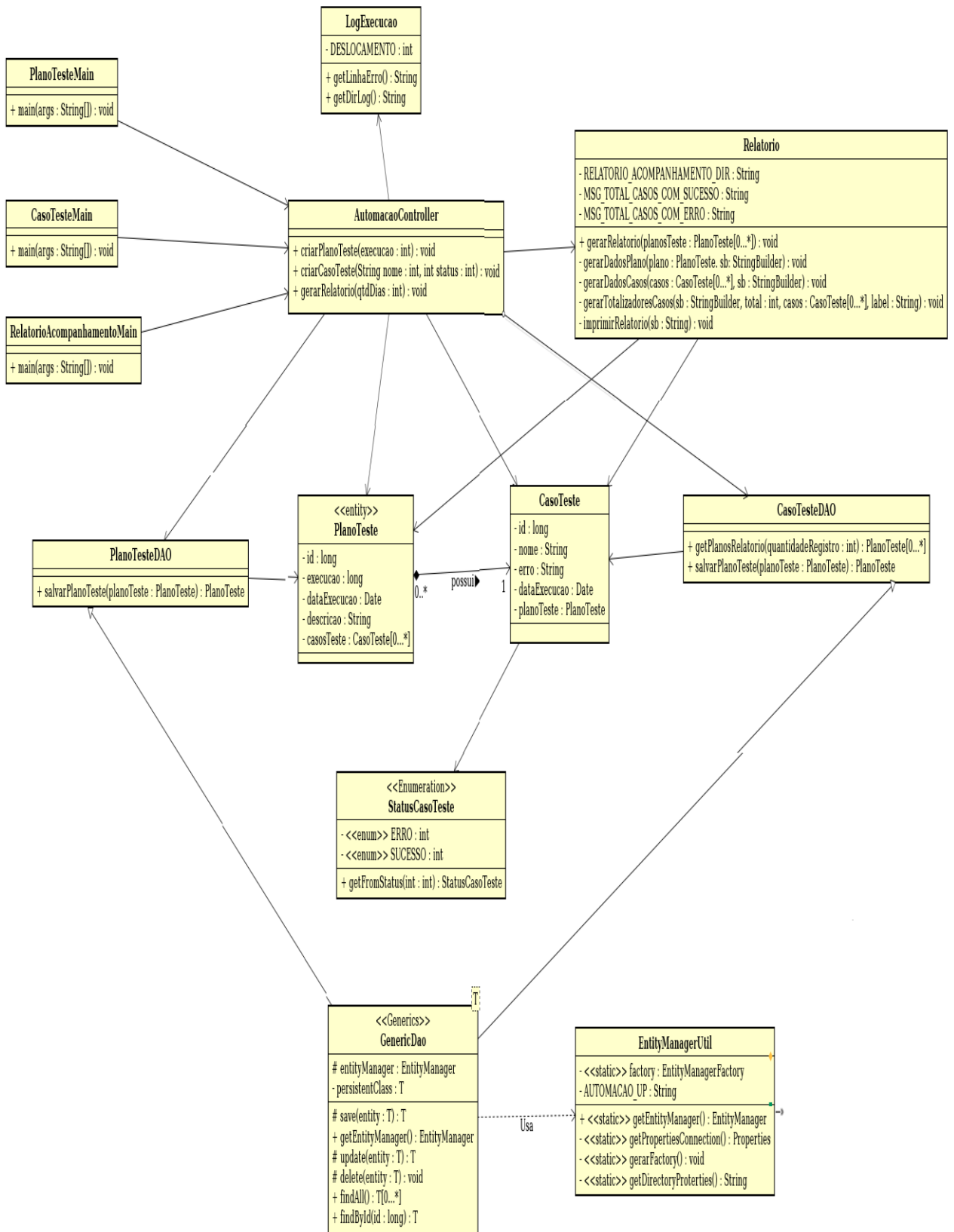
Além das funcionalidades de persistência, foi desenvolvida outra funcionalidade que gera um relatório listando os planos de testes e todos os casos de testes vinculado, separando os casos em casos de sucesso e erro. Para a geração do relatório é possível informar o total de dias aos quais se desejar ter as informações, sendo que padrão são os dados dos últimos sete dias.

Considerando a estrutura da aplicação e suas funcionalidades, assim como os padrões de desenvolvimento e projetos utilizados, as classes criadas foram agrupadas em pacotes.

- *Controller*: Contém as classes de controle da aplicação, ou seja, são as classes centralizadoras, delegando as requisições do console para as classes que devem processar essas requisições.
- *Dao*: Esse pacote contém as classes que interagem com o banco, são responsáveis pelas pesquisas e persistências dos dados.
- *Entity*: São as classes que representam as tabelas do banco de dados em objetos Java.
- *Enums*: São as enumerações utilizadas na aplicação.
- *Persistence*: Classe utilitária que controla a abertura de sessões no banco de dados.
- *Principal*: São as classes *main* da aplicação, que são chamadas via console.
- *Relatório*: Esse pacote agrupa as classes que interagem com os arquivos de log, e fazem a geração do relatório de acompanhamento.

Na figura 62 estão representas todas as classes criadas no desenvolvimento da aplicação.

Figura 62 - Diagrama de classes Ferramenta de Bugs



7.2 INTEGRAÇÃO JENKINS X RANOREX

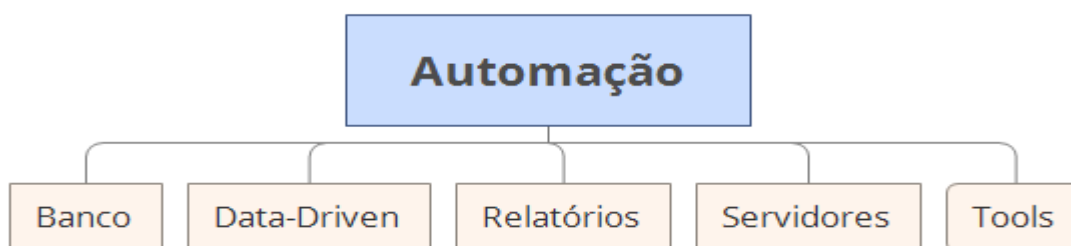
Com a criação dos scripts automatizados e configuração do Jenkins, teve-se início a integração entre ferramentas. Para que fosse possível essa integração foi necessária a criação de scripts integradores, e definição de algumas configurações de ambiente e estruturação de diretórios.

Com o objetivo de melhor estruturar os artefatos, utilizados por todas as ferramentas que compõem o processo de automação e integração contínua, foi definida uma árvore de diretórios padrões, sendo que a mesma é dividida conforme a funcionalidade dos arquivos ou scripts utilizados. No Jenkins os scripts definidos para os Jobs foram criados dentro dessa estrutura.

Após a criação destes diretórios, foi criada uma variável de ambiente com o nome de AUTOMACAO_HOME, contendo o caminho do diretório raiz desta estrutura.

Essa variável é utilizada por todos os scripts integradores e também pela ferramenta de controle de bugs. Sem a definição dessa variável no ambiente, os processos de integração não funcionam. A Figura 63 é uma representação dos diretórios criados para a integração de ferramentas.

Figura 63 - Diretórios de Integração



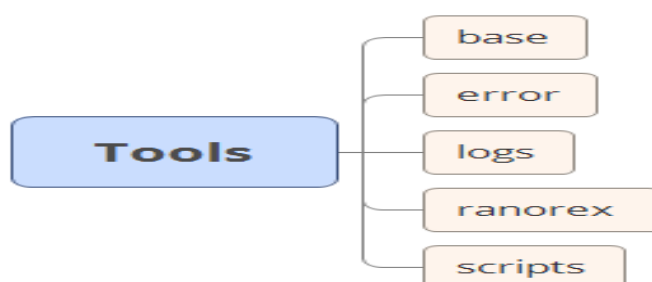
Fonte: O autor, 2016

Na estrutura de diretórios, os arquivos que são utilizados na integração entre as duas ferramentas foram criados dentro da pasta *tools*, sendo que a mesma foi subdividida nas seguintes pastas:

- Base: pasta que contém os scripts que são utilizados em vários processos

- Erro: diretório onde são gerados todos os arquivos de erros de execução dos processos de integração
- Logs: onde são criados os arquivos de logs de execução
- Ranorex: contém os projetos compilados gerados pela Ranorex
- Scripts: diretório onde são criados todos os scripts de integração dos testes

Figura 64 - Diretórios dos Scripts de integração



Fonte: O autor, 2016

Com a definição dos diretórios padrões, se deu início a criação dos scripts de integração. Inicialmente foi tentado integrar o Jenkins com Ranorex, seguindo a documentação disponibilizada pela Ranorex, porém a integração não funcionou adequadamente. O Jenkins não conseguiu executar os projetos compilados do Ranorex diretamente, então para ser possível integrar as ferramentas foram criados arquivos em Ant script, que são disparados pelo Jenkins para executar os scripts de automação.

Primeiro foi definido um script utilitário, contendo várias funções que são utilizadas durante o processo de integração, dessa forma se tem um maior reaproveitamento de código, e também reduz o número de arquivos alterados quando existem manutenções nas funções. Todos os arquivos que foram criados para a integração fazem uso deste arquivo utilitário. O mesmo encontra-se no *anexo L*.

Na sequência, foi criado no Jenkins um Job com o nome de Plano de Teste. Esse Job, é o responsável por disparar todos os demais Jobs de integração contínua e automação de testes.

Com a criação e agendamento de execução no Jenkins, foi definido o script de integração do Plano de Testes e o que ele deve executar. No script integrador, é

realizada uma chamada para a funcionalidade de criação de casos de teste da ferramenta de controle de bugs, passando via linha comando o número do Job do Jenkins. Além dessa chamada, esse script também cria uma pasta no diretório de logs, contendo o número da execução para que os logs das execuções dos casos de testes sejam salvos na estrutura de pastas correta. O conteúdo do script do Plano de testes encontra-se na figura 65.

Figura 65 - Script do Plano de Testes

```
<!-- Compilacao local -->

<project name="PlanoTeste" default="Main">

<taskdef resource="net/sf/antcontrib/antlib.xml"/>
<property environment="env"/>

<import file="${env.AUTOMACAO_HOME}/tools/base/AutomationBase.xml"/>

  <target name="Main">

    <property name="buildNumber" value="0"/>

    <antcall target="limpa_logs"/>

    <exec executable="cmd" output="${exec.db.output}" resultproperty="task.db.result">
      <arg value="/c"/>
      <arg value="java -jar ${env.AUTOMACAO_HOME}/banco/planoTeste.jar ${buildNumber}"/>
      <arg value="-p"/>
    </exec>

    <antcall target="verificaRetornoBanco"/>

    <propertyfile file="${plano.output}">
      <entry key="plano.number" value="${buildNumber}"/>
    </propertyfile>

    <mkdir dir="${env.AUTOMACAO_HOME}/tools/logs/${buildNumber}"/>

  </target>

</project>
```

Fonte: O autor, 2016

O passo seguinte, foi realizar a criação dos scripts dos casos de testes. São esses scripts que fazem as chamadas para os scripts criados no Ranorex. Todos os scripts de caso de teste, criam arquivos na pasta *error*. Com isso quando o script se inicia sempre são excluídos os arquivos desse diretório, garantindo que no fim do processo, os arquivos existentes são da execução corrente.

Após a remoção dos arquivos do diretório, é realizada a chamada via linha de comando para os projetos do Ranorex. Os projetos compilados, geram apenas um executável, contendo toda a Test Suite. Por isso no momento da chamada deve-se especificar qual o caso de teste que se deseja executar. Além de informar o caso também é especificado o diretório onde o relatório de execução deve ser gerado. Na sua documentação a Ranorex lista quais são os comandos reconhecidos pela ferramenta via console.

No final da chamada para o Ranorex foi incluída uma validação para verificar se o teste falhou, essa verificação feita baseada nos retornos definidos pela Ranorex, se o caso de teste falhou é chamada a funcionalidade de persistência passando o caso de teste como falha e também sinalizando o Jenkins de o Job falhou, no contrário o caso de teste de é salvo como sucesso e o Job é executado normalmente. Na figura 66 está o conteúdo de um desses scripts de integração.

Figura 66 - Script de Integração para Caso de Teste

```
<project name="LOGIN-TOTVS" default="Main">
<taskdef resource="net/af/antcontrib/antlib.xml"/>
<property environment="env"/>
<import file="${env.AUTOMACAO_HOME}/tools/base/AutomationBase.xml"/>
<!-- importa propriedades do plano de teste -->
<loadproperties srcFile="${plano.output}"/>
<!-- Define o nome do caso de teste que será gravado no banco -->
<property name="nomeCasoTeste" value="LOGIN-TOTVS"/>
<property name="diretorio.log.caso" value="${env.AUTOMACAO_HOME}\tools\logs\${plano.number}\${nomeCasoTeste}"/>
</project>
<target name="Main">
<antcall target="limpa_logs"/>
<antcall target="criaDiretorioLog"/>
<antcall target="Login"/>
<antcall target="gravaCasoTeste">
<param name="statusCasoTeste" value="0"/>
</antcall>
</target>
<target name="Login">
<exec executable="cmd" output="${exec.output}" resultproperty="task.result">
<arg value="/c"/>
<arg value="${env.AUTOMACAO_HOME}\tools\ranorex\Autenticacao\Autenticacao\bin\Debug\Autenticacao.exe /tc:LoginTotvs /zr /zrf:${diretorio.log.caso}report.rxzlog"/>
<arg value="-p"/>
</exec>
<antcall target="limpaLogsExecRanorex" />
<antcall target="verificaRetornoComFalha"/>
</target>
```

Cada Scripts de Integração criado, deve executar apenas um caso de teste definido no Ranorex, não é indicado que sejam agrupados vários casos de testes em um mesmo script, pois isso dificulta a identificação dos erros. Na estrutura criada, para cada caso de teste, deve-se ter um Jobs no Jenkins para se ter um controle melhor sobre as quebras de execução. Assim como os demais Jobs configurados no Jenkins, os de casos de teste também foram configurados para disparar e-mail, informando a quebra na execução.

7.3 AVALIAÇÃO DA INTEGRAÇÃO DE FERRAMENTAS

Com a criação das estruturas necessárias para a integração das ferramentas, foi possível realizar algumas avaliações sobre todo o processo de integração contínua, automação e integração entre as ferramentas.

A primeira avaliação foi em cima do Jenkins, onde foi analisado o desempenho da ferramenta no ambiente de protótipo. Durante alguns dias foram acompanhadas as execuções dos Jobs, analisando o tempo de execução, dependências entre Jobs, relatórios de acompanhamentos gerados e envio de e-mails.

Nos Testes, o Jenkins se mostrou estável, apesar que quando foram disparados vários Jobs em paralelo a ferramenta apresentou alguns erros, porém, isso não impactou na execução dos Jobs, uma vez que os mesmos foram criados para serem executados em cascata, evitando esse tipo de situação.

O Jenkins também se mostrou muito versátil, permitindo a integração com várias ferramentas, além de possuir uma vasta quantidade de plug-ins, que auxiliaram muito no processo de integração contínua, e integração com o Ranorex.

O envio de e-mails que ferramenta possui, se mostrou muito útil para o acompanhamento de todas as execuções, pois o feedback é instantâneo. Outro ponto interessante de ser mencionado são os relatórios das execuções geradas pela ferramenta, que permitem uma verificação rápida dos erros que causaram a quebra das execuções.

Para avaliar o Ranorex foram selecionados alguns casos de testes, seguindo o processo de automação definido. Foram selecionados três casos de testes documentados Kanoah, verificando se os mesmos podiam ser automatizados.

Tendo os casos de testes selecionados, foram definidos os dados que deviam ser utilizados nos testes, e criados os data-drivens iniciais, que posteriormente foram ajustados e importados para a ferramenta.

Na sequência, foram criados os scripts e casos de testes no Ranorex. Inicialmente se teve muita dificuldade para ajustar esses scripts para o ambiente que foi automatizado, principalmente para a parte Progress, onde a ferramenta teve alguns problemas para identificar elementos de tela, pois os mesmos não possuíam identificador.

No final do processo de criação dos scripts automatizados, os três casos de testes iniciais se desdobraram em 10 casos de testes no Ranorex. Com os casos criados, foram realizadas algumas análises sobre a ferramenta e os scripts de automação.

Não foi possível comparar as execuções manuais com os testes automatizados, pois seria necessário muito mais tempo de acompanhamento e execução de testes, para se criar uma base de dados que possibilitasse um comparativo dessa natureza. Dessa forma, foram analisados o desempenho das execuções, usabilidade da ferramenta, reaproveitamento de código e flexibilidade na criação dos casos de testes.

Nos testes executados, o Ranorex teve um desempenho muito bom. Os scripts podem ser executados de forma rápida e fácil, e não foram encontrados problemas de desempenho. A ferramenta possui uma interface intuitiva e de fácil aprendizagem.

A criação dos scripts é feita de forma rápida, permitindo criar elementos que podem ser reaproveitados em vários casos de testes, reduzindo o retrabalho na automação, além de ter vários recursos de depuração de código e relatórios.

Os data-drivens são simples, e não demandam muito tempo na criação, e permitem uma gama de variações para as execuções. Também foram analisadas se as validações criadas na ferramenta eram respeitadas nas execuções, as quais ocorreram sem maiores problemas.

Após a criação dos scripts foi analisada a integração entre as ferramentas, essa avaliação consistiu basicamente em criar os scripts de integração para cada um dos casos de testes do Ranorex.

Com a estrutura de integração definida, esse processo não demandou muito tempo e nem grande conhecimento técnico, uma vez que os scripts são simples e possuem muitas funções prontas para uso.

Nos testes, realizados, não ocorreram maiores erros nas execuções. O Ranorex e o Jenkins se integraram sem problemas. Os logs de acompanhamento foram gerados corretamente, e o envio de e-mails ocorreu sempre que algum caso de teste falhou. A ferramenta de controle de bugs se integrou ao processo sem problemas e armazenou todas as execuções corretamente.

8 CONCLUSÃO

Através deste trabalho, consolidaram-se os conceitos estudados através do contato direto com as atividades pertinentes ao processo de testes, automatizando parte dessas tarefas.

Primeiramente, aprendeu-se que o processo de garantia de qualidade de software é complexo e extremamente trabalhoso, principalmente em sistemas de grande porte, onde as mudanças são constantes, e a execução de testes deve ser realizada de forma contínua.

Para que se possa obter o mínimo de qualidade, é necessário que exista um processo de testes bem estruturado para orientar as atividades de testes, definindo recursos e ferramentas para cada etapa.

Um processo de testes pode ser dividido em diversas etapas, sendo que cada uma tem como objetivo validar um aspecto diferente da qualidade do software. O processo de testes pode ser dividido em teste de unidade, teste de integração e testes sistêmicos, sendo que são etapas sequenciais, e cada uma pode utilizar abordagens de caixa branca ou caixa preta.

Para se definir um processo de automação, primeiro pesquisou-se sobre automação de teste, e foi verificado que a mesma não surgiu para solucionar todos os problemas e nem para substituir os testes manuais, mas sim para reduzir o esforço necessário para a execução de determinados tipos de testes, possibilitando aumentar o número de casos e reduzir o tempo de execução.

Durante o mapeamento das atividades da equipe de testes para propor o processo de automação, foi possível verificar a complexidade que são as atividades de testes manuais, sendo que é inviável se aplicar todos os testes necessários, em um curto espaço de tempo para um sistema de grande porte. Outro ponto observado, foi a importância de se ter ambientes de desenvolvimento e testes separados, uma vez que alterações realizadas pelo desenvolvimento impactam diretamente nos testes.

Através deste mapeamento, foram sugeridas alterações no processo de testes atual da empresa para contemplar as atividades de automação, criando-se novas tarefas dentro do processo existente.

Para se escolher ferramentas de automação deve-se ter critérios bem definidos de avaliação, considerando disponibilidade financeira, o ambiente a ser

automatizado, processo de testes existente e recursos que irão fazer uso da ferramenta. No processo de avaliação teve-se dificuldades em encontrar ferramentas *open source* para automação de sistemas *desktop*, sendo que a maioria das ferramentas é proprietária, necessitando um investimento financeiro considerável.

Durante a configuração do ambiente de protótipo, foi possível observar que para se utilizar automação de testes, deve-se ter uma boa infraestrutura para dar suporte a todos os processos e ferramentas, que estão envolvidas nessas atividades, assim como o ambiente de testes deve ser isolado, ou a automação não trará os resultados esperados.

Na etapa de criação dos scripts de automação, evidenciou-se que automatizar é uma tarefa trabalhosa que demanda tempo. Foram encontradas muitas dificuldades na criação de scripts funcionais para o ambiente *desktop*, uma vez que o sistema que foi automatizado é antigo. Também foi possível evidenciar a importância de um processo de integração contínua, sendo que o mesmo centralizou quase todas as atividades necessárias, para manter o ambiente de protótipo funcional.

Na integração de ferramentas, aprendeu-se que esse é um aspecto importante, a ser considerado no momento da seleção de ferramentas que irão compor o processo de automação. Foram encontradas algumas dificuldades para fazer a integração entre o Jenkins e o Ranorex, uma vez que integração descrita na documentação da ferramenta de automação não funcionou para o ambiente de protótipo. Dessa forma foi necessária se encontrar uma saída alternativa, para esta situação, a qual demandou muito esforço.

Depois da criação do ambiente de protótipo e integração de ferramentas e também da criação de scripts de automação, foi possível comprovar que a automação é um processo caro na implantação, que demanda investimento de tempo, dinheiro, treinamento e infraestrutura, sendo deve ser bem planejado considerando todos os riscos envolvidos. Apesar desses fatores a automação traz muito ganhos em questão de qualidade, redução de tempo e também dinheiro, uma vez reduz o retrabalho e a manutenção no sistema, permitindo a construção de produto de qualidade em curto período de tempo.

Além da escolha de uma ferramenta que atenda às necessidades do ambiente a ser automatizado. É crucial que exista um processo de testes bem

estruturado para orientar as atividades de automação, para que dessa forma seja possível reduzir os riscos e maximizar o retorno do investimento.

Considerando a metodologia apresentada para o desenvolvimento e os objetivos propostos para esse trabalho, conclui-se que os mesmos foram alcançados com sucesso. O processo de automação proposto, pôde ser avaliado de forma prática, onde o mesmo foi utilizado juntamente com a ferramenta de automação, orientando todas as atividades de testes executadas.

Além disso, foi possível evidenciar a importância de um processo de processo de integração contínua, através da configuração do Jenkins no ambiente de protótipo, onde o mesmo centralizou todas as atividades para manter o ambiente de testes funcional.

Através dos scripts de automação foi possível comprovar alguns dos benefícios que a automação de testes pode trazer, como a execução rápida e fácil de casos de testes, e também um grande número de variações através de data-driven.

A integração entre ferramentas se mostrou de extrema importância uma vez que automatizou a execução dos scripts, reduzindo a interferência manual e gerando feedbacks instantâneos sobre as execuções.

Durante a realização deste trabalho foram identificados alguns pontos que podem ser objetos de estudo para trabalhos futuros:

- Desenvolver uma análise baseada nos dados das execuções persistidos em banco com a finalidade de melhorar o processo de testes.
- Evoluir a ferramenta de persistência para gerar relatórios de acompanhamentos mais detalhados e com uma análise mais profunda dos dados, por exemplo, criando um sistema de Inteligência de negócio para analisar os resultados e tomar ações com base neste dados.
- Aprimorar os sistemas de relatório criados e integrar os resultados das execuções com o plug-in Kanoah do Jira.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR ISO/IEC 14598-4: Engenharia de Software - Avaliação de Produto. Rio de Janeiro: Abnt, 2003.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR ISO/IEC 9126-1: Engenharia de Software - Qualidade de Produto. Rio de Janeiro: Abnt, 2003.

BARTIÉ, Alexandre. Garantia da Qualidade de Software: adquirindo maturidade organizacional. 1. ed. Rio de Janeiro: Elsevier, 2002.

BERNARDO, Paulo Cheque; KON, Fabio. A importância dos testes automatizados. Engenharia de Software Magazine, v. 1, n. 3, p. 54-57, 2008.

DUBEY, Neha; SHIWANI, Mrs Savita. Studying and Comparing Automated Testing Tools; Ranorex and TestComplete. IJECS, v. 3, n. 5, p. 5916-23, 2014.

F DA SILVA, Monique; G MORENO, Autran. AUTOMAÇÃO EM TESTES ÁGEIS. Revista de Sistemas e Computação-RSC, v. 1, n. 2, 2012.

FOWLER, Martin; FOEMMEL, Matthew. Continuous integration, 2006. URL <http://www.martinfowler.com/articles/continuousIntegration.html>, 2012.

HOLMES, Antawan; KELLOGG, Marc. Automating functional tests using selenium. In: Agile Conference, 2006. IEEE, 2006. p. 6 pp.-275.

JAIN, Abha; JAIN, Manish; DHANKAR, Sunil. A Comparison of RANOREX and QTP Automated Testing Tools and their impact on Software Testing. IJEMS, v. 1, n. 1, p. 8-12, 2014.

KAUR, Manjit; KUMARI, Raj. Comparative study of automated testing tools: Testcomplete and quicktest pro. International Journal of Computer Applications, v. 24, n. 1, p. 1-7, 2011.

KAUR, Harpreet; GUPTA, Dr Gagan. Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete. Harpreet kaur et al Int. Journal of Engineering Research and Applications ISSN, p. 2248-9622, 2013.

LIMA, T.; DANTAS, Ayla; VASCONCELOS, Livia MR. Usando o SilkTest para automatizar testes: um Relato de Experiência. In: 6TH BRAZILIAN WORKSHOP ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING. 2012.

MOLINARI, Leonardo. Testes de Software: Produzindo Sistemas melhores e mais confiáveis. 4. ed. São Paulo: Erica, 2008.

MOLINARI, Leonardo. Inovação e automação de testes de software. 1. Ed. São Paulo: Erica, 2010.

PFLEEGER, Shari L. Engenharia de Software: Teoria e Prática. 2. ed. São Paulo: Pearson Prentice Hall Brasil, 2004.

PRESSMAN, Roger S. Engenharia de software: uma abordagem profissional. 7ª Edição. Ed: McGraw Hill, 2011.

ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al., "Qualidade de software – Teoria e prática", Prentice Hall, São Paulo, 2001.

RIOS, Emerson, et al. Base de conhecimento em teste de software. 3.ed. São Paulo: Martins Fontes, 2012.

XIAOCHUN, Zhu, et al. "A test automation solution on GUI functional test." *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. IEEE, 2008.

Anexo A – Questionário Aplicado no Setor da Empresa

Editar este formulário

Processo de Testes Setor de Qualidade

*Obrigatório

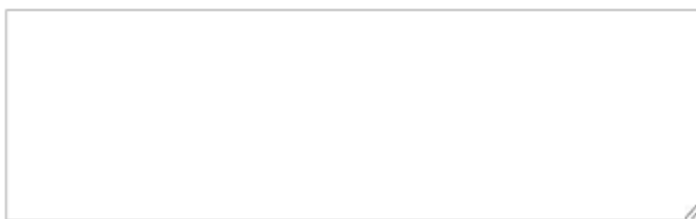
Você possui conhecimentos em programação? Caso sim, em quais linguagens?

Qual seu nível de conhecimento na regra de negócio do sistema ?

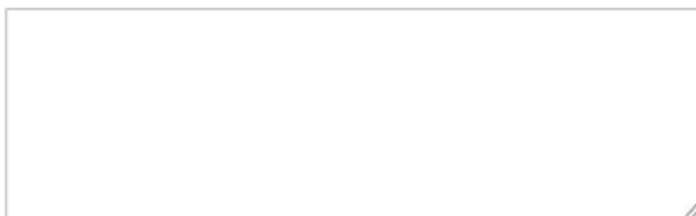
Quais as atividades você desempenha no processo de teste? Descreva em itens *

Descreva em itens as etapas de testes que são realizadas no setor (Planejamento, criação de cenários, etc...)

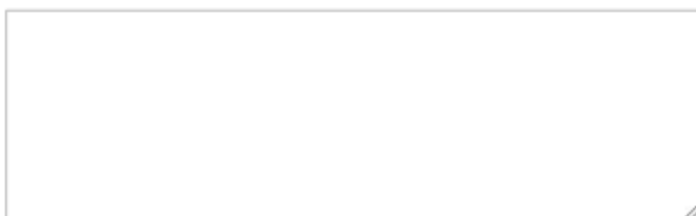
Como o plano de testes é definido ?



Como os cenários de testes são definidos ?



Como são definidos os casos de teste ?



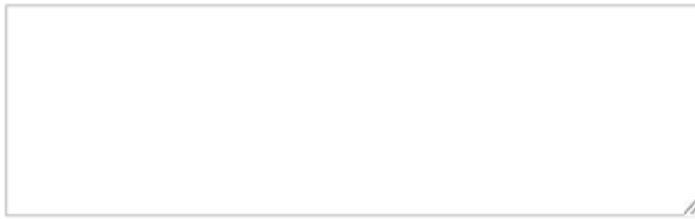
Quais são as abordagens de testes (caixa branca, caixa preta) utilizadas no setor?



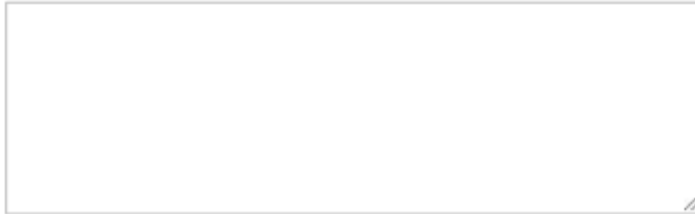
Quais dessas técnicas de teste são utilizadas no setor ?

- Teste de Unidade
- Teste de Integração
- Teste de Sistema

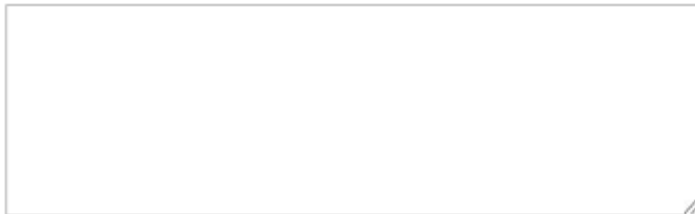
Como os testes de unidade são aplicados ? Descreva como processo é realizado e quais são as técnicas utilizadas




Como os testes de integração são aplicados ? Descreva como processo é realizado e quais são as técnicas utilizadas




Como os testes de sistema são aplicados? Descreva como processo é realizado e quais são as técnicas utilizadas



Descreva as características do ambiente de testes ?



Descreva como é realizada a documentação dos testes (ferramentas e metodologias utilizadas)



Enumere no máximo 5 erros comuns encontrados nos testes (erros de interface, requisitos, codificação, etc...)

Quais são as maiores dificuldades encontradas no processo de testes no setor hoje ?

Qual o seu ponto de vista sobre a automação de testes?

Anexo B – Compilação das respostas obtidas

1. Você possui conhecimentos em programação? Caso sim, em quais linguagens?	
A	“Não”
B	“Sim, pouco de Java, C, C++, HTML e Progress.”
C	“Não tenho conhecimento de programação, mas no momento estou cursando programação Java”
D	“Não possuo”
E	“Não.”
F	“Progress, Flex, Java, Metadados”
2. Qual seu nível de conhecimento na regra de negócio do sistema?	
A	Intermediário
B	Básico
C	Intermediário
D	Intermediário
E	Básico
F	Intermediário
3. Quais as atividades você desempenha no processo de teste? Descreva em itens	
A	<ul style="list-style-type: none"> • Análise • Criação de roteiro a ser testado • Teste do roteiro • Evidência de teste
B	<ul style="list-style-type: none"> • Teste integrado • Teste sistêmico
C	<ul style="list-style-type: none"> • PREPARAR TESTES: após ter a especificação do projeto, deve ser

	<p>realizado o detalhamento dos Casos de Testes, onde deve conter informações necessárias para a execução de um teste e devem ser registrados na planilha de Casos de Testes Reusáveis ou no template de Identificação dos Casos de Testes onde são rastreados os casos de testes reusáveis e detalhado os casos de testes específicos do projeto.</p> <ul style="list-style-type: none"> • TESTE INTEGRADO: O teste integrado é a fase do teste de software em que módulos são combinados e testados em grupo. Ele sucede o teste de unidade e antecede o teste de sistema. Tem como propósito verificar os requisitos funcionais, de desempenho e de confiabilidade do sistema, podendo detectar defeitos entre os componentes do sistema. • TESTE SISTÊMICO: O teste de sistema é uma fase do processo de teste de software em que o sistema já completamente integrado é verificado quanto a seus requisitos num ambiente correspondente o máximo possível ao objetivo final, ou o ambiente de produção
D	<ul style="list-style-type: none"> • Avaliação e entendimento do problema e/ou solicitação do cliente; • Entendimento da resolução do problema e/ou engenharia por parte do analista do chamado; • Efetuar o teste o mais preciso e detalhado possível para identificar se foi realmente corrigido/desenvolvido conforme solicitado pelo cliente.
E	<ul style="list-style-type: none"> • Leitura; • Análise; • Busca de conhecimento sobre o teste a ser realizado; • Variações de testes;
F	<ul style="list-style-type: none"> • Verificar documentação • Verificar e realizar variações a partir do teste unitário • Verificar fontes no TFS • Liberação

4. Descreva em itens as etapas de testes que são realizadas no setor (Planejamento, criação de cenários, etc....)

A	<ul style="list-style-type: none"> • Planejamento • Documentação • Criação de roteiros • Teste • Evidência do teste
B	<ul style="list-style-type: none"> • Planejamento • Preparação do caso de testes • Testes • Evidências • Encerramento •
C	<ul style="list-style-type: none"> • PLANEJAR TESTES: O objetivo desta atividade é dimensionar o esforço e os recursos necessários para realizar as atividades de Testes, definir diretrizes do que será testado e possibilitando um bem-sucedido gerenciamento e condução dos testes realizados no projeto • PREPARAR TESTES: após ter a especificação do projeto, deve ser realizado o detalhamento dos Casos de Testes, onde deve conter informações necessárias para a execução de um teste e devem ser registrados na planilha de Casos de Testes Reusáveis ou no template de Identificação dos Casos de Testes onde são rastreados os casos de testes reusáveis e detalhado os casos de testes específicos do projeto. • TESTE INTEGRADO: O teste integrado é a fase do teste de software em que módulos são combinados e testados em grupo. Ele sucede o teste de unidade e antecede o teste de sistema. Tem como propósito verificar os requisitos funcionais, de desempenho e de

	<p>confiabilidade do sistema, podendo detectar defeitos entre os componentes do sistema.</p> <ul style="list-style-type: none"> • TESTE SISTÊMICO: O teste de sistema é uma fase do processo de teste de software em que o sistema já completamente integrado é verificado quanto a seus requisitos num ambiente correspondente o máximo possível ao objetivo final, ou o ambiente de produção
D	<ul style="list-style-type: none"> • Planejamento • Preparação • Especificação • Execução • Entrega
E	<ul style="list-style-type: none"> • Organização de chamados por vencimento de SLA • Projetos de inovação e manutenção são divididos • Uma equipe focada nos testes de inovação e outra em manutenção
F	<ul style="list-style-type: none"> • Unidade • Integrado • Sistemico
5. Como o plano de testes é definido?	
A	<p>“Aos que atendem a área de inovação, deve ser criado os casos de testes e, estipulado a quantidade de horas e quem vai ser alocado conforme complexidade e conhecimento do teste. ”</p>
B	<ul style="list-style-type: none"> • Itens a serem testados. • As atividades e recursos a serem empregados. • Os tipos de testes a serem realizados e ferramentas empregadas. • Os critérios para avaliar os resultados obtidos
C	<p>“Nesta atividade o Analista de Testes deve:</p> <ul style="list-style-type: none"> • Avaliar com o Líder de Projeto a criação do ambiente (se necessário) e solicitar a equipe de Gestão de Ambientes e Expedição; • Validar os requisitos do projeto dentro do contexto do produto;

	<ul style="list-style-type: none"> • Avaliar o que deve ser automatizado e quais tipos de testes serão automatizados; • Confeccionar o plano de testes. <p>O objetivo do Plano de Teste é registrar o que será testado, como estes testes serão realizados e também documentar os aspectos globais relacionados aos testes. Isto possibilitará uma bem-sucedida coordenação e condução dos testes no projeto.</p> <p>O Plano de Teste deve ser elaborado utilizando o template padrão e sua aprovação deve ser feita via workflow ou via e-mail conforme definido na Matriz de Dados e Comunicação, no prazo máximo de 7 dias úteis. Após homologação este artefato deve ser enviado ao Testador responsável pelo projeto. ”</p>
D	<ul style="list-style-type: none"> • Itens a serem testados: o escopo e objetivos do plano devem ser estabelecidos no início do projeto. • As atividades e recursos a serem empregados: as estratégias de testes e recursos utilizados devem ser definidas, bem como toda e qualquer restrição imposta sobre as atividades e/ou recursos. • Os tipos de testes a serem realizados e ferramentas empregadas: os tipos de testes e a ordem cronológica de sua ocorrência são estabelecidos no plano. • Os critérios para avaliar os resultados obtidos: métricas devem ser definidas para acompanhar os resultados alcançados.
E	“O plano de teste é definido com base no teste de unidade, entretanto, com uma maior variação de possibilidades. ”
F	“É definido um cronograma de atividades e feita a alocação de recurso. ”
6. Como os cenários de testes são definidos?	
A	“Geralmente é buscado junto à analista referência os processos para colocar no roteiro de testes, o qual deve seguir as parametrizações e passar por todas as situações possíveis antes de ser liberado. ”

B	“Com base nos chamados abertos pelos clientes e nos processos onde há mais alterações e mais histórico de erros. ”
C	“É definido juntamente com o analista do projeto, onde em uma reunião de kick-off é reunida analistas que mais entendem do processo a ser inovado, onde os mesmos sugerem o que é importante testar, o que aquela inovação pode afetar no sistema em geral. A partir dai o analista de testes prepara os cenários. ”
D	“Através dos chamados abertos pelos clientes e por módulos com maior índice de erros. ”
E	“São definidos de acordo com os chamados abertos e por módulos com maiores problemas encontrados. ”
F	“Não realizo os cenários. ”

7. Como são definidos os casos de teste?

A	“Estudando a engenharia do projeto, e baseado nas regras de negócio e conhecimento do tester. ”
B	“Com base nos chamados abertos pelos clientes e nos processos onde há mais alterações e mais histórico de erros. ”
C	“Com base na especificação do projeto - Fontes alterados, criados, é escrito de forma detalhada todo o teste que deverá ser realizado. Com base no caso de teste o analista responsável por testar o projeto, segue o detalhamento dos casos. ”
D	“Através da identificação das solicitações de mudança ou nova implementação requisitada pelo cliente, descrevendo um escopo para cada requisito para garantir um fluxo padronizado e com menor margem de erro possível para os testes. ”
E	“São definidos de acordo com os chamados abertos e por módulos com maiores problemas encontrados. ”
F	“São definidos a partir dos testes unitários. ”

8. Quais são as abordagens de testes (caixa branca, caixa preta) utilizadas no setor?

A	<ul style="list-style-type: none"> • Caixa branca: não é utilizado • Caixa preta: atingem quase por completo os testes
B	<ul style="list-style-type: none"> • Caixa preta.
C	<ul style="list-style-type: none"> • Caixa preta: técnicas baseadas em especificação
D	<ul style="list-style-type: none"> • Caixa preta.
E	<ul style="list-style-type: none"> • Caixa preta.
F	“O processo é todo caixa preto. ”

9. Quais dessas técnicas de teste são utilizadas no setor (Teste de unidade, Teste de Integração, Teste de Sistema)?

A	<ul style="list-style-type: none"> • Teste de Unidade • Teste de Integração • Teste de Sistema
B	<ul style="list-style-type: none"> • Teste de Integração • Teste de Sistema
C	<ul style="list-style-type: none"> • Teste de Sistema
D	<ul style="list-style-type: none"> • Teste de Integração • Teste de Sistema
E	<ul style="list-style-type: none"> • Teste de Integração • Teste de Sistema
F	<ul style="list-style-type: none"> • Teste de Integração • Teste de Sistema

10. Como os testes de unidade são aplicados? Descreva como processo é realizado e quais são as técnicas utilizadas

A	“Testado apenas se as funcionalidades estão corretas. ”
B	“Não utilizamos. ”
C	“Processo de teste de unidade está dentro da etapa de Codificação, onde o desenvolvedor, após realizar a alteração /inclusão de fontes, realiza o teste de unidade anexando a evidencia de testes. Entende-se como evidência a descrição breve na tarefa dos testes realizados (podendo ser anexado print screen de telas e logs) ou a geração de um vídeo do desenvolvedor, demonstrando o teste do código. ”
D	“Não faz parte do escopo de testes do setor. ”
E	“Depois da codificação pelo analista o teste de unidade é aplicado pelo programador, a técnica utilizada são: Caixa preta e branca. ”
F	“Testes realizados baseados na funcionalidade. ”

11. Como os testes de integração são aplicados? Descreva como processo é realizado e quais são as técnicas utilizadas

A	“Utilizamos uma tela pronta, onde é testado cada funcionalidade disponível nela, bem como suas variações. ”
B	“Rodamos o programa testando todas as suas funcionalidades, independente se foi alterada somente uma delas, testamos todas a fim de encontrar algum erro. ”
C	
D	Rodamos o sistema a fim de encontrar falhas provenientes da integração interna dos componentes do sistema e descobrir possíveis falhas associadas à interface do sistema.
E	“São aplicados de acordo com o teste de unidade, o processo é realizado através da técnica da caixa preta. ”
F	“São realizados testes com base nos testes unitários, realizando variações

	em cima dos mesmos e verificando se não impacta em outros programas. ”
12. Como os testes de sistema são aplicados? Descreva como processo é realizado e quais são as técnicas utilizadas	
A	“Utilizamos uma tela pronta, onde é testado cada funcionalidade disponível nela, porém, com variações restritas. Efetuamos teste de regressão. ”
B	“Executamos o programa com o ponto de vista do usuário final, com ambiente o mais próximo possível do usado por eles. ”
C	“A cada término de versão, é atualizado o ambiente de testes com os fontes que estão prontos para ser expedido ao cliente. Seguindo calendário de testes é realizado o teste sistêmico, onde é realizado o teste sistêmico de processo, não validando fonte a fonte, e sim a funcionalidade do processo como um todo. ”
D	“Executamos o sistema sob ponto de vista do cliente, varrendo as funcionalidades em busca de falhas em relação aos objetivos originais. Os testes são executados o mais próximo possível da realidade do sistema que o cliente utiliza no seu dia-a-dia.”
E	“Os testes sistêmicos são realizados através de roteiros e por módulos do sistema. E caixa preta é técnica. ”
F	“Os testes Sistêmicos são realizados por toda a equipe de testes, divididos por módulos e é testado todo o sistema na sua última versão. ”
13. Descreva as características do ambiente de testes?	
A	“Ambiente comum com a área de desenvolvimento. ”
B	<ul style="list-style-type: none"> • Versão mais recente do sistema. • Compilação dos fontes ocorre todos os dias durante a madrugada. • Grande número de dados para a realização dos testes.
C	<ul style="list-style-type: none"> • “O ambiente de testes deve estar com todos os fontes atualizados e sem erros de compilação”
D	<ul style="list-style-type: none"> • Versão mais recente do sistema. • Compilação dos fontes ocorre todos os dias durante a madrugada.

	<ul style="list-style-type: none"> • Grande número de dados para a realização dos testes.
E	<ul style="list-style-type: none"> • Lento • Furo de base • Robusto • Instável
F	“Ambiente de testes possui banco Oracle e 4 linguagens de programação diferentes. ”
14. Descreva como é realizada a documentação dos testes (ferramentas e metodologias utilizadas)	
A	“Ferramenta própria com templates definidos pela empresa. ”
B	“É feita através de um documento WORD no qual colocamos as evidências, prints e resumo do que foi feito. ”
C	“O teste deve ser evidenciado com informações: do produto, da versão, do banco de dados em que este componente será testado, da sequência dos testes realizados e dos resultados atingidos. Esta evidência pode ser gerada através do template proposto ou outros meios (vídeo, por exemplo) desde que observados os itens mínimos citados acima. Para complementar as evidências geradas poderão ser anexados print screen de telas e/ou logs. ”
D	“É feita através de um documento Word, no qual colocamos as evidências (prints) e resumo do que foi feito. ”
E	“A documentação dos testes é feita pela evidência de testes, utilizamos como ferramenta o SSIM (Protheus). ”
F	“A documentação de testes é feita através de prints de tela do processo, planilha em Excel com check-list, evidências de testes e realizado White-box, ferramenta que verifica compilação dos programas progress. ”
15. Enumere no máximo 5 erros comuns encontrados nos testes (erros de interface, requisitos, codificação, etc....)	
A	<ul style="list-style-type: none"> • Codificação

	<ul style="list-style-type: none"> • Erro de ambiente • Interface • Tratativa de erros • Documentação
B	<ul style="list-style-type: none"> • Codificação • Teste unitário • Release notes • Módulo da evidência • Erro de compilação
C	<ul style="list-style-type: none"> • Especificação com falta de informações • Problemas com ambiente • Erros de Compilação (erros no fonte) • Documentação que é encaminhada para o cliente com informações muito vagas.
D	<ul style="list-style-type: none"> • Codificação; • Módulo de evidência; • Teste unitário; • Erro compilação; • Release notes;
E	<ul style="list-style-type: none"> • Erro de compilação • Erro de codificação • Erro de interface • Erro de ambiente • Erro de parametrização
F	<ul style="list-style-type: none"> • Codificação • Interface • Falta de documentação • Ambiente • Furo de Base

16. Quais são as maiores dificuldades encontradas no processo de testes no setor

hoje?	
A	<ul style="list-style-type: none"> • Efetuar os testes em ambiente comum com o desenvolvimento • Base de dados com erros ou sem informações necessárias para o teste • Tempo de teste do projeto/chamado restrito • Conhecimento às vezes não suficiente do módulo a ser testado • Ambiente de teste instável (erros compilação, EAR, conexões...)
B	<ul style="list-style-type: none"> • Ambientes não funcionais, com erros de compilação. • Falta de suporte por parte dos analistas.
C	<ul style="list-style-type: none"> • Ambiente funcionando corretamente
D	<ul style="list-style-type: none"> • A funcionalidade do ambiente e a falta de suporte por parte de alguns analistas.
E	<ul style="list-style-type: none"> • Ambiente instável • Prazo para teste curto • Falta de apoio dos analistas
F	<ul style="list-style-type: none"> • Falta de testes unitários, pois muitas vezes não se sabe a condição específica do teste, mesmo com explicação por escrito.
17. Qual o seu ponto de vista sobre a automação de testes?	
A	“Certamente o processo iria trazer mais assertividade e qualidade para o produto, além de otimizar muito o tempo de teste, e eliminar quase que por completo a parte operacional, a qual é mais passível de erros. ”
B	“Muito útil, facilitaria muito a realização dos testes mais simples do sistema, fazendo com que possamos nos focar nos testes mais complexos. ”
C	“Agilidade nos testes e acredito que aumenta a qualidade do produto final. Como são muitos processos de testes a serem realizados, às vezes, não se tem tempo para testar todo o sistema e com a automação acredito que agilize e que desta forma possamos passar por praticamente todos os processos do sistema”

D	“Na minha opinião, a automação de testes não deve ser usada para substituir o teste manual e sim introduzida como uma técnica adicional agregando valor sem invalidar o teste manual. ”
E	“Favorável, tendo em vista a assertividade do teste e principalmente o tempo de realização. Também é uma ótima forma de redução de custo. ”
F	“Ferramenta muito importante, principalmente em testes sistêmicos. ”

Anexo C – Modelo de evidência de testes da empresa alvo



Evidência de Testes

Produto: <i>[obrigatório]</i>	[Informar o produto]
Versão: <i>[obrigatório]</i>	[Informar a versão do produto em que será executado o teste. Exemplo: P10]
Produto/Versão Integrado: <i>[obrigatório para projetos de integração]</i>	[Informar em quais linhas de produtos e versões haverá expedição]
Banco de Dados: <i>[obrigatório]</i>	[Progress, Oracle, SQL, Informix]

1. Evidência e sequência do passo a passo *[obrigatório]*

Teste Unitário (Codificação)

[Informar uma breve descrição dos testes realizados. O Print Screen de telas é opcional]

Teste Integrado (Equipe de Testes)

[Descrever a sequência dos testes realizados em detalhes e os resultados atingidos. O Print Screen de telas é opcional]

Teste Automatizado (Central de Automação) *[Opcional]*

[Informar as suites executadas e descrever os resultados atingidos]

Dicionário de Dados (Codificação) *[Opcional]*

[O objetivo é incluir o print-screen da tela do dicionário de dados atualizado quando necessário.]

2. Outras Evidências *[Opcional]*

O objetivo é indicar para a equipe de Testes que a informação criada deve ser validada, como por exemplo, publicação de ponto de entrada, etc.

Anexo D – Modelo plano de testes empresa alvo

Plano Testes		TOTVS
1.1. Escopo		

Para esse projeto não serão construídos scripts de testes automatizados. Os testes ocorrerão de forma manual

Assinale abaixo os tipos de testes que poderão ser utilizados.

Fase de Análise	Execução	Esforço
() Montagem de Casos de testes	<input checked="" type="checkbox"/> Manual	1hrs
	Observações:	
	<input type="checkbox"/> Automático	
	Observações:	

Testes de Inovação	Execução	Release	Ambiente	Chamado	Esforço
(x) Unidade	<input checked="" type="checkbox"/> Manual	12.1.7	Oracle		4hs
	Observações:				
	<input type="checkbox"/> Automático				
	Observações:				
(x) Integrado	<input type="checkbox"/> Preparar base para Automação				
	Observações:				
	<input type="checkbox"/> Atualização/Criação e Execução dos scripts na Automação				
	Observações:				
	<input checked="" type="checkbox"/> Testes Manuais + Análise da Automação	12.1.7	SQL		7hs
(x) Sistêmico	<input type="checkbox"/> Atualização e Execução dos Scripts de Automação				
	Observações:				
	<input checked="" type="checkbox"/> Testes Manuais + Análise da Automação	12.1.7	Progress		3hs
	Observações:				

1.2. Critérios de Aceitação

Inovação

*Programas disponíveis no ambiente de testes.
Ambiente Fluig e ERP Totvs 11 devidamente parametrizados para integração.*

Réplica

*Programas disponíveis no ambiente de testes.
Ambiente Fluig e ERP Totvs 11 devidamente parametrizados para integração.*

1.3. Critérios de Conclusão

Inovação

*Ser possível por meio da Ferramenta Fluig realizar a Auditoria de Contas e ao final do processo liberar os movimentos da base do ERP.
Ser possível por meio da Ferramenta Fluig realizar a inclusão do Prestador no ERP Totvs*

Réplica

*Ser possível por meio da Ferramenta Fluig realizar a Auditoria de Contas e ao final do processo liberar os movimentos da base do ERP.
Ser possível por meio da Ferramenta Fluig realizar a inclusão do Prestador no ERP Totvs*

Anexo E – Exemplo de cenários/casos de testes

SCRIPTS/CENÁRIOS DE TESTE	AUTOMATIZAÇÃO	DESCRIÇÃO DO TESTE	RESULTADOS	COMPLEXIDADE	PRIORIDADE
1	Não	Verificar o layout da tela principal dos cadastros: - A tela principal deverá conter a régua padrão dos cadastros do sistema, contendo os botões Primeiro, Anterior, Próximo, Último, Código, Inclui, Modifica, Elimina, Prorrogar, Fechar, Função, Relatório, Sair e Ajuda.	Tela principal com todos os campos e funções sendo apresentados corretamente	Baixa	
2	Não	Verificar o funcionamento da navegação do CRUD (primeiro, anterior, próximo, último) Acessar a aba Procedimento Teste 001	Funcionalidade do CRUD conforme TESTE 001 descrito	Baixa	
3	Não	Verificar o funcionamento da opção "Código" Acessar a aba Procedimento Teste 002	Funcionalidade do Código conforme TESTE 002 descrito	Baixa	
4	Não	Verificar o funcionamento da opção Inclui Acessar a aba Procedimento Teste 003	Funcionalidade do Inclui conforme TESTE 003 descrito	Baixa	
5	Não	Verificar o funcionamento da opção Modifica Acessar a aba Procedimento Teste 004	Funcionalidade do Modifica conforme TESTE 004 descrito	Baixa	
6	Não	Verificar o funcionamento da opção Elimina Acessar a aba Procedimento Teste 005	Funcionalidade do Elimina conforme TESTE 005 descrito	Baixa	

Anexo F – Exemplo de detalhamento de um caso de teste

INFORMAÇÕES SOBRE O PROCEDIMENTO	
OBJETIVO	Descrever o funcionamento da opção Código
PASSOS	
1	Clicar na opção código .
2	Deve ser informado um código existente , desta forma , deve ser possível visualizar todos os campos em tela , sendo : "Area de Atuação"; "Descrição"; "Data Limite"; "Bloq.Export PTU A400"; "Guia Médico"; "Usuário"; "Atualização".
3	Deve ser informado um código de area de atuação inexistente , desta forma , deve ser apresentado
4	Na opção código não deve ser permitido alterar nenhum campo do cadastro.
5	
6	
7	

Anexo G – Exemplo roteiro de teste

CAMINHO	DESCRIÇÃO	Variações	PRÉ REQUISITO	RESULTADO
Importar ABI'S		Importação do arquivo.		Importação do Arquivo
Importar ABI'S ---> Importar Atendimento		Importação dos atendimentos.		Importação dos atendimentos dentro do processo de Importação do Arquivo
Selecionar ABI; clicar sobre "analisar atendimentos" selecionar atendimento na lista; clicar sobre "declarações"; Selecione o tipo de declaração a ser gerada "beneficiário ou operadora"; Informar dados (obs: campos não obrigatórios); clicar sobre "gerar"; Verificar o PDF gerado;		Geração das declarações.		Declaração gerada com sucesso em PDF com informações corretas..
		Edição da Situação da AIH		Edição da Situação da AIH
		Edição Justificativa na AIH		Edição da Justificativa na AIH
		Edição Motivos na AIH		Edição do Motivo na AIH
		Edição Número de Ofício (na tela das ABI's)		Edição Número de Ofício
		Percentual		Permitir cadastrar e alterar percentual
		Cadastro de Motivos e Justificativas		No cadastro de Motivos e Justificativas testar consistência questionando se deseja confirmar a ação de
		Ordem dos Motivos		No cadastro de Motivos, arrastar com o mouse os motivos e verificar se está
		Geração da Impugnação		Conferir se os dados são impressos corretamente no
		Geração do Recurso		Conferir se os dados são impressos corretamente no formulário e na justificativa:sem rodapé,

Anexo H – Relatório SyncProgress Fontes para Compilação

15/11/2016 19:17:49 - Sincronizacao iniciada. Aguarde...
15/11/2016 19:18:05 - afp\af0800a.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:05 - afp\af0800b.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:05 - api\api-administrativeintegration.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-fp0006.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:06 - api\api-gps-ws.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:06 - api\api-recebe-ptu60.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-replica-benef.p teve a data de ultima atualizacao alterada (19/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-solic-sadt-tiss3.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-solicita-ptu60.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-solicitafoundation-aud.p teve a data de ultima atualizacao alterada (31/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-transf-benef.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-usuario.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:06 - api\api-yp0007.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atbo\boatauditory.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atbo\boquotationopme.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:07 - atbo\botisslotegua.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atp\at0110c5.p teve a data de ultima atualizacao alterada (19/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atp\at0110r.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atp\at0112c.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:07 - atp\at1000.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:08 - atp\atapi021.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:08 - atp\atapi023-ptu60.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:08 - bosau\bosauproposals.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:08 - bosau\bosauvalorizationrules.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:08 - bosau\global\bosauattachment.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:08 - bosau\global\bosauattachmentconfigurations.p teve a data de ultima atualizacao alterada (19/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:08 - bosau\hcg\bosauassociativeprocessattachments.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:09 - bosau\hfp\bosaubillet.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:09 - bosau\hfp\bosaubilletlayout.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:09 - bosau\hrc\bosaudocuments.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:09 - bosau\hrc\bosau Movements.p teve a data de ultima atualizacao alterada (09/11/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:09 - bosau\hrc\bosaupooler.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:09 - bosau\hvp\bosaubeneficiarytransference.p teve a data de ultima atualizacao alterada (30/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:09 - bosau\hvp\bosauhealthcondvsprocassoc.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:09 - bosau\hvp\bosauhvpglobal.p nao possui .r equivalente no diretorio de compilados. Sera compilado.
15/11/2016 19:18:10 - cgp\cg0111t.p teve a data de ultima atualizacao alterada (19/10/2016 para 15/11/2016). Sera compilado.
15/11/2016 19:18:10 - csw\cs0111u.p teve a data de ultima atualizacao alterada (19/10/2016 para 15/11/2016). Sera compilado.

Anexo I – Relatório SyncProgress Acompanhamento da Compilação

15/11/2016 19:18:33 - atp\unix-at0110c.f sera recompilado pois declara atp/at0110c1.f
15/11/2016 19:18:33 - Propath da compilacao: F:\fontes\workspace\GPS-FONTES-PROG\F:\fontes\workspace\GPS-FONTES-PROG\F:\fontes\workspace\EMS5-GP\F:\fontes\works
15/11/2016 19:18:33 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\bosau\hcg\bosauassociativeProcessAttachments.p. Salvar .r em: C:\compilados\bosau\hcg\Situa
15/11/2016 19:18:33 - ** Unknown or ambiguous table gerencia-proces-anexo. (725)
15/11/2016 19:18:33 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\bosau\hcg\bosauassociativeprocessattachments.i Nao entendi a linha 2. (196)
15/11/2016 19:18:33 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\acp\ac2026d.p. Salvar .r em: C:\compilados\acp\Situa#Eo programa: 0
15/11/2016 19:18:42 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\afp\af0120a.p. Salvar .r em: C:\compilados\afp\Situa#Eo programa: 0
15/11/2016 19:18:45 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800a.p. Salvar .r em: C:\compilados\afp\Situa#Eo programa: 0
15/11/2016 19:18:48 - ** Unknown or ambiguous table dwf-emit. (725)
15/11/2016 19:18:48 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800a.p Nao entendi a linha 1711. (196)
15/11/2016 19:18:48 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800b.p. Salvar .r em: C:\compilados\afp\Situa#Eo programa: 0
15/11/2016 19:18:49 - ** Unknown or ambiguous table dwf-estab. (725)
15/11/2016 19:18:49 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\afp\af0800b.p Nao entendi a linha 649. (196)
15/11/2016 19:18:49 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-administrativeintegration.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:18:56 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-fp0006.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:18:56 - ** Unknown or ambiguous table para-fat. (725)
15/11/2016 19:18:56 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\api\api-fp0006.p Nao entendi a linha 78. (196)
15/11/2016 19:18:56 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-gps-ws.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:18:57 - ** Unknown or ambiguous table gerencia-proces-anexo. (725)
15/11/2016 19:18:57 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\api\api-gps-ws.p Nao entendi a linha 1256. (196)
15/11/2016 19:18:57 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-replica-benef.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:02 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-solic-sadt-tiss3.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:08 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-solicitafoundation-aud.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:13 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-transf-benef.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:25 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-usuario.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:35 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\api\api-vp0007.p. Salvar .r em: C:\compilados\api\Situa#Eo programa: 0
15/11/2016 19:19:40 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\boatauditory.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:19:56 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\boinsuranceuserdetail.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:20:02 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\boprotocolmanager.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:20:09 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\boquotationopme.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:20:10 - ** Unknown or ambiguous table param-global. (725)
15/11/2016 19:20:10 - ** ** F:\fontes\workspace\GPS-FONTES-PROG\atbo\boquotationopme.p Nao entendi a linha 241. (196)
15/11/2016 19:20:10 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\bo recursogloportal.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:20:17 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atbo\botisslotegua.p. Salvar .r em: C:\compilados\atbo\Situa#Eo programa: 0
15/11/2016 19:20:28 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atp\at0110b.p. Salvar .r em: C:\compilados\atp\Situa#Eo programa: 0
15/11/2016 19:20:32 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atp\at0110c.p. Salvar .r em: C:\compilados\atp\Situa#Eo programa: 0
15/11/2016 19:20:35 - Compilando: F:\fontes\workspace\GPS-FONTES-PROG\atp\at0110c4.p. Salvar .r em: C:\compilados\atp\Situa#Eo programa: 0

Anexo J– Relatório de Atualização de Diretórios

Saída do console

Ignorando 152.154 KB.. [Ver log completo](#)

	JV01\mateus.leal	JVD0002910	
TSLZYL		JV01\juliana.sacht	JVD007773
PRJ046_DIFAL_Decreto_46930_2015		JV01\rafael.leithold	JVD0003333
DIFAL - Decreto nº 46.930/2015			
100-BI-e506p10Apw-Toninhas		JV01\tech-compilador	TONINHAS
IPIRANGA_docs_12110		JV01\tech-compilador	IPIRANGA
TUDP40		JV01\jefferson.ramos	JVD0000533
TUDP40			
Situação :Tela de visualização da narrativa não permite visualizar a narrativa por completo.			
Solução : Corrigido a forma que a narrativa é apresentada.			
TUDQVF	JV01\mateus.leal		JVD0002910
Autorizador-12.1.10-TS-NERS	JV01\felipe.battistotti		SEVEN91
TUFQJT-11SUST	JV01\alexandre.dalabona		JVD003603
TSIUWF	JV01\cleane		JVD009378
Piata-12.1.10-CRM-Oracle-Windows	JV01\tech-compilador		PIATA
TSISZA	JV01\sean.pablo		JVD003337
JVD003757	JV01\yuri.cavalcante		JVD003757
TSOJTV	JV01\laura.lemos		JVD0003595
Avare-11.5.X-C.R.M-Xref	JV01\tech-compilador		AVARE
TUHOJC	JV01\geovani.figueira		JVD003004
12.1.10-CashFlow-Java-Tartaruga	JV01\tech-compilador		tartaruga.jv01.local
TSOBZP	JV01\michelle.ramos		JVD002672
TQSOVT	JV01\bruno.rocco		SEVEN29
TUOXF9	JV01\ricardo.ferreira		JVD0000496
TUMR01	JV01\maykon.rodrigues		JVD003075
TSOLR8	JV01\michele.pacheco		JVD001229
TUNZH5	JV01\geovani.figueira		JVD003004

Anexo L– Script Utilitário utilizado na Integração

```

Scripts com as targets basicas para as chamadas dos scripts de automação
-->

<project name="AutomationBase">

<taskdef resource="net/sf/antcontrib/antlib.xml"/>

  <!-- Arquivo de saída do log  execução do caso de teste -->
  <property name="exec.output" value="${env.AUTOMACAO_HOME}\tools\error\exec-output.txt"/>

  <!-- Arquivo de saída do log  execução do log da gravação do caso de teste -->
  <property name="exec.db.output" value="${env.AUTOMACAO_HOME}\tools\error\exec-db-output.txt"/>

  <!-- Arquivo de saída de erros de execução do projeto como um todo -->
  <property name="error.output" value="${env.AUTOMACAO_HOME}\tools\error\error-output.txt"/>

  <!-- Arquivo de saída para armazenar o numero do ultimo plano de teste criado -->
  <property name="plano.output" value="${env.AUTOMACAO_HOME}\tools\error\plano-output.properties"/>

  <!-- Mensagem padrão para builds que falham -->
  <property name="msg.falha" value="ERRO - FALHA AO EXECUTAR O CASO DE TESTE"/>

  <!-- Mensagem padrao para buids com sucesso -->
  <property name="msg.sucesso" value="SUCESSO AO EXECUTAR O CASO DE TESTE"/>

  <!-- Mensagem padrão para builds que falham -->
  <property name="msg.db.falha" value="ERRO - FALHA AO PERSISTIR DADOS SOBRE O PLANO DE TESTE"/>

  <!-- Mensagem padrao para buids com sucesso -->
  <property name="msg.db.sucesso" value="DADOS DO PLANO DE TESTE PERSISTIDOS COM SUCESSO"/>

  <!-- Verifica se o build falhou baseado no arquivo de execução criado -->
  <target name="verificaErros">

    <!-- verifica se existem saidas de erro no arquivo do projeto -->
    <property name="file" value="${error.output}"/>
    <resourcecount property="file.lines">
      <tokens>
        <concat>
          <filterchain>
            <tokenfilter>
              <linetokenizer/>
            </tokenfilter>
          </filterchain>
          <fileset file="${file}"/>
        </concat>
      </tokens>
    </resourcecount>

    <!-- Se existirem saidas de erro projeto falha -->
    <fail>
      <condition>
        <not>
          <equals arg1="${file.lines}" arg2="0"/>
        </not>
      </condition>
    </fail>

  </target>

```

```

<target name="verificaRetornoBanco">

    <loadfile srcfile="${exec.db.output}" property="task.db.result" />

    <echo message="{task.db.result}"/>

    <if>
        <equals arg1="{task.db.result}" arg2="0" />
        <then>
            <echo message="{msg.db.sucesso}{line.separator}"/>
        </then>
        <else>
            <echo message="{msg.db.falha}{line.separator}"/>

            <antcall target="mostraLogs">
                <param name="log.exec" value="{exec.db.output}"/>
            </antcall>

        </else>
    </if>

    <fail>
        <condition>
            <not>
                <equals arg1="{task.db.result}" arg2="0" />
            </not>
        </condition>
    </fail>

</target>

<target name="verificaRetornoComFalha">

    <loadfile srcfile="{exec.output}" property="task.output" />

    <if>
        <equals arg1="{task.result}" arg2="0" />
        <then>
            <echo message="{msg.sucesso}{line.separator}"/>
        </then>
        <else>
            <echo message="{msg.falha}{line.separator}"/>

            <antcall target="gravaCasoTeste">
                <param name="statusCasoTeste" value="1"/>
            </antcall>

            <antcall target="mostraLogs">
                <param name="log.exec" value="{exec.output}"/>
            </antcall>
        </else>
    </if>

    <fail>
        <condition>
            <not>
                <equals arg1="{task.result}" arg2="0" />
            </not>
        </condition>
    </fail>

</target>

```

```

<!-- Limpa os logs de erros -->
<target name="limpa_logs">
  <delete file="${error.output}"/>
  <delete file="${exec.output}"/>
  <delete file="${exec.db.output}"/>
</target>

<target name="gravaCasoTeste">
  <exec executable="cmd" output="${exec.db.output}" resultproperty="task.db.result">
    <arg value="/c"/>
    <arg value="java -jar ${env.AUTOMACAD_HOME}/banco/casoTeste.jar ${nomeCasoTeste} ${statusCasoTeste}"/>
    <arg value="-p"/>
  </exec>

  <antcall target="verificaRetornoBanco"/>
</target>

<target name="mostraLogs">
  <echo>Log Execução ${log.exec}</echo>
  <exec executable="cmd">
    <arg value="/c"/>
    <arg value="type ${log.exec}"/>
  </exec>
</target>

<target name="criaDiretorioLog">
  <echo message="Plano de teste: ${plano.number}" />
  <mkdir dir="${diretorio.log.caso}"/>
</target>

<target name="limpaLogsExecRanorex">
  <delete>
    <fileset dir="." includes="**/*.jpg **/*.png **/*.xsl **/*.css **/*.data **/*.rxlog"/>
  </delete>
</target>

</project>

```